

6.3 Paralleles Suchen und optimales Mischen

6.3 Paralleles Suchen und optimales Mischen

parallele Algorithmen

6.3 Paralleles Suchen und optimales Mischen

parallele Algorithmen

Darstellung als „Pseudocode“

6.3 Paralleles Suchen und optimales Mischen

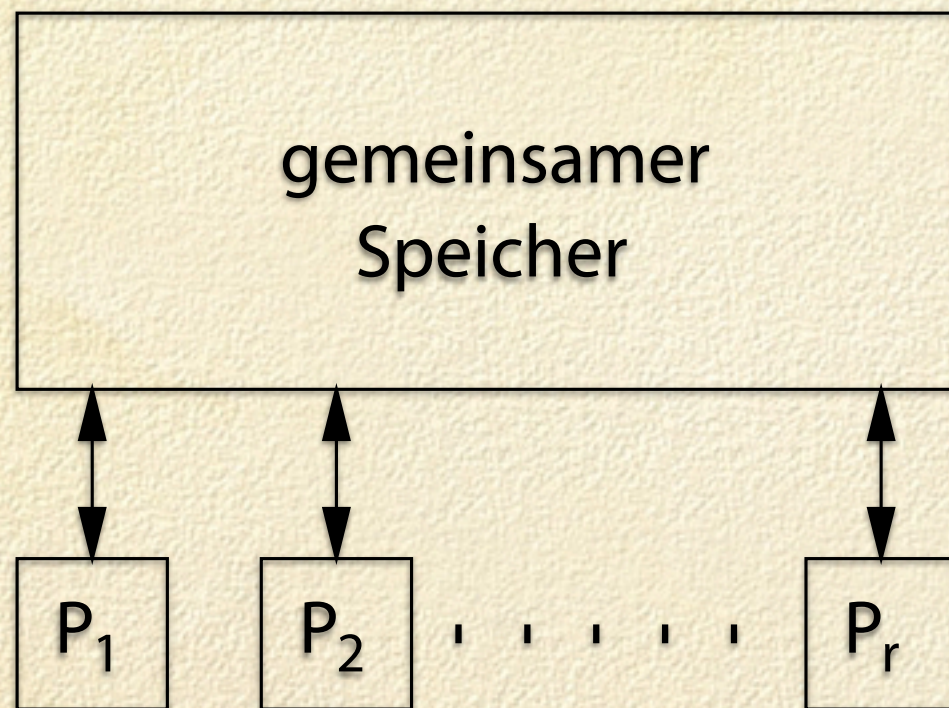
parallele Algorithmen

Darstellung als „Pseudocode“

*übersetzbar in PRAM-Kode
(in linearer Zeit)*

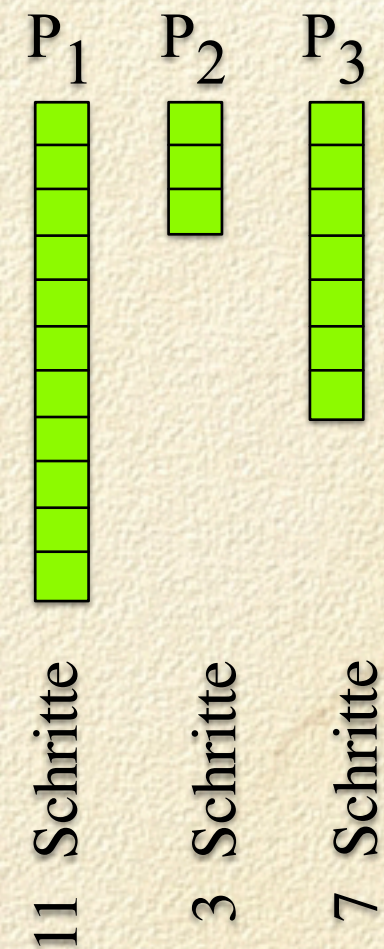
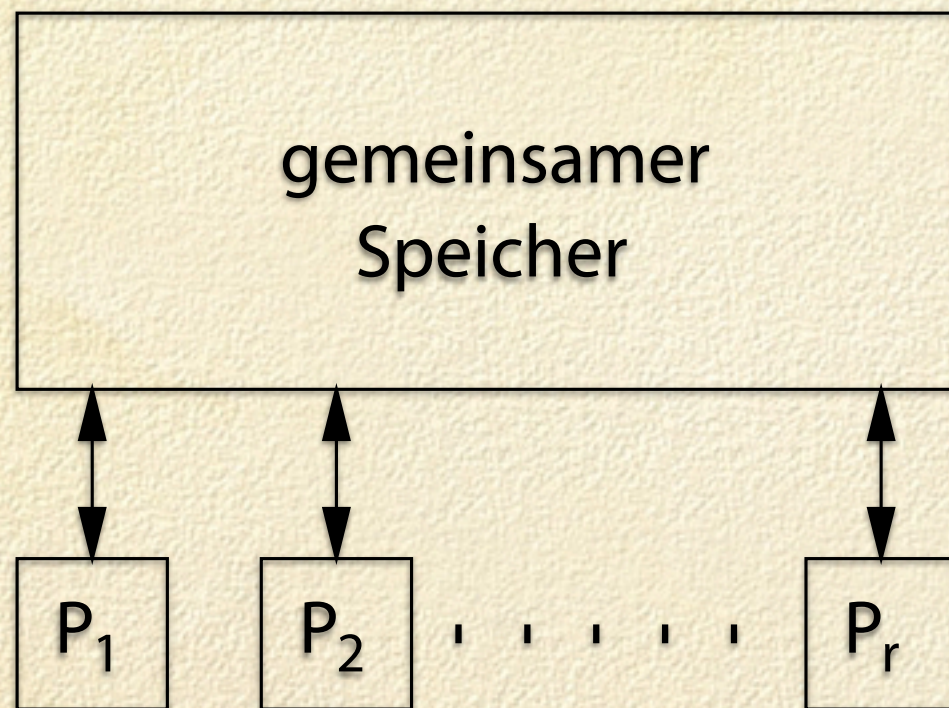
Komplexitätsmaße

- *Operationenkomplexität*: maximale Anzahl $W_A(n)$ der Operationen aller Prozessoren bis zur Termination bei Probleminstanzen der Größe n (d.h. parallele Operationen werden einzeln gezählt, “W” für “work”).



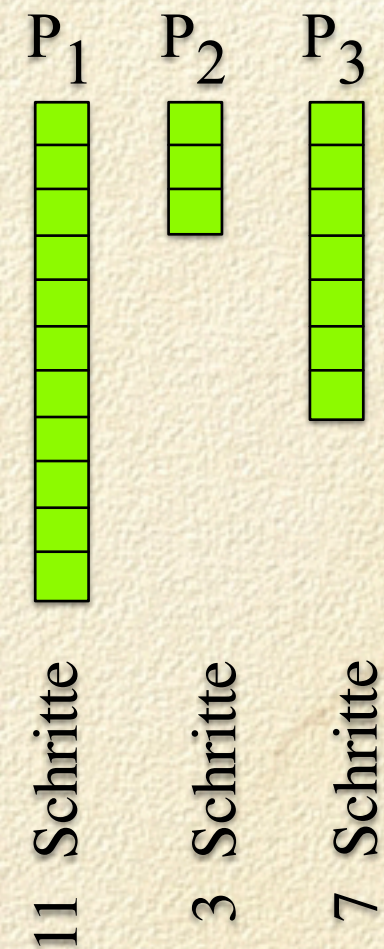
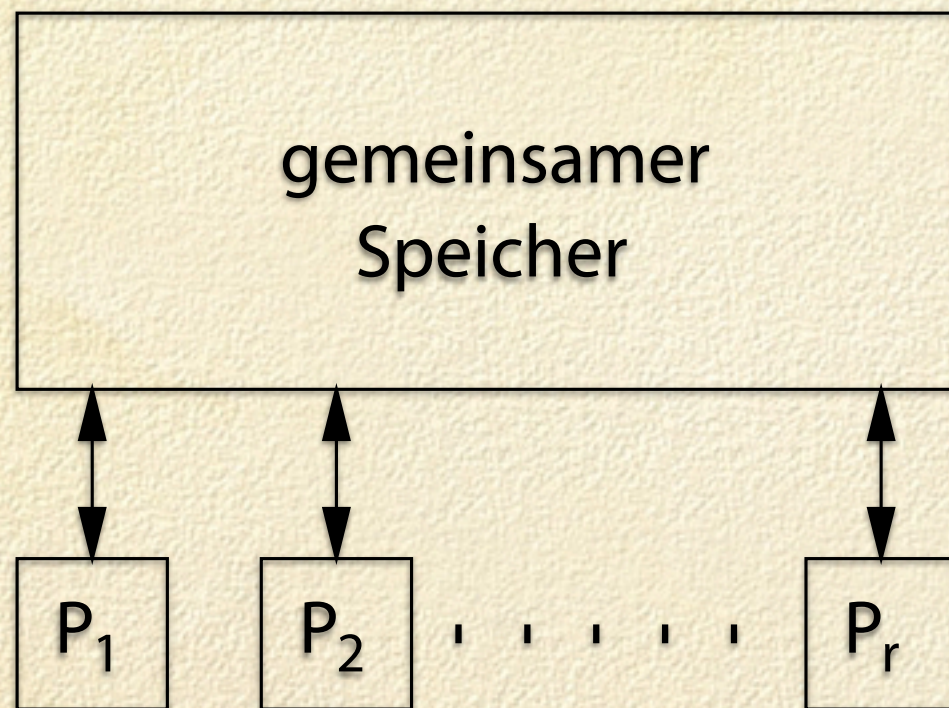
Komplexitätsmaße

- *Operationenkomplexität*: maximale Anzahl $W_A(n)$ der Operationen aller Prozessoren bis zur Termination bei Probleminstanzen der Größe n (d.h. parallele Operationen werden einzeln gezählt, “W” für “work”).



Komplexitätsmaße

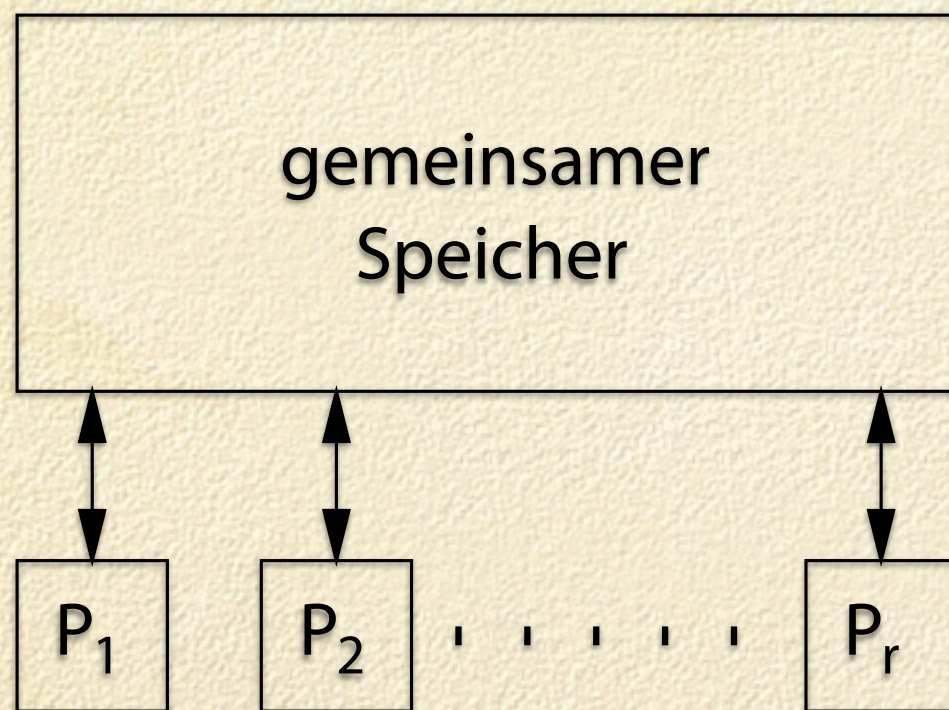
- *Operationenkomplexität*: maximale Anzahl $W_A(n)$ der Operationen aller Prozessoren bis zur Termination bei Probleminstanzen der Größe n (d.h. parallele Operationen werden einzeln gezählt, “W” für “work”).



$$P_A(n) = 3$$

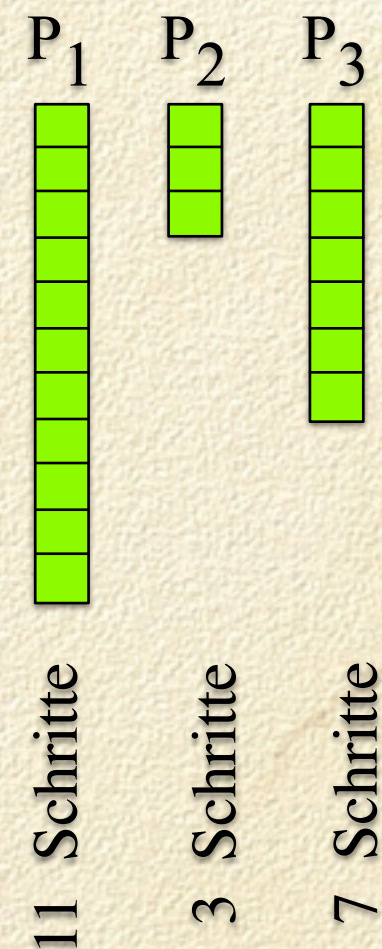
Komplexitätsmaße

- *Operationenkomplexität*: maximale Anzahl $W_A(n)$ der Operationen aller Prozessoren bis zur Termination bei Probleminstanzen der Größe n (d.h. parallele Operationen werden einzeln gezählt, “W” für “work”).



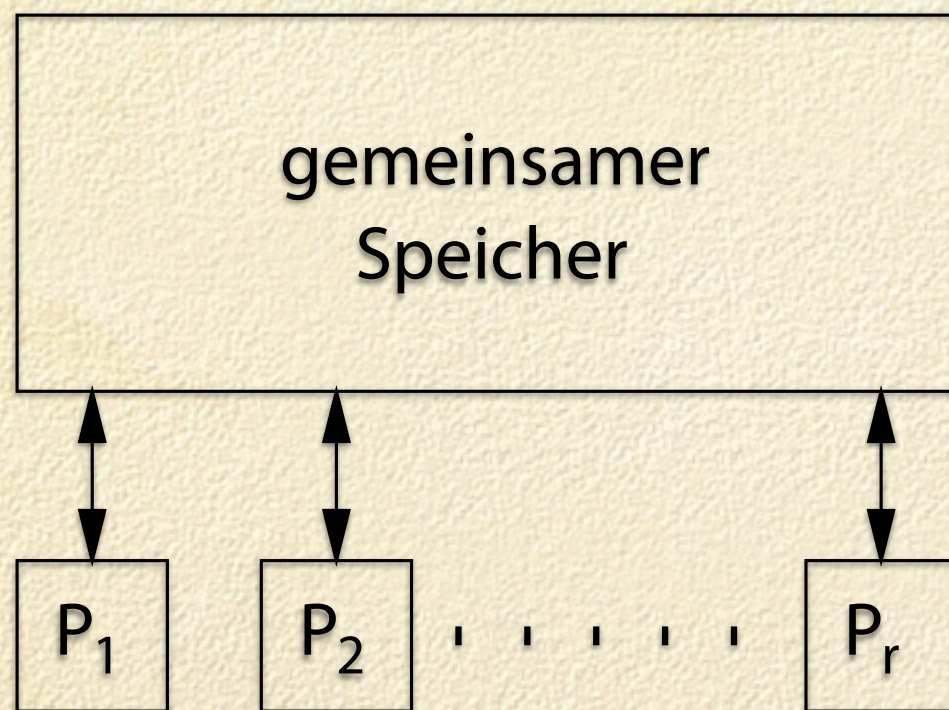
$$P_A(n) = 3$$

$$T_A(n) = 11$$



Komplexitätsmaße

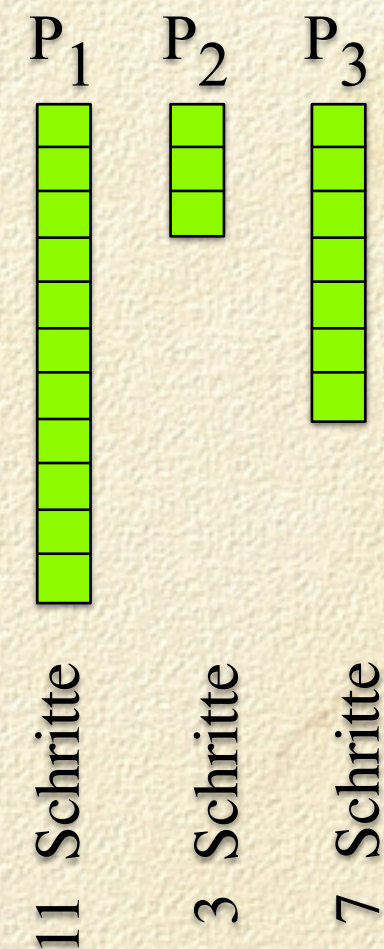
- *Operationenkomplexität*: maximale Anzahl $W_A(n)$ der Operationen aller Prozessoren bis zur Termination bei Probleminstanzen der Größe n (d.h. parallele Operationen werden einzeln gezählt, “W” für “work”).



$$P_A(n) = 3$$

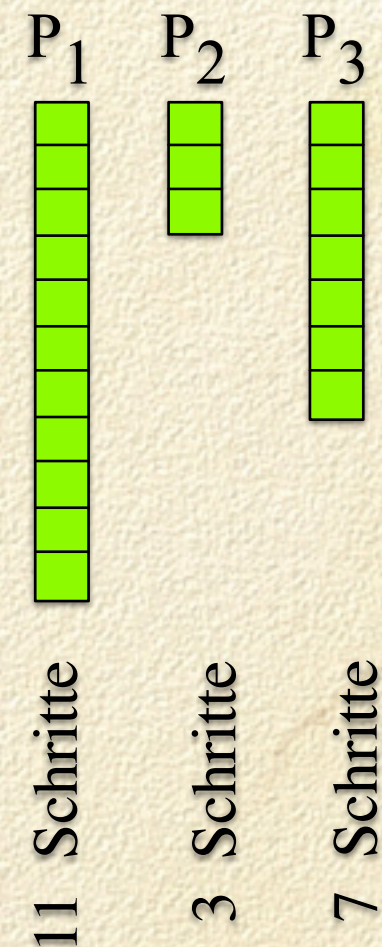
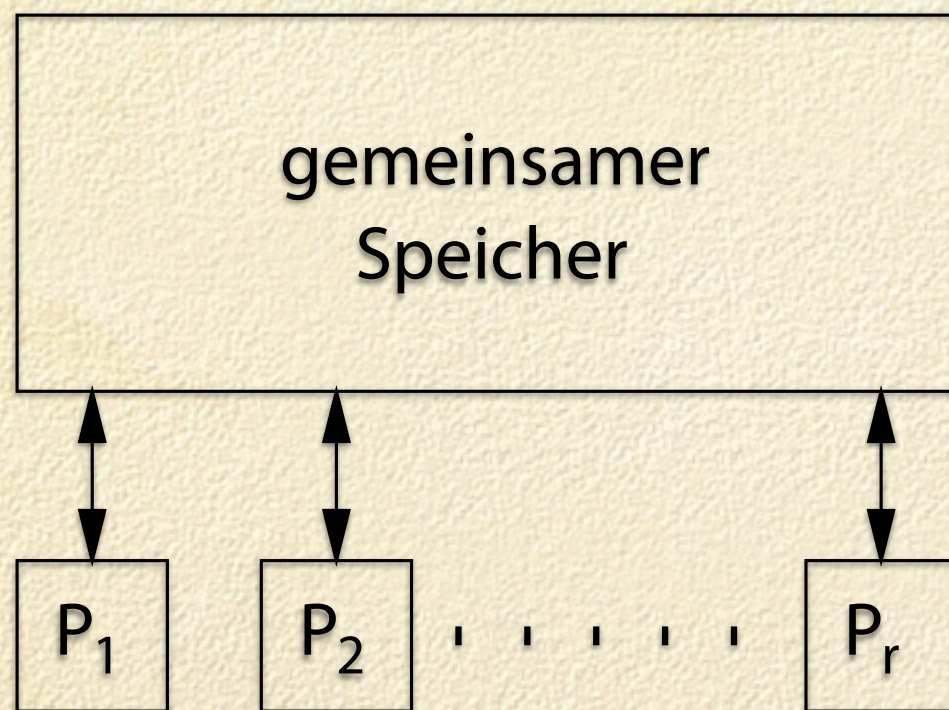
$$T_A(n) = 11$$

$$W_A(n) = 21$$



Komplexitätsmaße

- *Operationenkomplexität*: maximale Anzahl $W_A(n)$ der Operationen aller Prozessoren bis zur Termination bei Probleminstanzen der Größe n (d.h. parallele Operationen werden einzeln gezählt, “W” für “work”).



$$P_A(n) = 3$$

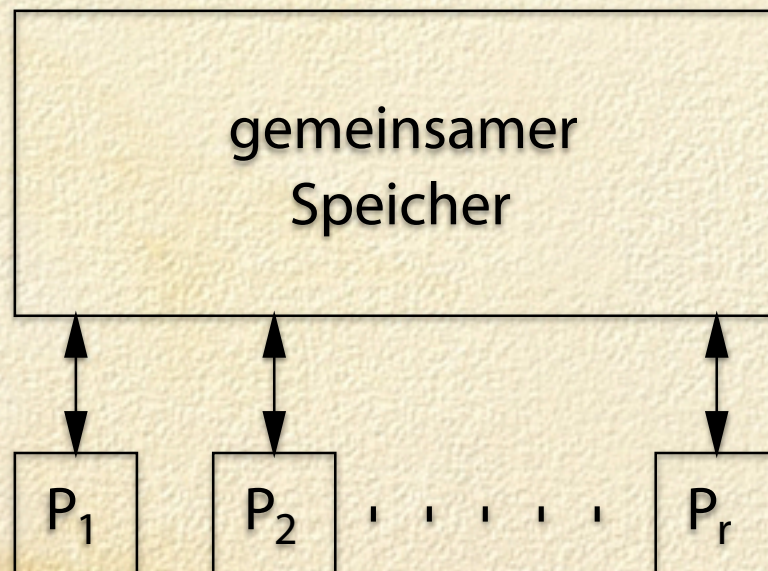
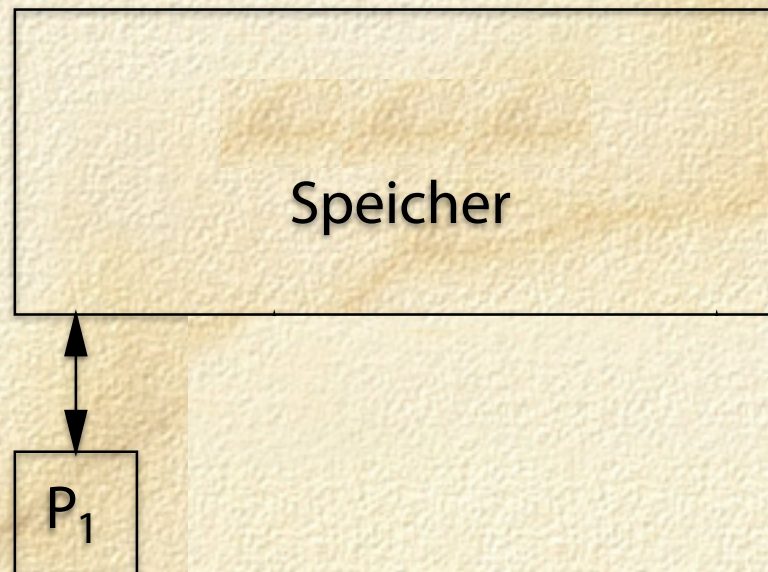
$$T_A(n) = 11$$

$$W_A(n) = 21$$

oft: $W_A(n) \approx P_A(n) \times T_A(n)$

optimaler paralleler Algorithmus

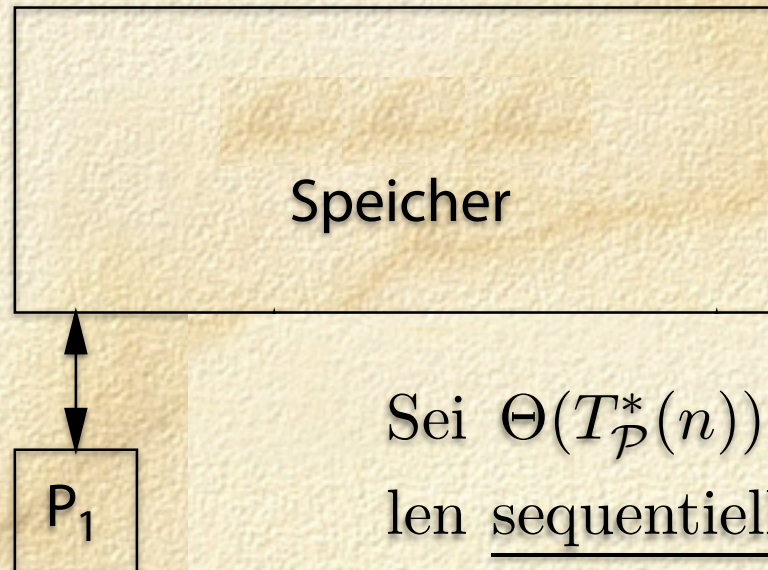
*sequentieller Alg.
mit $T(n)$*



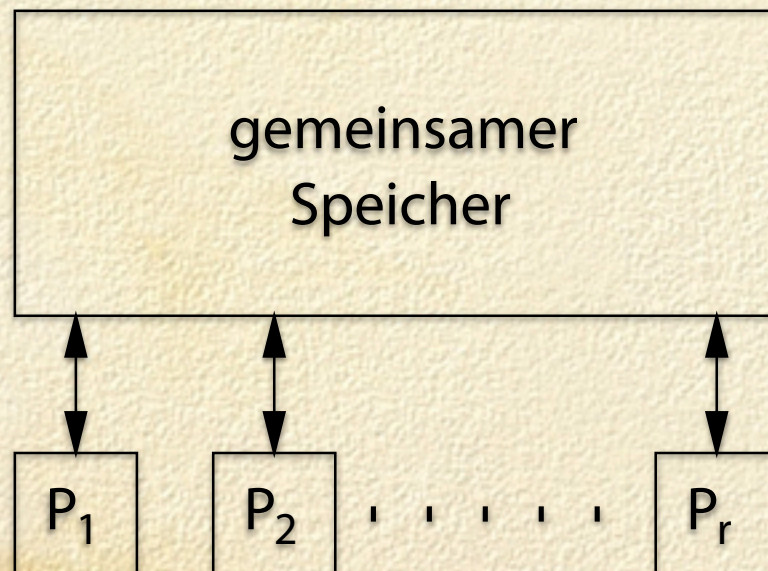
*paralleler Alg.
 $T(n) = ?$*

optimaler paralleler Algorithmus

*sequentieller Alg.
mit $T(n)$*



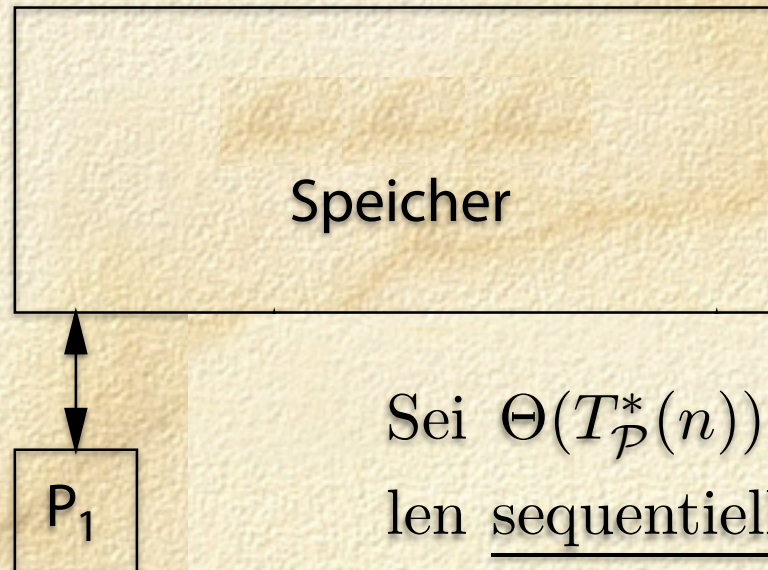
Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen



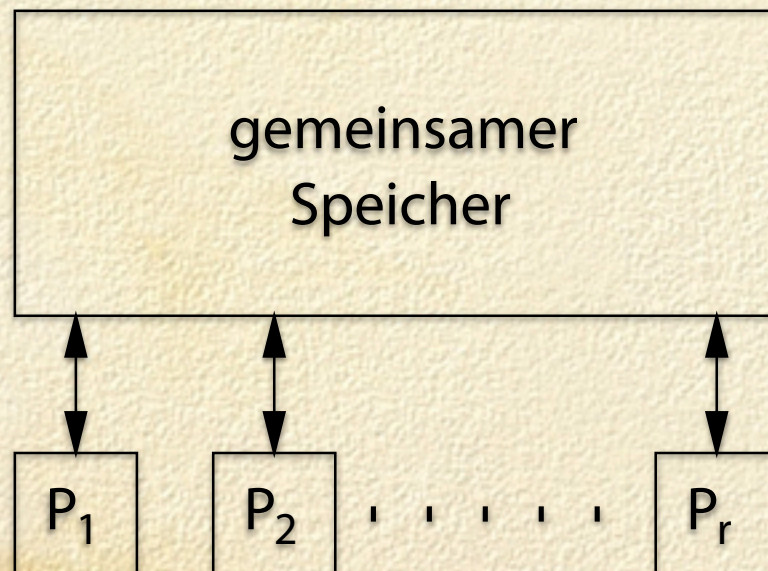
*paralleler Alg.
 $T(n) = ?$*

optimaler paralleler Algorithmus

*sequentieller Alg.
mit $T(n)$*



Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen

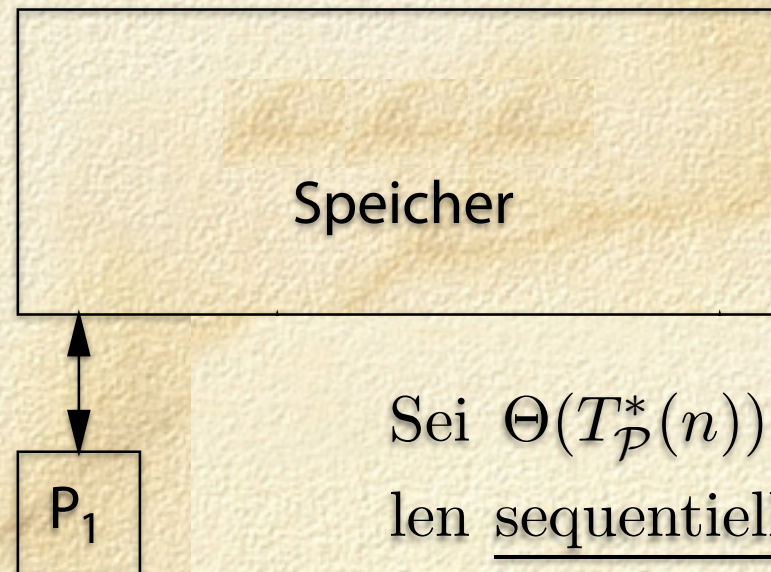


paralleler Alg.

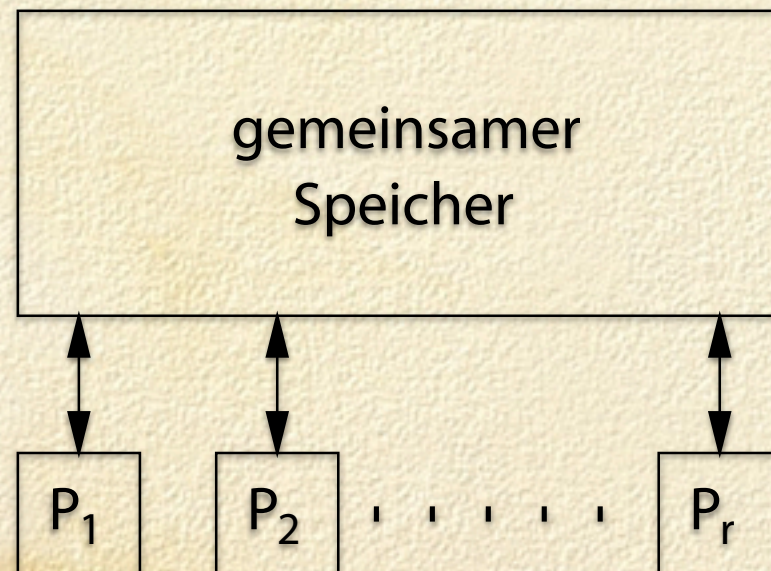
$T(n) = ?$ *besser!*

optimaler paralleler Algorithmus

*sequentieller Alg.
mit $T(n)$*



Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen



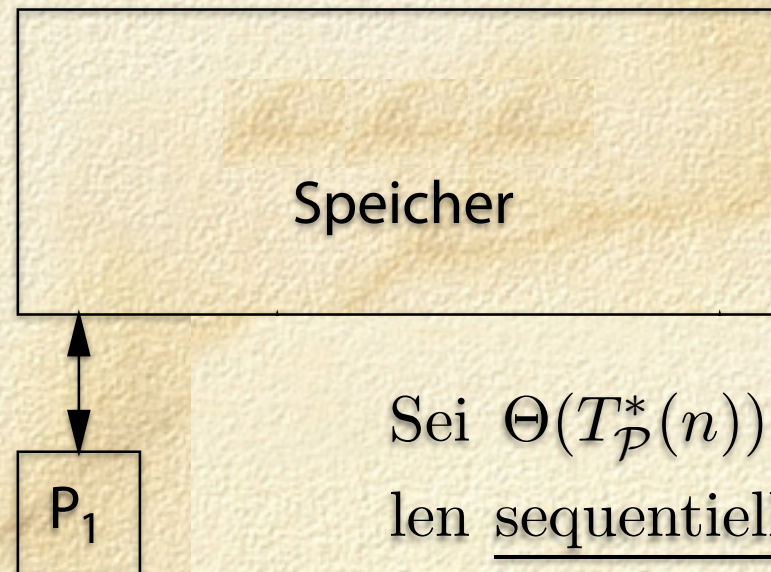
paralleler Alg.

$T(n) = ?$ *besser!*

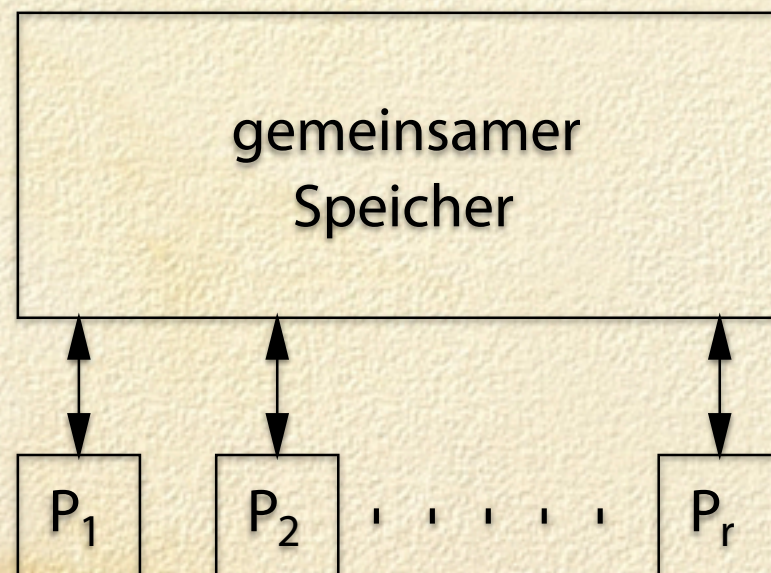
optimal ???

optimaler paralleler Algorithmus

*sequentieller Alg.
mit $T(n)$*



Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen



paralleler Alg.

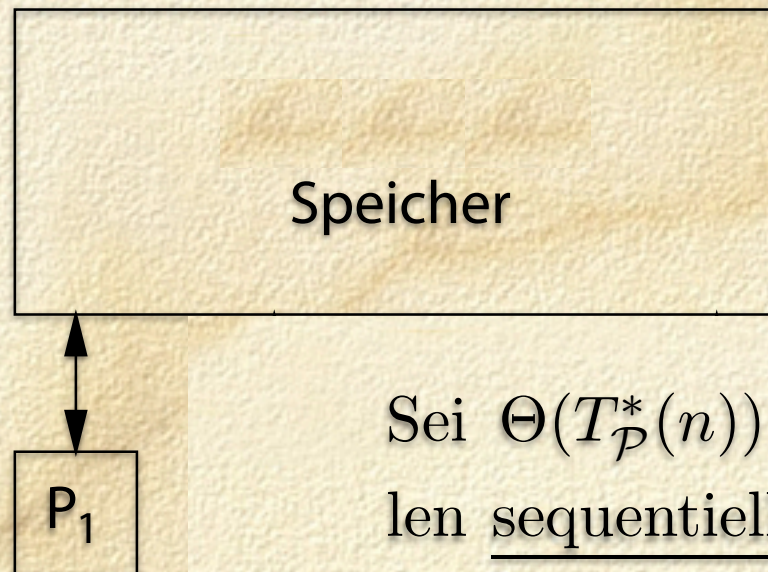
$T(n) = ?$ *besser!*

optimal ???

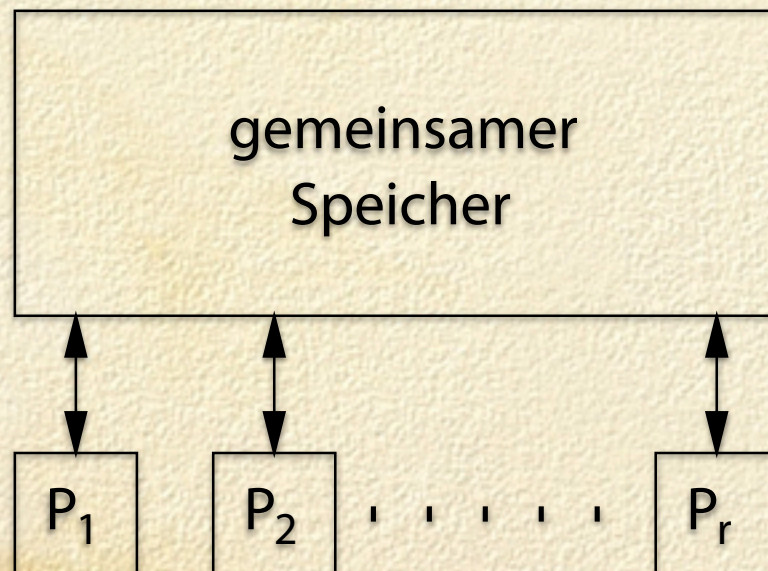
Der Alg. heißt *optimal*, falls

optimaler paralleler Algorithmus

*sequentieller Alg.
mit $T(n)$*



Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen



paralleler Alg.

$T(n) = ?$ *besser!*

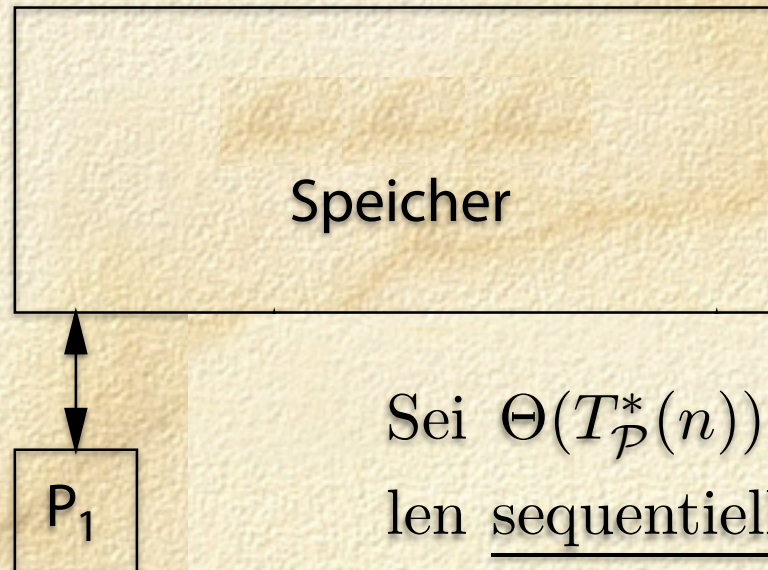
optimal ???

Der Alg. heißt *optimal*, falls

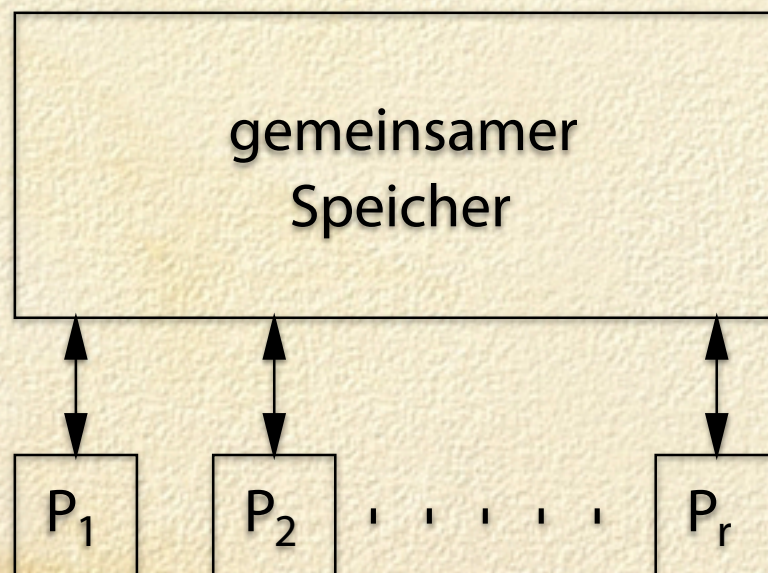
$$W_A(n) = \Theta(T_{\mathcal{P}}^*(n))$$

optimaler paralleler Algorithmus

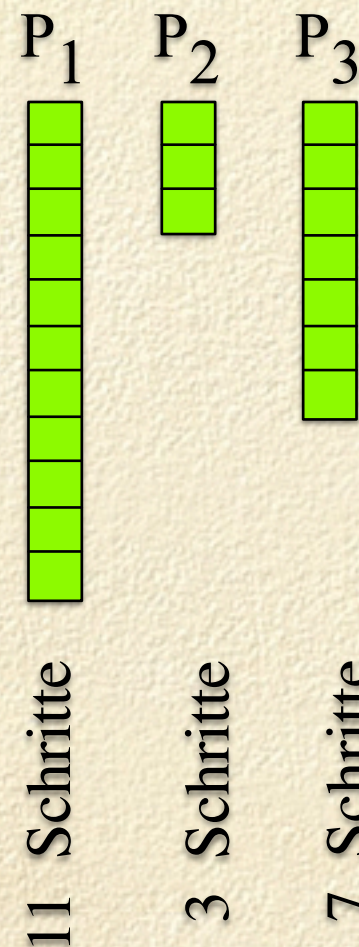
sequentieller Alg.
mit $T(n)$



Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen



$$W_A(n) = 21$$



paralleler Alg.

$T(n) = ?$ *besser!*

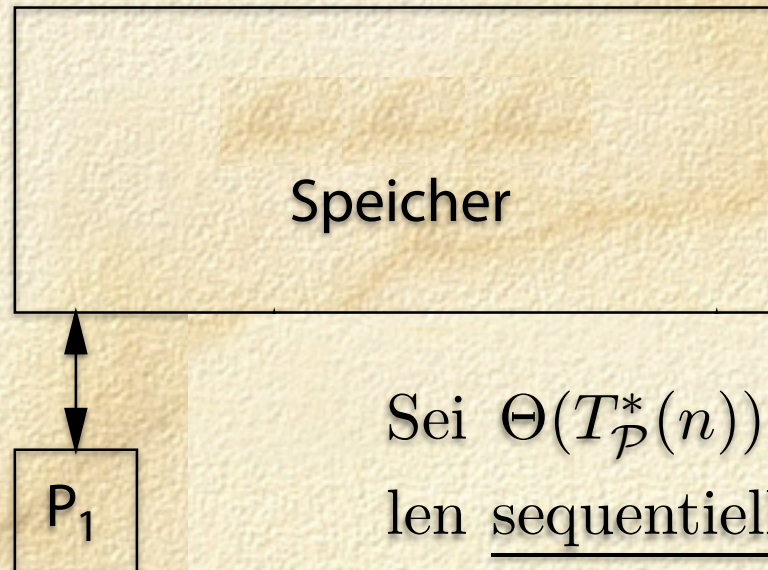
optimal ???

Der Alg. heißt *optimal*, falls

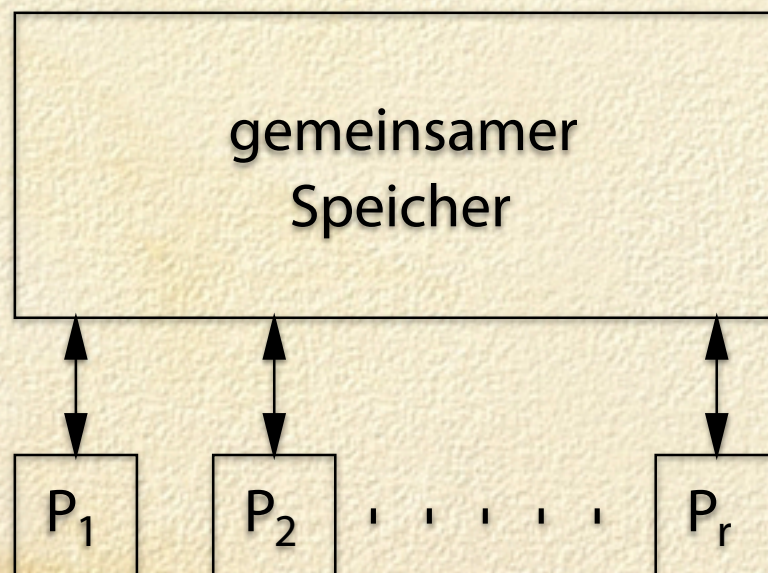
$$W_A(n) = \Theta(T_{\mathcal{P}}^*(n)) \\ = 21$$

optimaler paralleler Algorithmus

sequentieller Alg.
mit $T(n)$

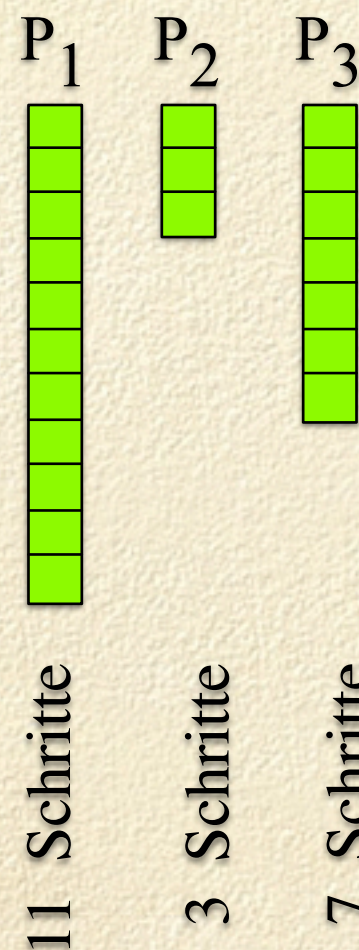


Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen



$$W_A(n) = 21$$

$$W_{A'}(n) = 17$$



paralleler Alg.

$T(n) = ?$ *besser!*

optimal ???

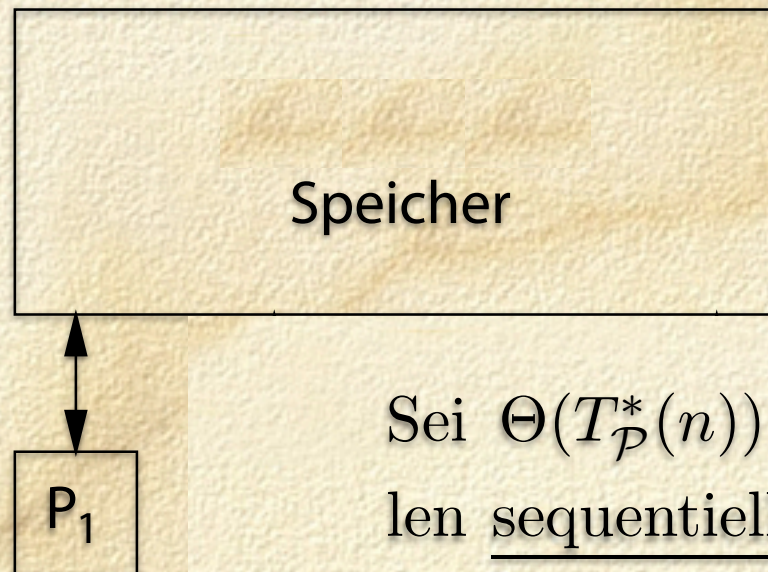
Der Alg. heißt *optimal*, falls

$$W_A(n) = \Theta(T_{\mathcal{P}}^*(n))$$

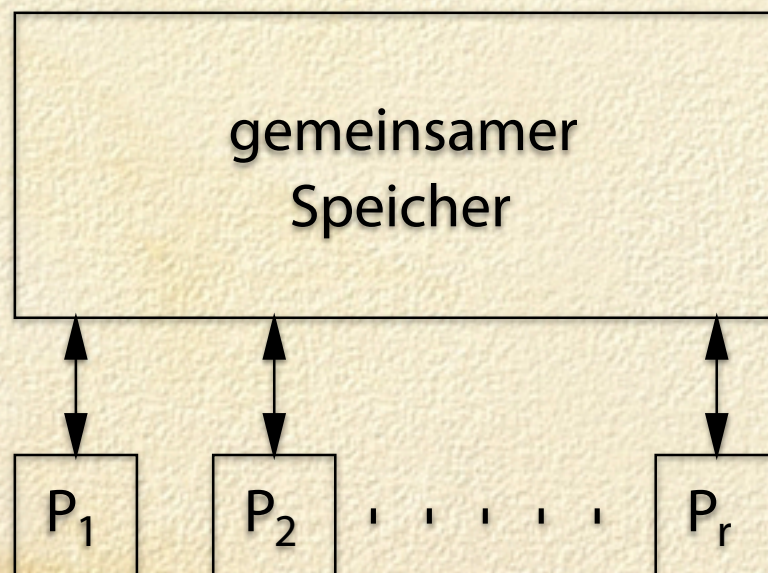
$$= 21$$

optimaler paralleler Algorithmus

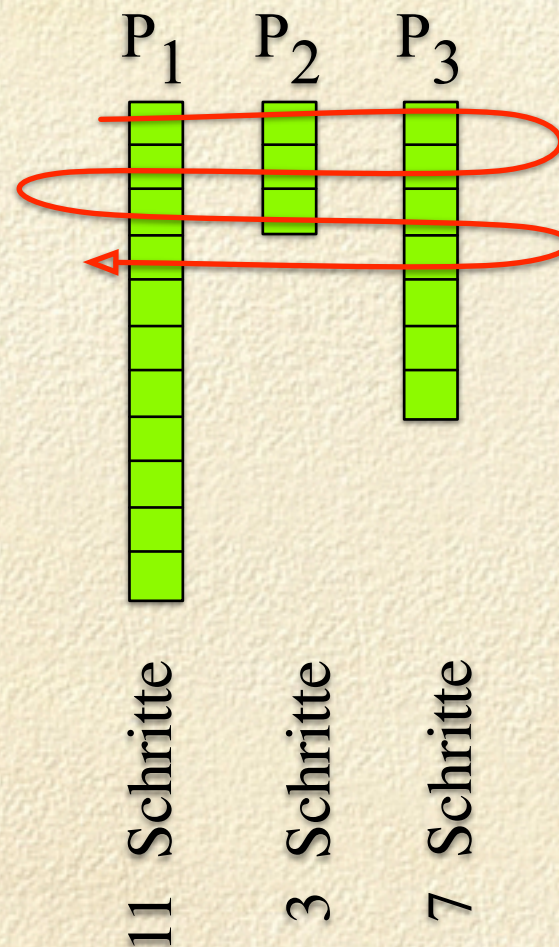
sequentieller Alg.
mit $T(n)$



Sei $\Theta(T_{\mathcal{P}}^*(n))$ die beste Zeitkomplexität für ein Problem $\mathcal{P}$ unter allen sequentiellen Algorithmen, die das Problem $\mathcal{P}$ lösen



$$W_A(n) = 21$$
$$W_{A'}(n) = 17$$



paralleler Alg.

$T(n) = ?$ besser!

optimal ???

Der Alg. heißt *optimal*, falls

$$W_A(n) = \Theta(T_{\mathcal{P}}^*(n))$$
$$= 21$$

Ist der Max-Algorithmus
mit $O(n)$ Zeit und
 $O(n^2)$ Prozessoren optimal?

Ist der Max-Algorithmus
mit $O(n)$ Zeit und
 $O(n^2)$ Prozessoren optimal?

d.h. $W_A(n) = \Theta(T_{\mathcal{P}}^*(n))$?

Prozessoren: P_{ij} , $1 \leq i, j \leq n$;

Input: $M(0) \leftarrow n$, $M(1) \leftarrow x_1, \dots, M(n) \leftarrow x_n$. Die Register des globalen Speichers erhalten die Zahlen x_i , unter denen das Maximum gefunden werden soll;

Arbeitsspeicher: $Y(1), \dots, Y(n)$ – Register des globalen Speichers;

Output: Maximum liegt in $M(0)$.

Algorithmus für den Prozessor P_{ij} :

1. $Y(i) \leftarrow 0$;
2. if $M(i) < M(j)$ then $Y(i) \leftarrow 1$;
3. if $Y(i) = 0$ then $M(0) \leftarrow M(i)$

$$T_A(n) = O(1)$$

$$P_A(n) = O(n^2)$$

$$W_A(n) = O(n^2)$$

M(0)= 4 ⁷	M(1)=2	M(2)=7	M(3)=3	M(4)=4
M(1)=2	F	T	T	T
M(2)=7	F	F	F	F
M(3)=3	F	T	F	T
M(4)=4	F	T	F	F

Prozessoren: P_{ij} , $1 \leq i, j \leq n$;

Input: $M(0) \leftarrow n$, $M(1) \leftarrow x_1, \dots, M(n) \leftarrow x_n$. Die Register des globalen Speichers erhalten die Zahlen x_i , unter denen das Maximum gefunden werden soll;

Arbeitsspeicher: $Y(1), \dots, Y(n)$ – Register des globalen Speichers;

Output: Maximum liegt in $M(0)$.

Algorithmus für den Prozessor P_{ij} :

1. $Y(i) \leftarrow 0$;
2. if $M(i) < M(j)$ then $Y(i) \leftarrow 1$;
3. if $Y(i) = 0$ then $M(0) \leftarrow M(i)$

$$T_A(n) = O(1)$$

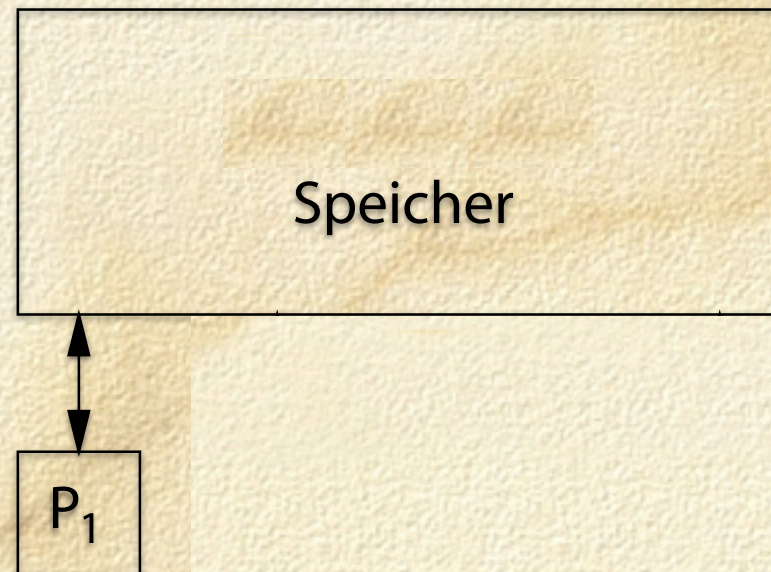
$$P_A(n) = O(n^2)$$

$$W_A(n) = O(n^2)$$

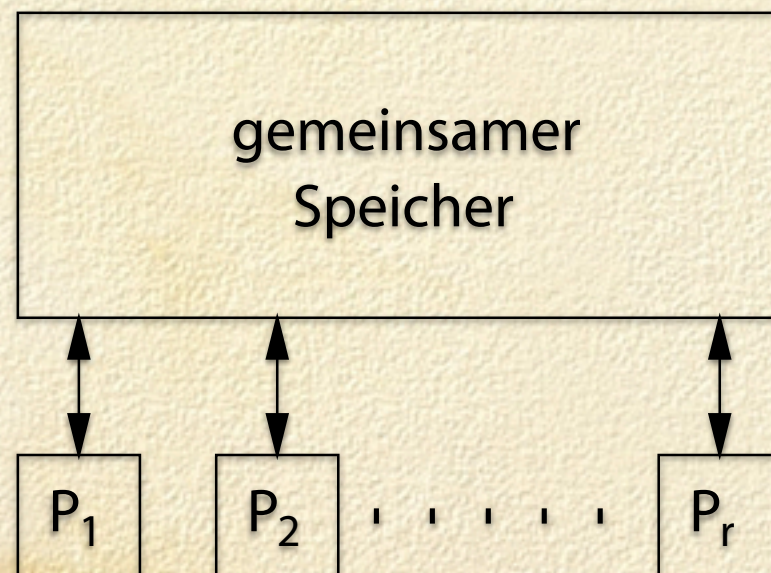
$$\text{d.h. } W_A(n) \stackrel{?}{=} \Theta(T_{\mathcal{P}}^*(n))$$

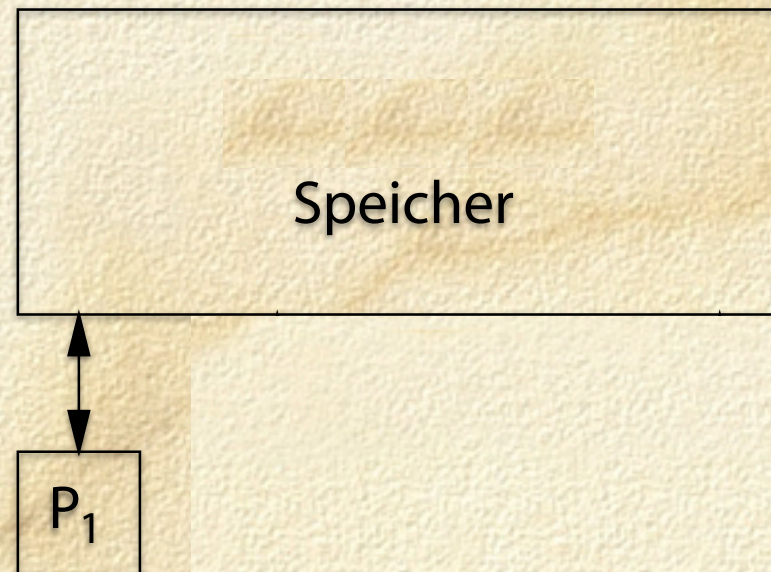
M(0)= 4 ⁷	M(1)=2	M(2)=7	M(3)=3	M(4)=4
M(1)=2	F	T	T	T
M(2)=7	F	F	F	F
M(3)=3	F	T	F	T
M(4)=4	F	T	F	F

sequentielles Sortieren



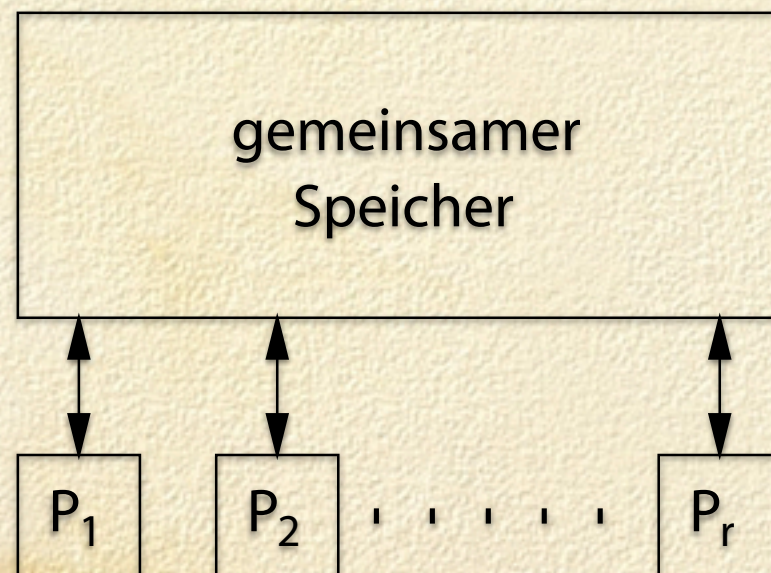
paralleles Sortieren
(am Ende des Kapitels)



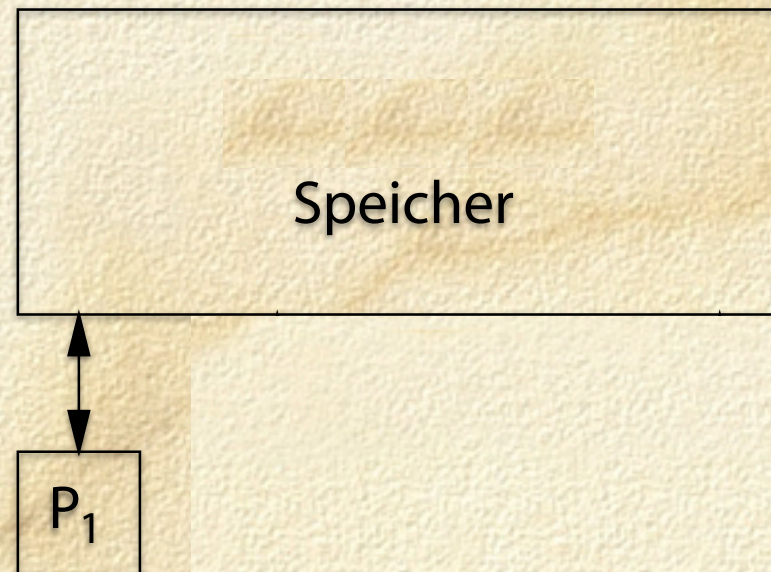


sequentielles Sortieren

$T(n) = ?$

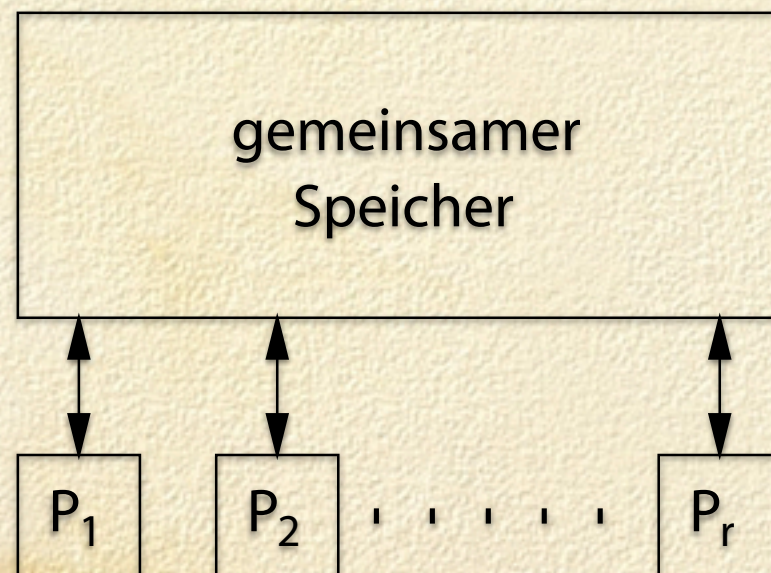


paralleles Sortieren
(am Ende des Kapitels)

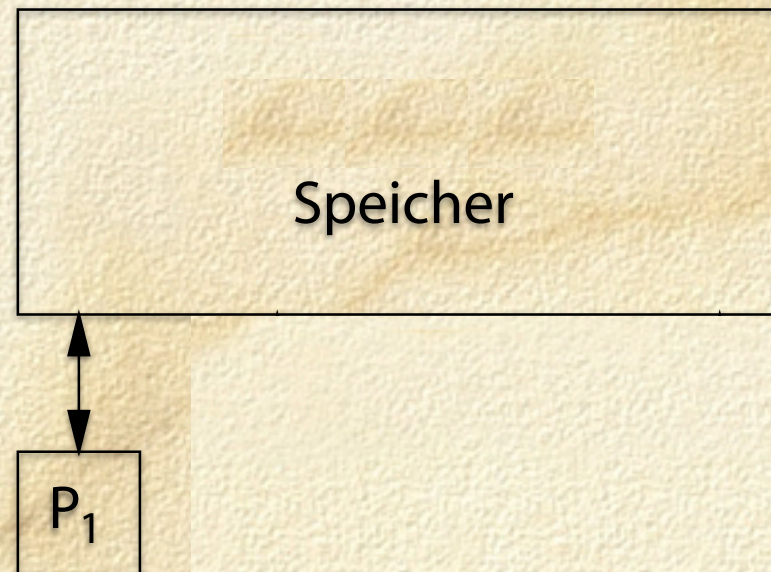


sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

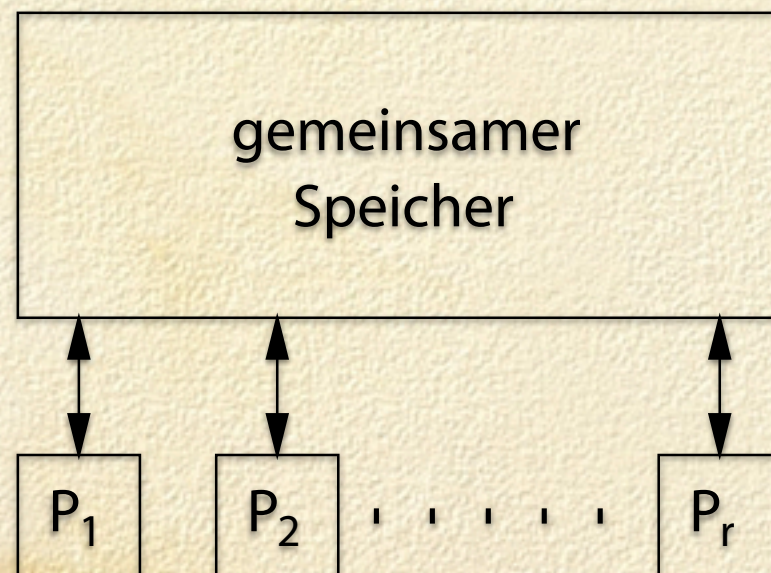


paralleles Sortieren
(am Ende des Kapitels)



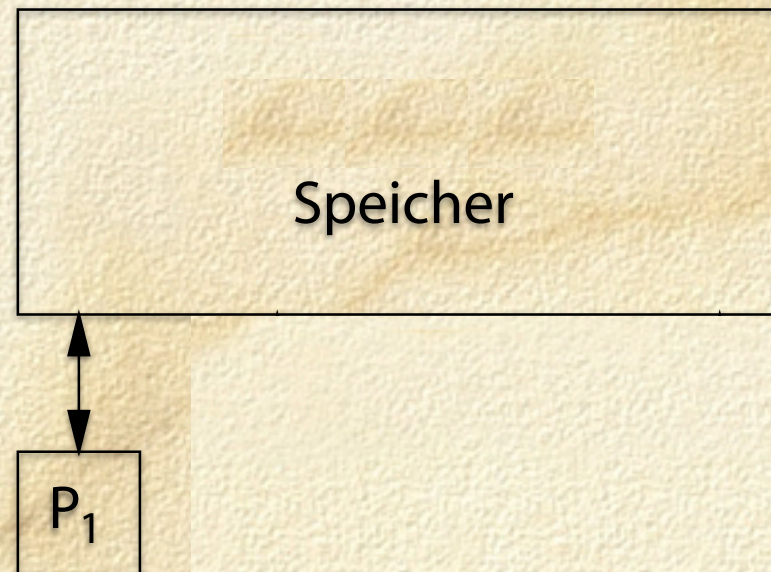
sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$



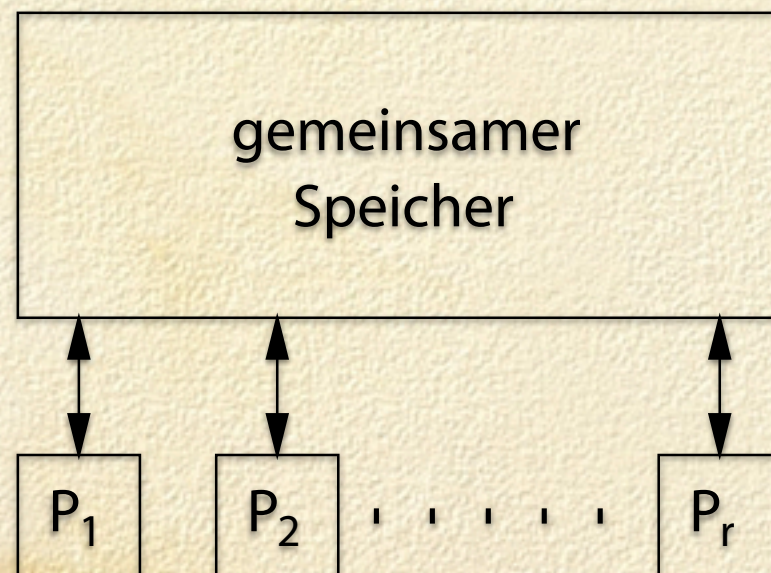
paralleles Sortieren
(am Ende des Kapitels)

$$T(n) = ?$$



sequentielles Sortieren

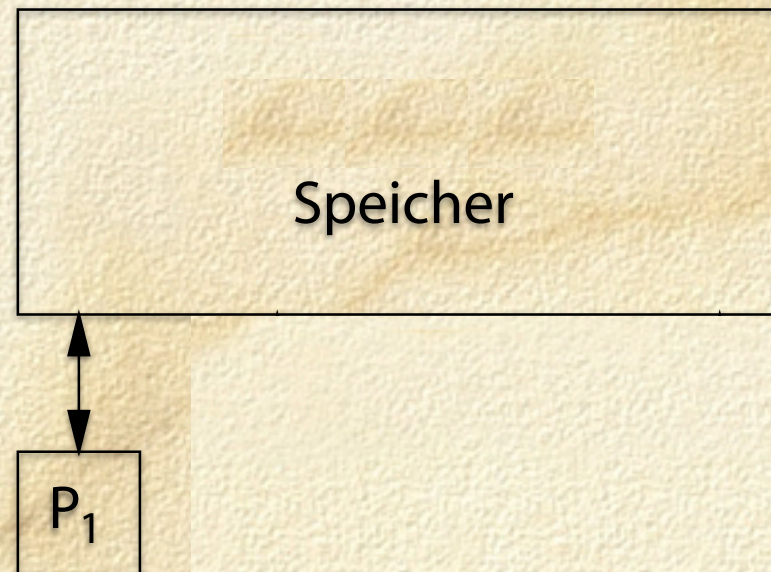
$$T(n) = ? = O(n \log n)$$



paralleles Sortieren
(am Ende des Kapitels)

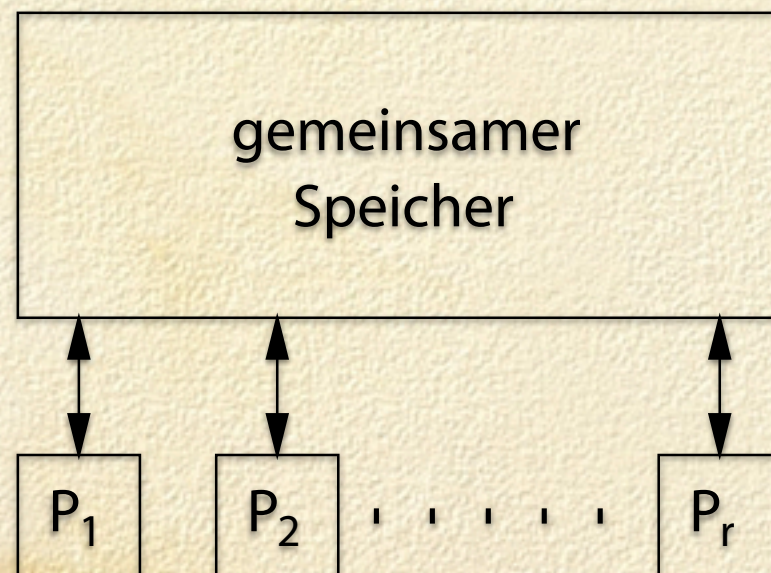
$$T(n) = ?$$

$$= O(\log n \log \log n)$$



sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

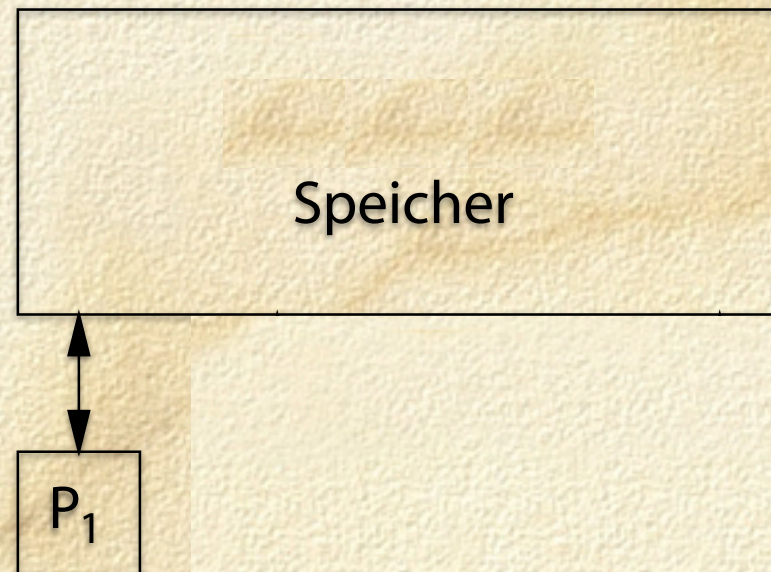


paralleles Sortieren
(am Ende des Kapitels)

$$T(n) = ?$$

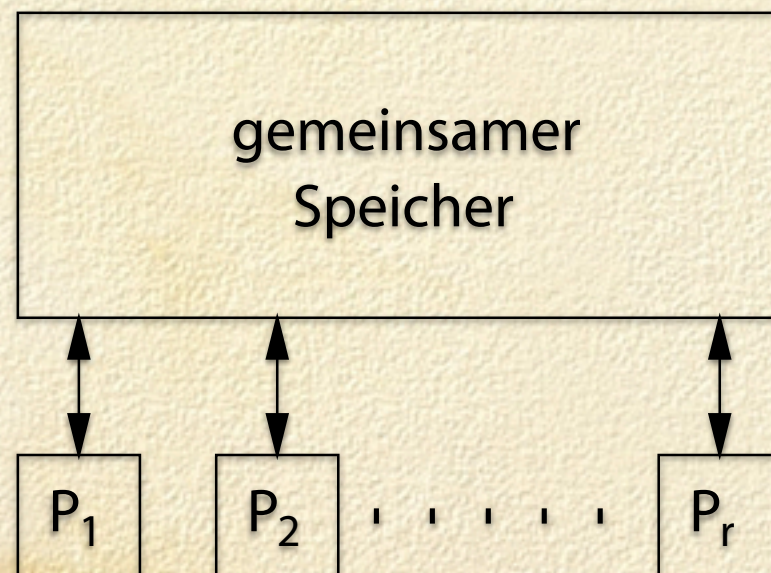
$$= O(\log n \log \log n)$$

$$W(n) = ?$$



sequentielles Sortieren

$$T(n) = ? \quad = O(n \log n)$$



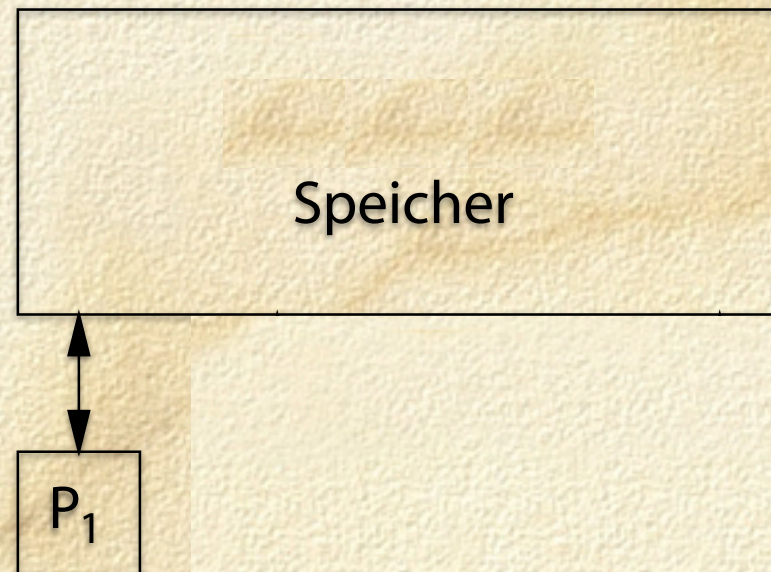
paralleles Sortieren
(am Ende des Kapitels)

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

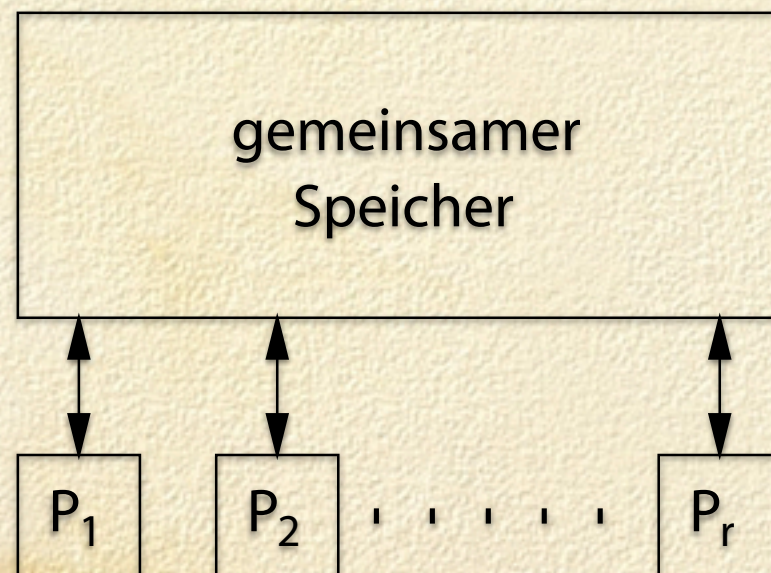
$$W(n) = ?$$

$$= O(n \log n)$$



sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$



paralleles Sortieren
(am Ende des Kapitels)

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$P(n) = n/2 = O(n) \quad = O(n \log n)$$

sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

paralleles Sortieren

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$P(n) = n/2 = O(n) \quad = O(n \log n)$$

n	$\log n$	$\log \log n$	$\log n \log \log n$	$n \log n$
-----	----------	---------------	----------------------	------------

sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

paralleles Sortieren

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$P(n) = n/2 = O(n) \quad = O(n \log n)$$

n	$\log n$	$\log \log n$	$\log n \log \log n$	$n \log n$
2				
4				
8				
16				
32				
64				
128				
256				

sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

paralleles Sortieren

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$P(n) = n/2 = O(n) \quad = O(n \log n)$$

n	$\log n$	$\log \log n$	$\log n \log \log n$	$n \log n$
2	1			
4	2			
8	3			
16	4			
32	5			
64	6			
128	7			
256	8			

sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

paralleles Sortieren

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$P(n) = n/2 = O(n) \quad = O(n \log n)$$

n	$\log n$	$\log \log n$	$\log n \log \log n$	$n \log n$
2	1	0		
4	2	1		
8	3			
16	4	2		
32	5			
64	6			
128	7			
256	8	3		

sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

paralleles Sortieren

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$P(n) = n/2 = O(n) \quad = O(n \log n)$$

n	$\log n$	$\log \log n$	$\log n \log \log n$	$n \log n$
2	1	0	0	
4	2	1	2	
8	3			
16	4	2	8	
32	5			
64	6			
128	7			
256	8	3	24	

sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

paralleles Sortieren

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$= O(n \log n)$$

$$P(n) = n/2 = O(n)$$

n	$\log n$	$\log \log n$	$\log n \log \log n$	$n \log n$
2	1	0	0	2
4	2	1	2	8
8	3			
16	4	2	8	64
32	5			
64	6			
128	7			
256	8	3	24	2048

sequentielles Sortieren

$$T(n) = ? = O(n \log n)$$

paralleles Sortieren

$$T(n) = ?$$

$$= O(\log n \log \log n)$$

$$W(n) = ?$$

$$= O(n \log n)$$

$$P(n) = n/2 = O(n)$$

6.3 Paralleles Suchen und optimales Mischen

Definition 6.24 Gegeben sei eine lineare Ordnung $(S, \leq)$ (Def. 1.21) und zwei Teilmengen $\tilde{A}, \tilde{B} \subseteq S$ mit $|\tilde{A}| = |\tilde{B}| = n > 0$, zu denen wir die Folgen: $A = (a_1, a_2, \dots, a_n)$ und $B = (b_1, b_2, \dots, b_n)$, mit $i < j \Rightarrow a_i \leq a_j \wedge b_i \leq b_j$ ($1 \leq i, j \leq n$) benutzen. Die **Mischung** (merge) von A und B ist die Folge $C = (c_1, c_2, \dots, c_{2n})$ mit $i < j \rightarrow c_i \leq c_j$, die A und B als disjunkte Teilfolgen enthält.

6.3 Paralleles Suchen und optimales Mischen

Definition 6.24 Gegeben sei eine lineare Ordnung $(S, \leq)$ (Def. 1.21) und zwei Teilmengen $\tilde{A}, \tilde{B} \subseteq S$ mit $|\tilde{A}| = |\tilde{B}| = n > 0$, zu denen wir die Folgen: $A = (a_1, a_2, \dots, a_n)$ und $B = (b_1, b_2, \dots, b_n)$, mit $i < j \Rightarrow a_i \leq a_j \wedge b_i \leq b_j$ ($1 \leq i, j \leq n$) benutzen. Die **Mischung** (merge) von A und B ist die Folge $C = (c_1, c_2, \dots, c_{2n})$ mit $i < j \rightarrow c_i \leq c_j$, die A und B als disjunkte Teilfolgen enthält.

Beispiel:

(2,4,4)

(3,4,7)

6.3 Paralleles Suchen und optimales Mischen

Definition 6.24 Gegeben sei eine lineare Ordnung $(S, \leq)$ (Def. 1.21) und zwei Teilmengen $\tilde{A}, \tilde{B} \subseteq S$ mit $|\tilde{A}| = |\tilde{B}| = n > 0$, zu denen wir die Folgen: $A = (a_1, a_2, \dots, a_n)$ und $B = (b_1, b_2, \dots, b_n)$, mit $i < j \Rightarrow a_i \leq a_j \wedge b_i \leq b_j$ ($1 \leq i, j \leq n$) benutzen. Die **Mischung** (merge) von A und B ist die Folge $C = (c_1, c_2, \dots, c_{2n})$ mit $i < j \rightarrow c_i \leq c_j$, die A und B als disjunkte Teilfolgen enthält.

Beispiel:

(2,4,4)

(3,4,7)

(2,3,4,4,4,7)

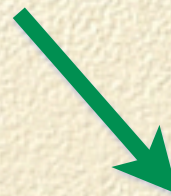
6.3 Paralleles Suchen und optimales Mischen

Definition 6.24 Gegeben sei eine lineare Ordnung $(S, \leq)$ (Def. 1.21) und zwei Teilmengen $\tilde{A}, \tilde{B} \subseteq S$ mit $|\tilde{A}| = |\tilde{B}| = n > 0$, zu denen wir die Folgen: $A = (a_1, a_2, \dots, a_n)$ und $B = (b_1, b_2, \dots, b_n)$, mit $i < j \Rightarrow a_i \leq a_j \wedge b_i \leq b_j$ ($1 \leq i, j \leq n$) benutzen. Die **Mischung** (merge) von A und B ist die Folge $C = (c_1, c_2, \dots, c_{2n})$ mit $i < j \rightarrow c_i \leq c_j$, die A und B als disjunkte Teilfolgen enthält.

Beispiel:

(2,4,4)

(3,4,7)

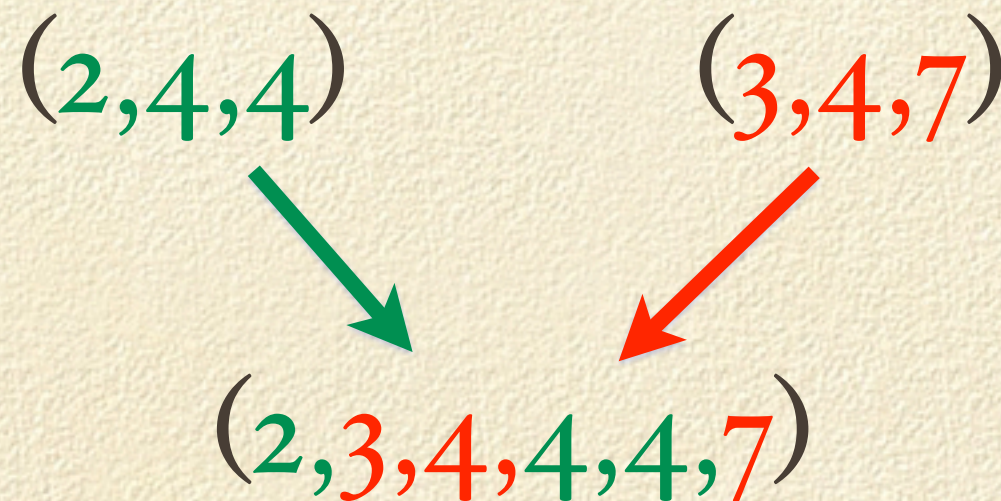


(2,3,4,4,4,7)

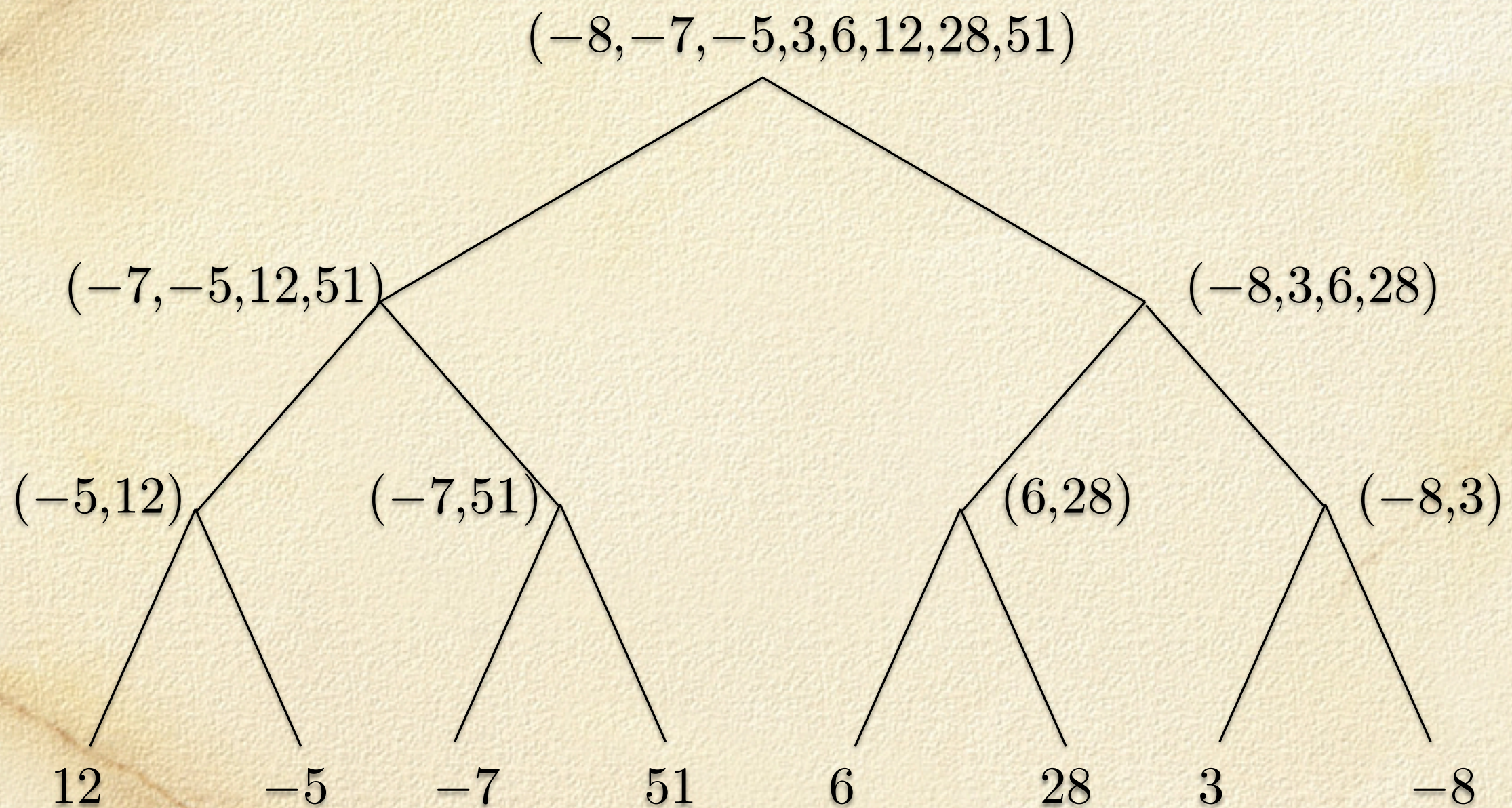
6.3 Paralleles Suchen und optimales Mischen

Definition 6.24 Gegeben sei eine lineare Ordnung $(S, \leq)$ (Def. 1.21) und zwei Teilmengen $\tilde{A}, \tilde{B} \subseteq S$ mit $|\tilde{A}| = |\tilde{B}| = n > 0$, zu denen wir die Folgen: $A = (a_1, a_2, \dots, a_n)$ und $B = (b_1, b_2, \dots, b_n)$, mit $i < j \Rightarrow a_i \leq a_j \wedge b_i \leq b_j$ ($1 \leq i, j \leq n$) benutzen. Die **Mischung** (merge) von A und B ist die Folge $C = (c_1, c_2, \dots, c_{2n})$ mit $i < j \rightarrow c_i \leq c_j$, die A und B als disjunkte Teilfolgen enthält.

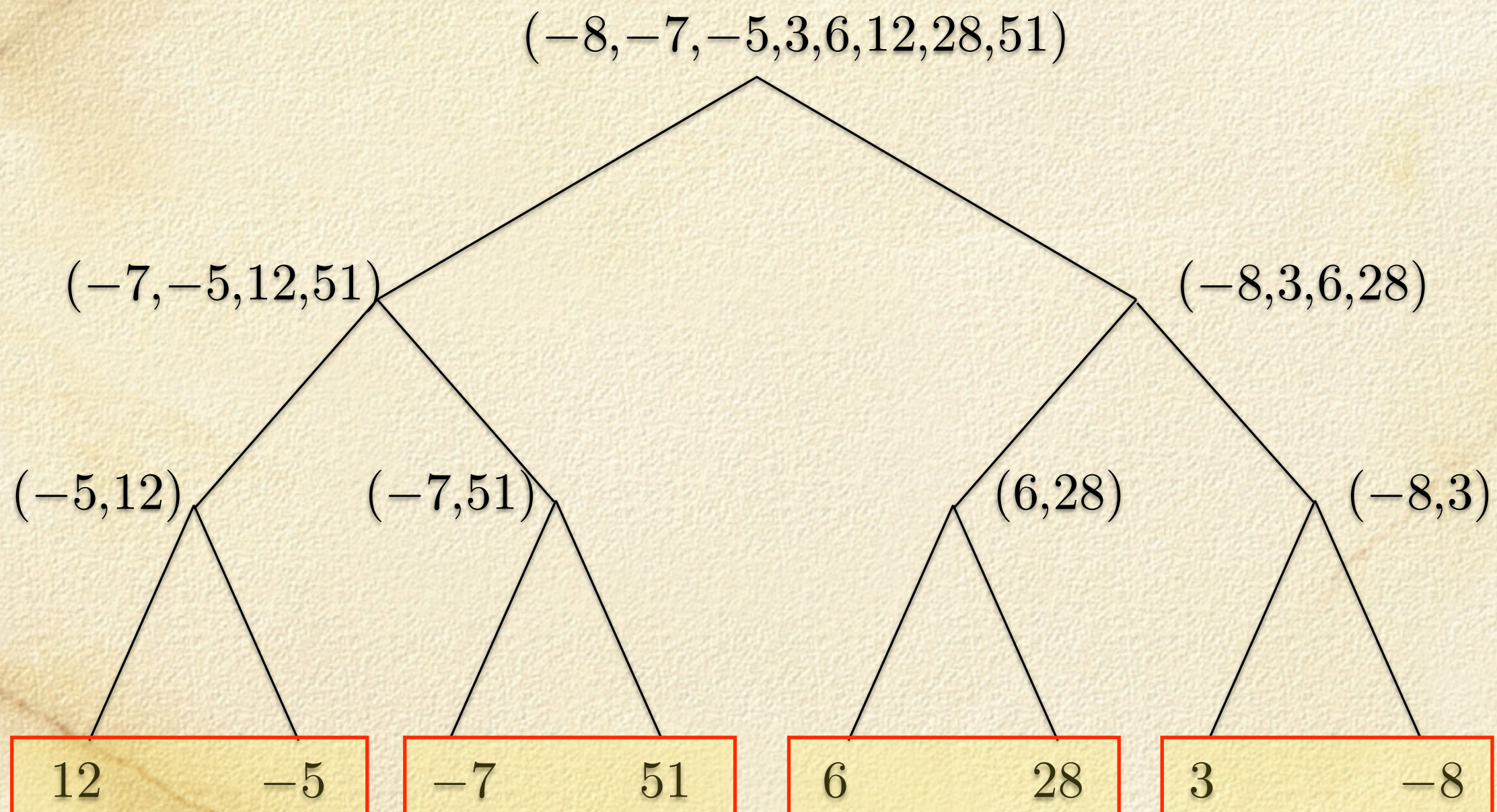
Beispiel:



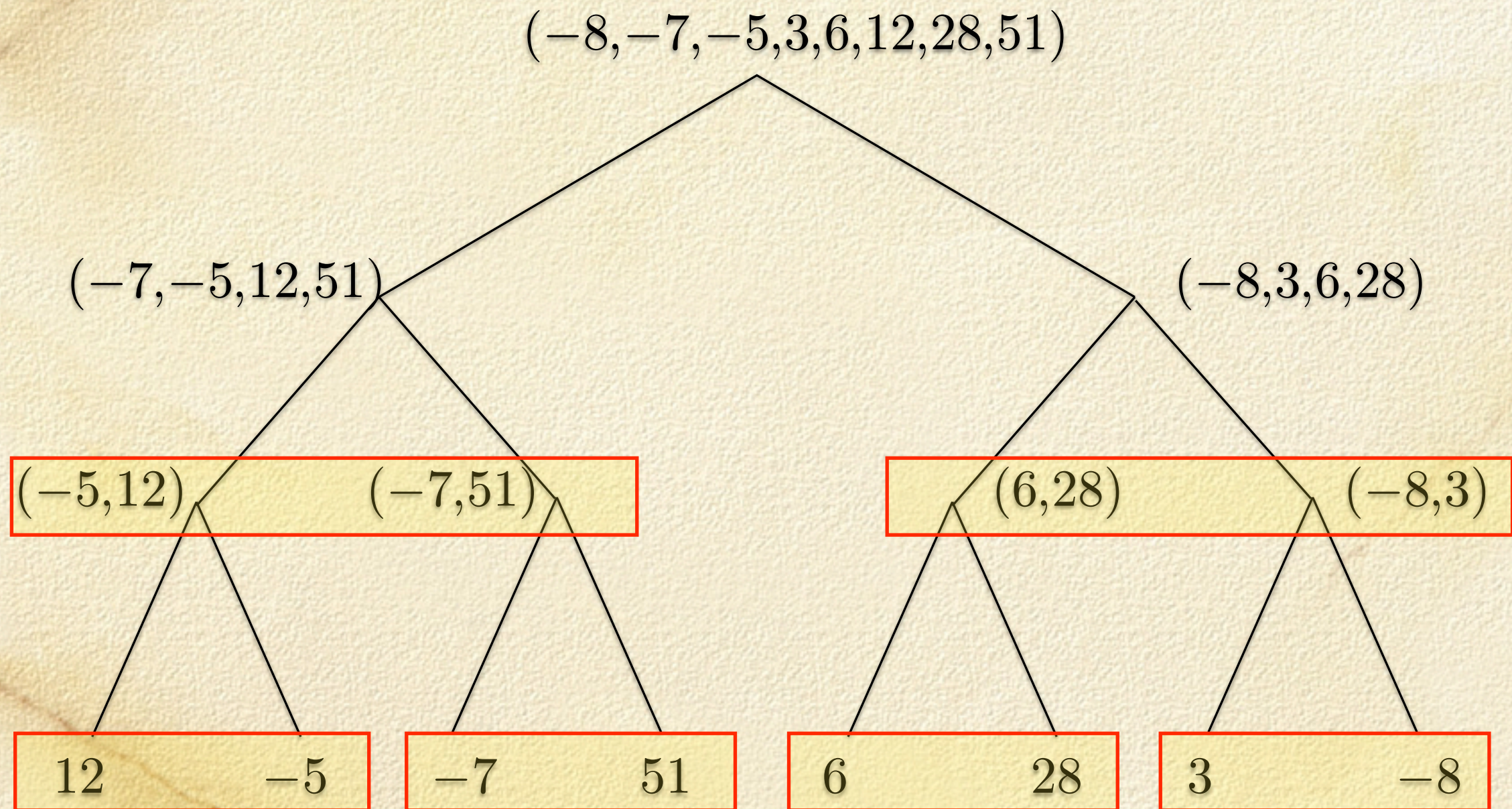
Anwendung: z.B.: *paralleles Sortieren*
paralleles Merge-Sort



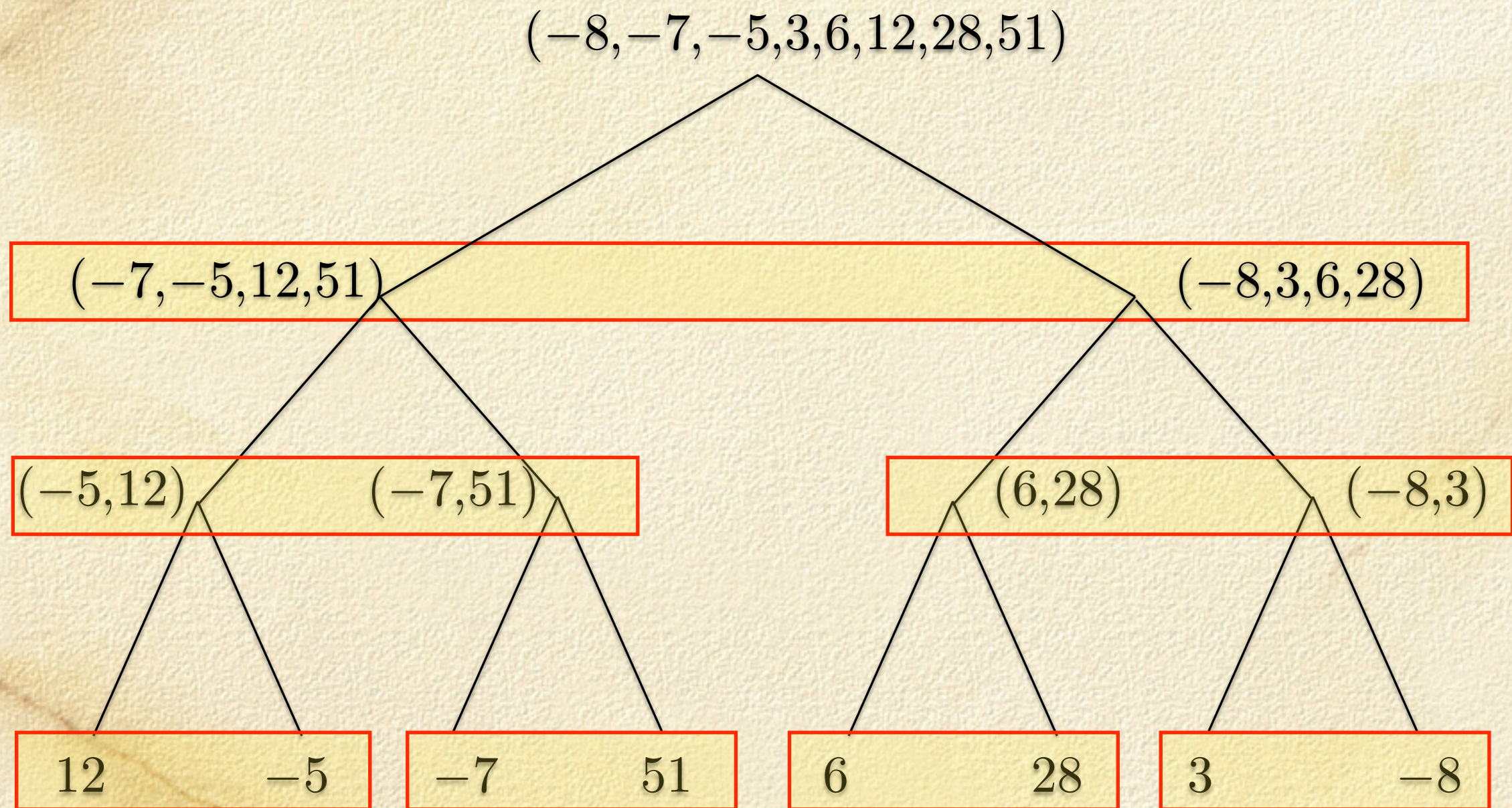
Anwendung: z.B.: *paralleles Sortieren*
paralleles Merge-Sort



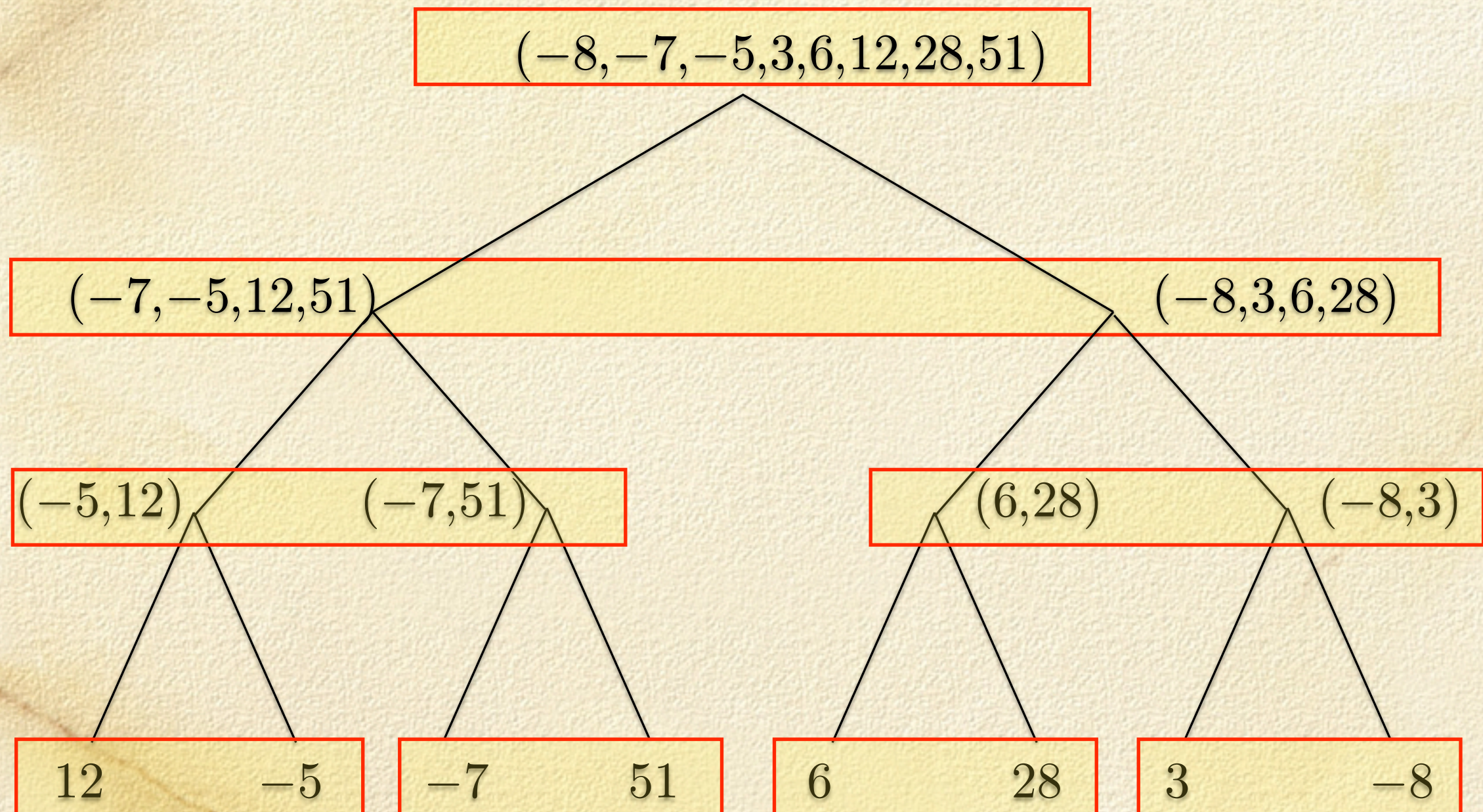
Anwendung: z.B.: *paralleles Sortieren*
paralleles Merge-Sort



Anwendung: z.B.: *paralleles Sortieren*
paralleles Merge-Sort



Anwendung: z.B.: *paralleles Sortieren*
paralleles Merge-Sort



Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

$$X = (25, -13, 26, 31, 54, 7)$$

Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

$$X = (25, -13, 26, 31, 54, 7)$$

$$\text{rank}(13, X) = ?$$

Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

$$X = (25, -13, 26, 31, 54, 7)$$

$$\text{rank}(13, X) = ?$$

$$X' = (-13, 7, 25, 26, 31, 54)$$

Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

$X = (25, -13, 26, 31, 54, 7)$

$\text{rank}(13, X) = ?$

$X' = (-13, 7, 25, 26, 31, 54)$



Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

$$X = (25, -13, 26, 31, 54, 7)$$

$$\text{rank}(13, X) = ?$$

$$X' = (-13, 7, 25, 26, 31, 54)$$



$$\text{rank}(13:X) = 2$$

Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

$$X = (25, -13, 26, 31, 54, 7)$$

$$\text{rank}(13, X) = ?$$

$$X' = (-13, 7, 25, 26, 31, 54)$$



$$\text{rank}(13 : X) = 2$$

$$Y = (13, 27, -27)$$

Anzahl der Elemente x_i
mit $x_i < x$

Definition 6.25 Sei $X = (x_1, x_2, \dots, x_t)$ eine Folge mit Elementen aus der linear geordneten Menge S . Für $x \in S$ ist der **Rang** von x in X definiert als:

$\text{rank}(x : X) := |\{i | i \in \{1, \dots, t\} \wedge x_i < x\}|$. Für eine weitere solche Folge

$Y = (y_1, \dots, y_s)$ sei $\text{rank}(Y : X) := (r_1, r_2, \dots, r_s)$ mit $r_i = \text{rank}(y_i : X)$

$$X = (25, -13, 26, 31, 54, 7)$$

$$\text{rank}(13, X) = ?$$

$$X' = (-13, 7, 25, 26, 31, 54)$$



$$\text{rank}(13 : X) = 2$$

$$Y = (13, 27, -27)$$

$$\text{rank}(Y : X) = (2, 4, 0)$$

Zur Vereinfachung nehmen wir jetzt $A \cap B = \emptyset$ an. Das Problem, die Folgen A und B zu mischen, wird dann zu:

Bestimme für jedes $x \in A \cup B : i := \text{rank}(x : A \cup B)$
(dann ist x das $(i + 1)$ -te Element c_i in C)

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

Berechne $\text{rank}(B : A)$

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:

sortiert

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Für $b_i \in B$ kann durch **binäres Suchen** j_i mit $a_{j_i} < b_i < a_{j_i+1}$ in $O(\log n)$ (sequentieller) Zeit berechnet werden, d.h. $j_i = \text{rank}(b_i : A)$. Dieses Verfahren kann man parallel auf alle Elemente von B anwenden, d.h. wir erhalten einen parallelen Algorithmus mit $O(\log n)$ (paralleler) Zeit und $O(n \log n)$ Operationen.

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Für $b_i \in B$ kann durch **binäres Suchen** j_i mit $a_{j_i} < b_i < a_{j_i+1}$ in $O(\log n)$ (sequentieller) Zeit berechnet werden, d.h. $j_i = \text{rank}(b_i : A)$. Dieses Verfahren kann man parallel auf alle Elemente von B anwenden, d.h. wir erhalten einen parallelen Algorithmus mit **$O(\log n)$** (paralleler) Zeit und $O(n \log n)$ Operationen.

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Für $b_i \in B$ kann durch **binäres Suchen** j_i mit $a_{j_i} < b_i < a_{j_i+1}$ in $O(\log n)$ (sequentieller) Zeit berechnet werden, d.h. $j_i = \text{rank}(b_i : A)$. Dieses Verfahren kann man parallel auf alle Elemente von B anwenden, d.h. wir erhalten einen parallelen Algorithmus mit **$O(\log n)$** (paralleler) Zeit und **$O(n \log n)$** Operationen.

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Für $b_i \in B$ kann durch **binäres Suchen** j_i mit $a_{j_i} < b_i < a_{j_i+1}$ in $O(\log n)$ (sequentieller) Zeit berechnet werden, d.h. $j_i = \text{rank}(b_i : A)$. Dieses Verfahren kann man parallel auf alle Elemente von B anwenden, d.h. wir erhalten einen parallelen Algorithmus mit **$O(\log n)$** (paralleler) Zeit und **$O(n \log n)$** Operationen.

optimal ???

Wegen $\text{rank}(x : A \cup B) = \text{rank}(x : A) + \text{rank}(x : B)$ genügt uns ein Algorithmus für:

binäres Suchen

Berechne $\text{rank}(B : A)$

A:



sortiert

Für $b_i \in B$ kann durch **binäres Suchen** j_i mit $a_{j_i} < b_i < a_{j_i+1}$ in $O(\log n)$ (sequentieller) Zeit berechnet werden, d.h. $j_i = \text{rank}(b_i : A)$. Dieses Verfahren kann man parallel auf alle Elemente von B anwenden, d.h. wir erhalten einen parallelen Algorithmus mit **$O(\log n)$** (paralleler) Zeit und **$O(n \log n)$** Operationen.

optimal ???

Nein, denn es gibt seq. Alg. mit $O(n)$ Zeit!!

nächstes Ziel: optimaler paralleler Algorithmus dafür

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!



nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

$$4/2 = 2$$

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

$$4/2 = 2$$

$$8/3 = 2,66$$

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

$$4/2 = 2$$

$$8/3 = 2,66$$

$$16/4 = 4$$

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

$$4/2 = 2$$

$$8/3 = 2,66$$

$$16/4 = 4$$

$$32/5 = 6,4$$

nächstes Ziel: optimaler paralleler Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

<

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

$$4/2 = 2$$

$$8/3 = 2,66$$

$$16/4 = 4$$

$$32/5 = 6,4$$

nächstes Ziel: *optimaler* *paralleler* Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

<

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

alle $x \in A_i \cup B_i$ größer als *alle* die Elemente in $A_{i-1} \cup B_{i-1}$

$$4/2 = 2$$

$$8/3 = 2,66$$

$$16/4 = 4$$

$$32/5 = 6,4$$

nächstes Ziel: *optimaler* *paralleler* Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

<

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

$$4/2 = 2$$

$$8/3 = 2,66$$

$$16/4 = 4$$

$$32/5 = 6,4$$

alle $x \in A_i \cup B_i$ größer als die Elemente in $A_{i-1} \cup B_{i-1}$
alle

$$m = 4 \text{ und } k_m := k(m) = \frac{m}{\log m} = 2$$

a_1	$\dots$	a_8	b_1	$\dots$	b_4
A=(4, 6, 7, 10, 12, 15, 18, 20)			B=(3, 9, 16, 21)		

nächstes Ziel: *optimaler* *paralleler* Algorithmus dafür

Einteilung von A und B in Blöcke “passender” Länge

A hat Länge n

B hat Länge m

<

A:	A_0	A_1	$\dots$	A_{k_m-1}
B:	B_0	B_1	$\dots$	B_{k_m-1}

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

$$4/2 = 2$$

$$8/3 = 2,66$$

$$16/4 = 4$$

$$32/5 = 6,4$$

alle $x \in A_i \cup B_i$ größer als die Elemente in $A_{i-1} \cup B_{i-1}$
alle

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

a_1	$\dots$	a_8	b_1	$\dots$	b_4
A=(4,	6, 7, 10, 12, 15, 18,	20)	B=(3,	9, 16,	21)

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

$a_1 \quad \dots \quad a_8 \quad b_1 \quad \dots \quad b_4$
 $A=(4, \quad 6, 7, 10, 12, 15, 18, \quad 20) \quad B=(3, \quad 9, 16, \quad 21)$

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

a_1	...	a_8	b_1	...	b_4
A=(4,	6, 7, 10, 12, 15, 18,	20)	B=(3,	9, 16,	21)

A = (4, 6, 7, 10, 12, 15, 18, 20)

B = (3, 9, 16, 21)

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

a_1	$\dots$	a_8	b_1	$\dots$	b_4
A=(4, 6, 7, 10, 12, 15, 18, 20)			B=(3, 9, 16, 21)		

$$|B_i| = \log m$$

A = (4, 6, 7, 10, 12, 15, 18, 20)

B = (3, 9, 16, 21)

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

a_1	$\dots$	a_8	b_1	$\dots$	b_4
A=(4, 6, 7, 10, 12, 15, 18, 20)			B=(3, 9, 16, 21)		

$$|B_i| = \log m$$

A = (4, 6, 7, 10, 12, 15, 18, 20)

B = (3, 9, ~~16~~, 21)

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

a_1	$\dots$	a_8	b_1	$\dots$	b_4
A=(4, 6, 7, 10, 12, 15, 18, 20)			B=(3, 9, 16, 21)		

$$|B_i| = \log m$$

A = (4, 6, 7, 10, 12, 15, 18, 20)

B = (3, 9, ~~16~~, 21)

B_0 B_1

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

a_1 $\dots$ a_8 b_1 $\dots$ b_4
 $A = (4, 6, 7, 10, 12, 15, 18, 20)$ $B = (3, 9, 16, 21)$

$|B_i| = \log m$

$A = (4, 6, 7, 10, 12, 15, 18, 20)$

$B = (3, 9, 16, 21)$

B_0 B_1

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

a_1 $\dots$ a_8 b_1 $\dots$ b_4
 $A = (4, 6, 7, 10, 12, 15, 18, 20)$ $B = (3, 9, 16, 21)$

$|B_i| = \log m$

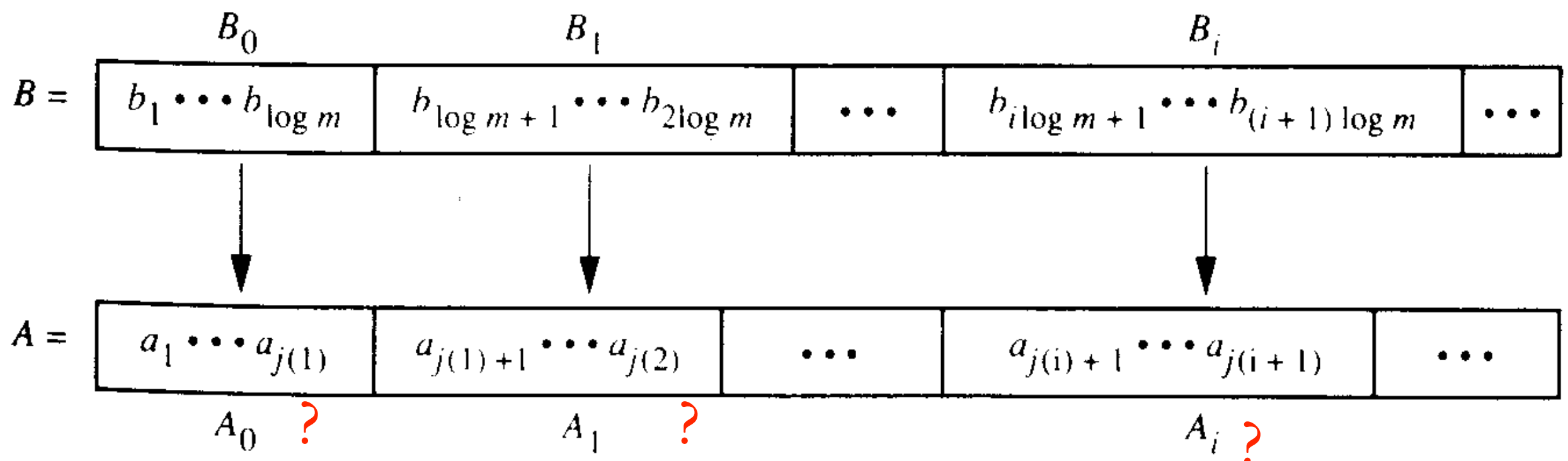
A_0 A_1
 $A = (4, 6, 7, 10, 12, 15, 18, 20)$
 $B = (3, 9, 16, 21)$
 B_0 B_1

$m = 4$ und $k_m := k(m) = \frac{m}{\log m} = 2$ *Schreibweise für Indizes!!!*

$a_1 \quad \dots \quad a_8 \quad b_1 \quad \dots \quad b_4$
 $A = (4, 6, 7, 10, 12, 15, 18, 20) \quad B = (3, 9, 16, 21)$

$$|B_i| = \log m$$

$A_0 \quad A_1$
 $A = (4, 6, 7, 10, 12, 15, 18, 20)$
 $B = (3, 9, 16, 21)$
 $B_0 \quad B_1$



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

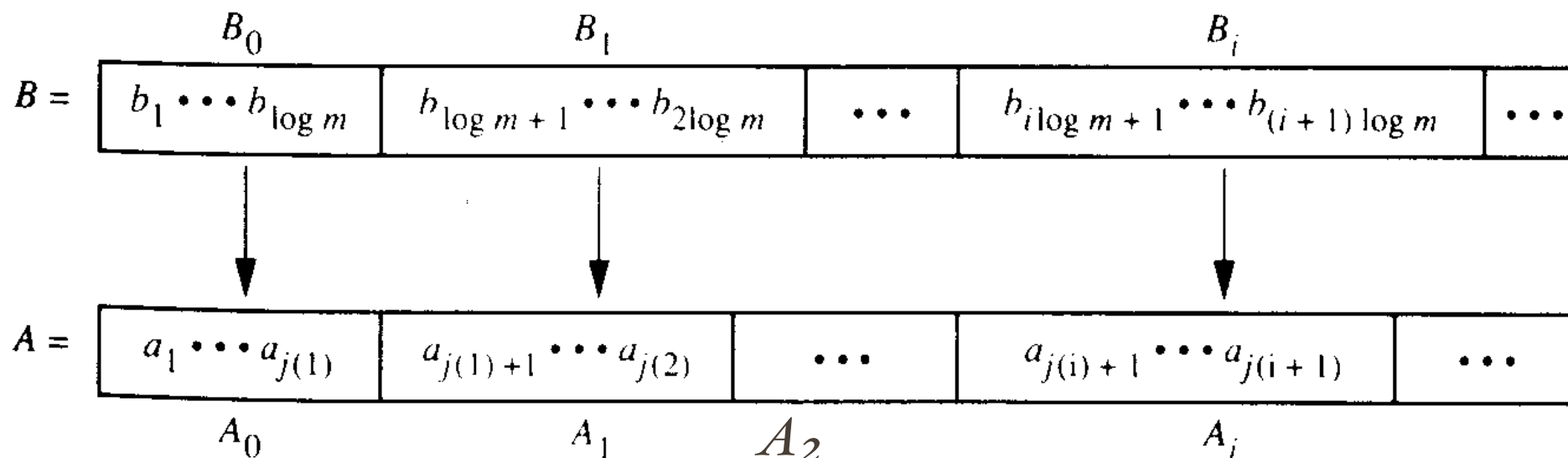
Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

- (1) $|B_i| = \log m$
- (2) $\sum_i |A_i| = n$ und
- (3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$
2. **for** $1 \leq i \leq k(m) - 1$ **pardo**
 - 2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$
3. **for** $0 \leq i \leq k(m) - 1$ **pardo**
 - 3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$
 - 3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

$$m = 4$$

$$k_m = 2$$

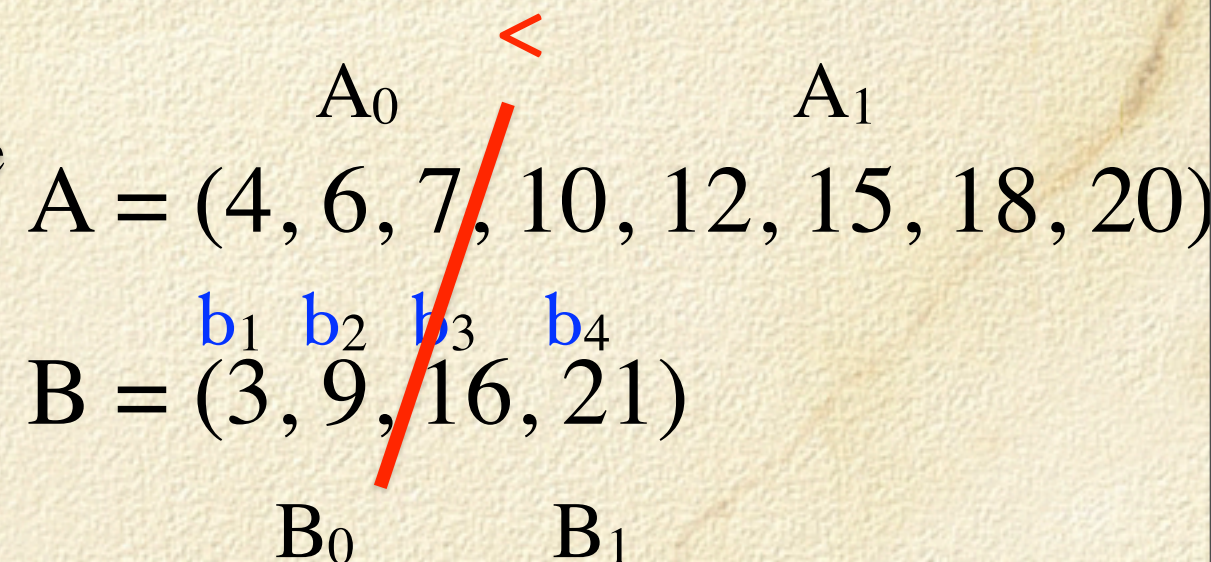
$$(1) |B_i| = \log m$$

$$(2) \sum_i |A_i| = n \text{ und}$$

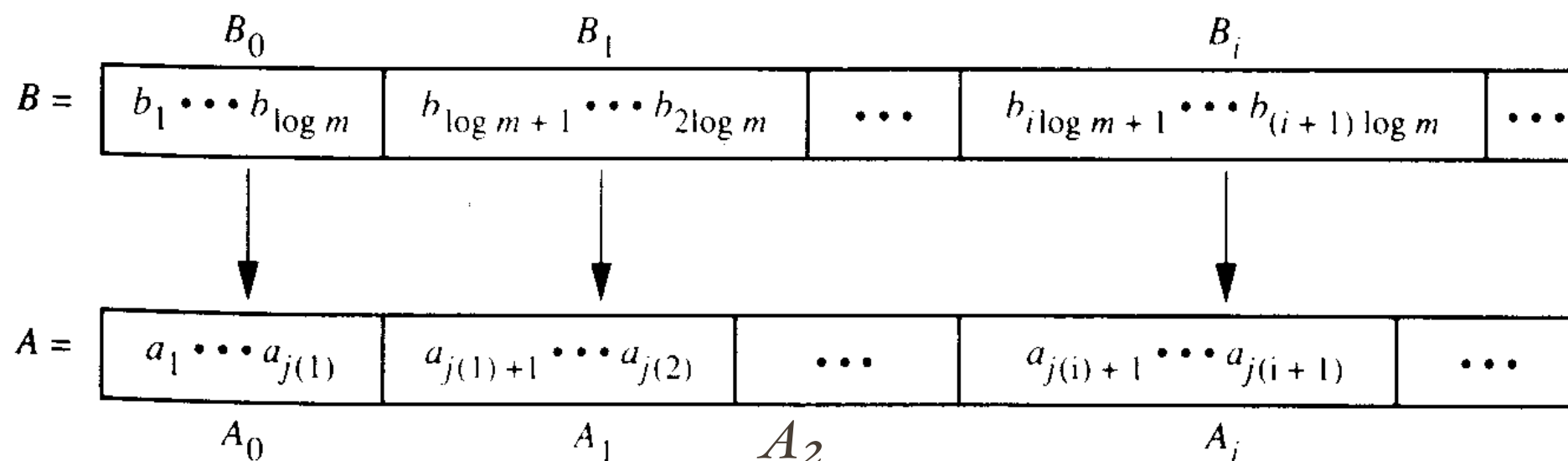
(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$
2. **for** $1 \leq i \leq k(m) - 1$ **pardo**
 - 2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$
3. **for** $0 \leq i \leq k(m) - 1$ **pardo**
 - 3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$
 - 3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)



end



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

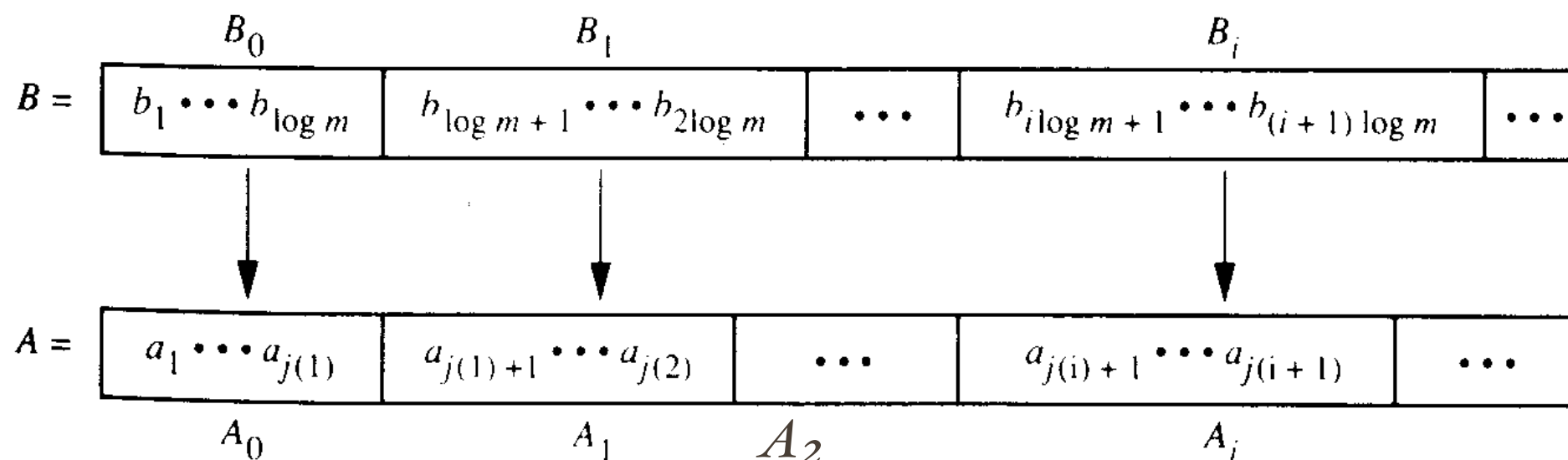
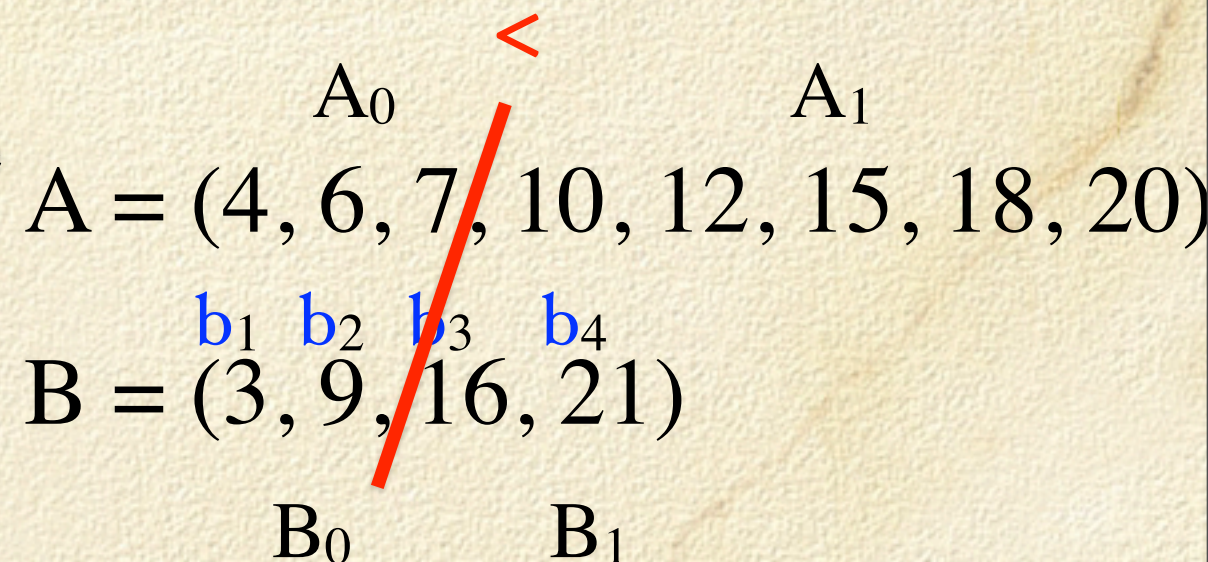
$m = 4$

$k_m = 2$

begin

1. Set $j(0) := 0, j(k(m)) := n$
2. **for** $1 \leq i \leq k(m) - 1$ **pardo**
 - 2.1 Berechne $rank(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = rank(b_{i \log m} : A)$
3. **for** $0 \leq i \leq k(m) - 1$ **pardo**
 - 3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$
 - 3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

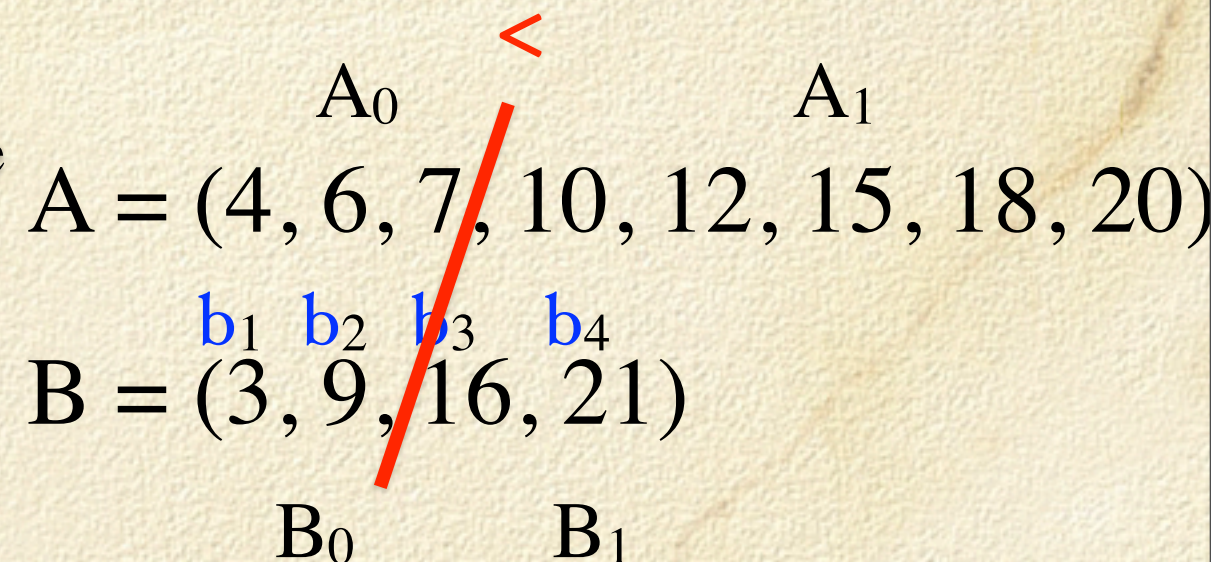
$m = 4$

$k_m = 2$

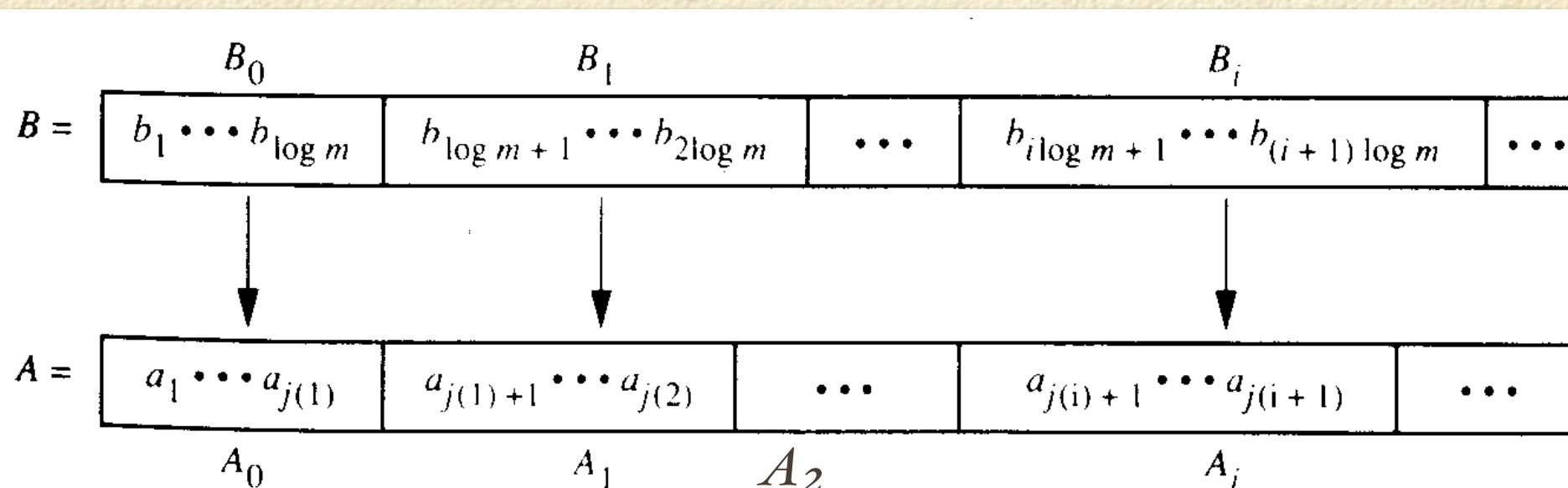
begin

1. Set $j(0) := 0, j(k(m)) := n$
2. **for** $1 \leq i \leq k(m) - 1$ **pardo**
 - 2.1 Berechne $rank(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = rank(b_{i \log m} : A)$
3. **for** $0 \leq i \leq k(m) - 1$ **pardo**
 - 3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$
 - 3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



$i = 2$



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

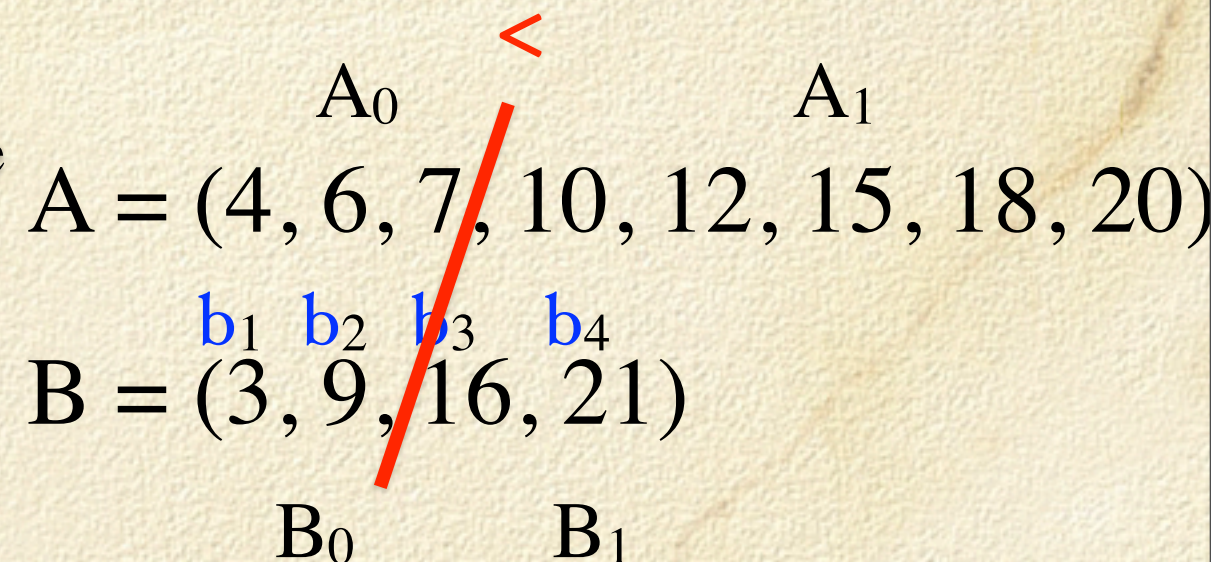
$m = 4$

$k_m = 2$

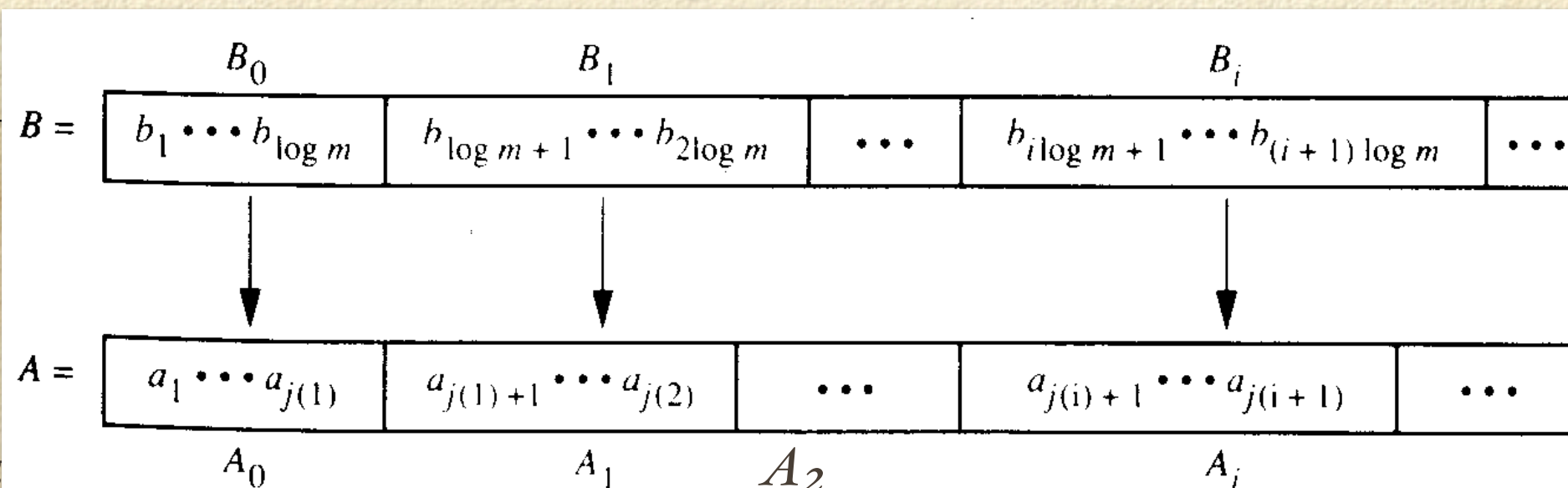
begin

1. Set $j(0) := 0, j(k(m)) := n$
2. **for** $1 \leq i \leq k(m) - 1$ **pardo**
 - 2.1 Berechne $rank(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = rank(b_{i \log m} : A)$
3. **for** $0 \leq i \leq k(m) - 1$ **pardo**
 - 3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$
 - 3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



$i = 2$



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

$$m = 4$$

$$k_m = 2$$

$$(1) |B_i| = \log m$$

$$(2) \sum_i |A_i| = n \text{ und}$$

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

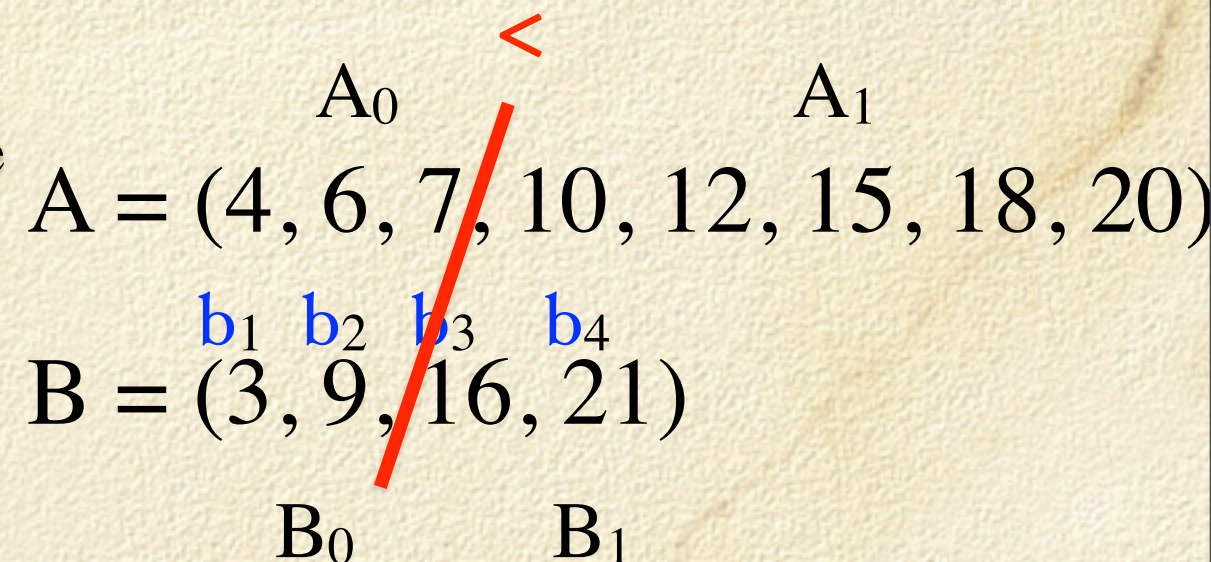
2.1 Berechne $rank(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = rank(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

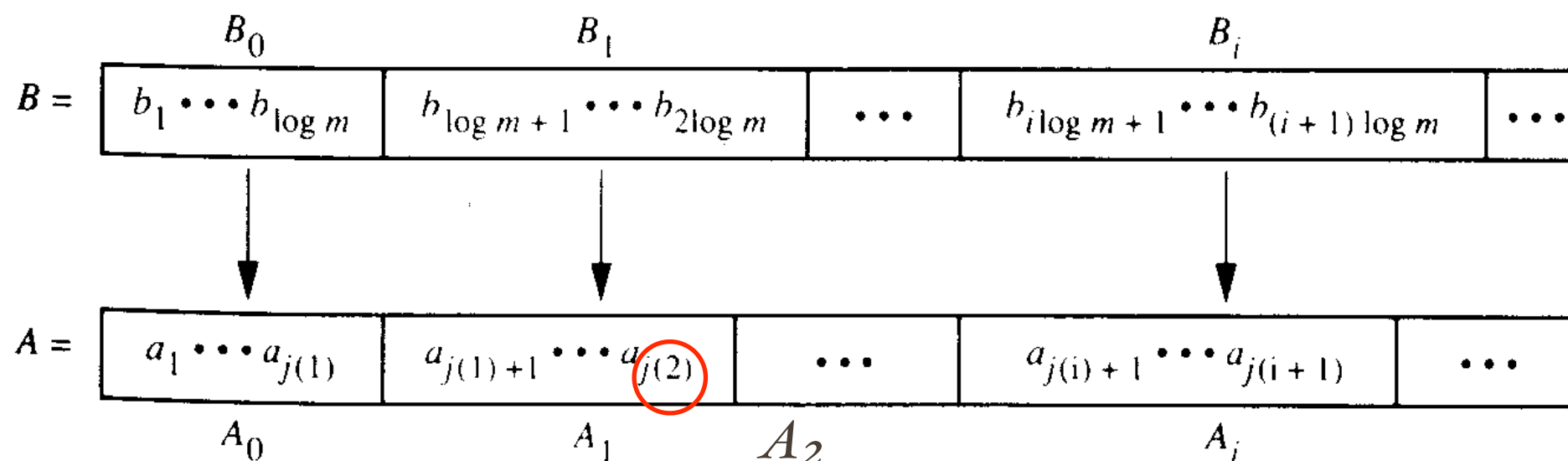
3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



$i = 2$



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

$$m = 4$$

$$k_m = 2$$

$$(1) |B_i| = \log m$$

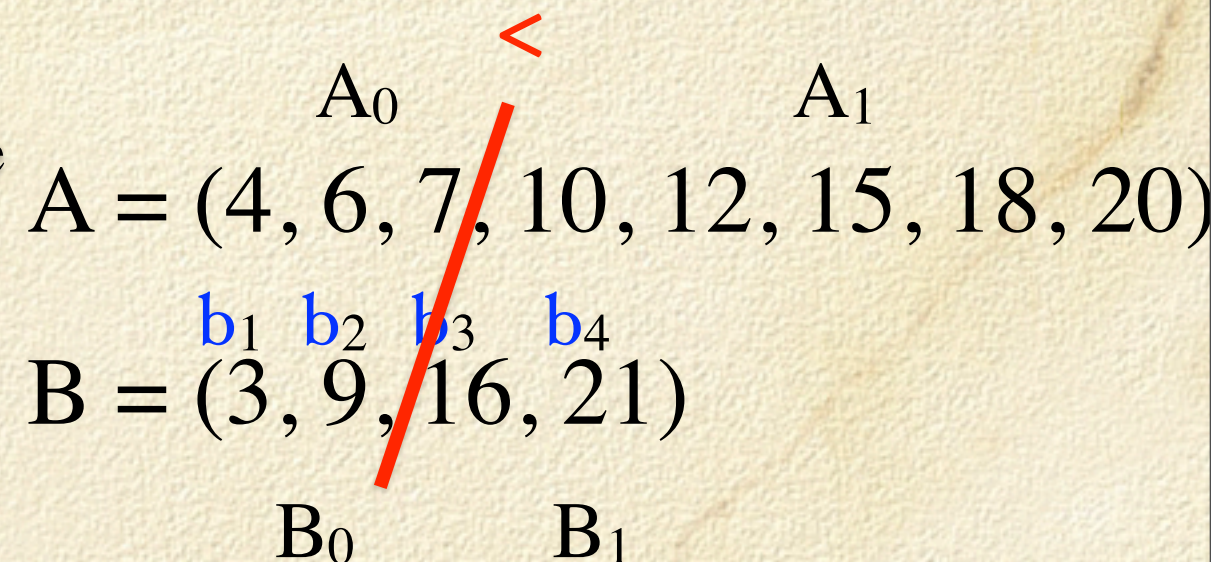
$$(2) \sum_i |A_i| = n \text{ und}$$

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

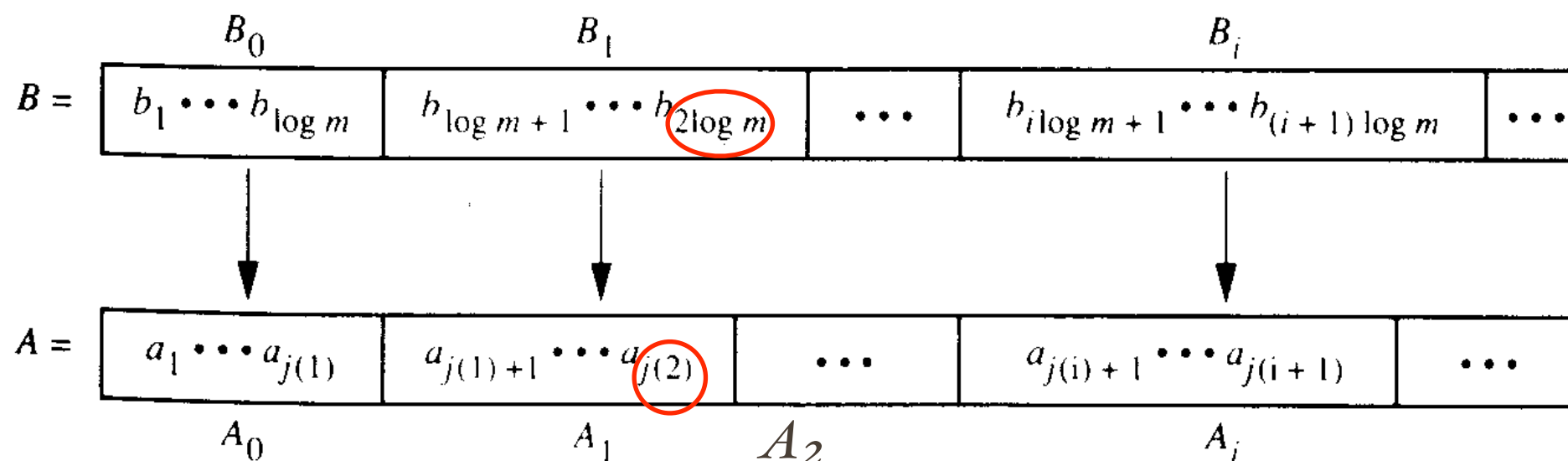
begin

1. Set $j(0) := 0, j(k(m)) := n$
2. **for** $1 \leq i \leq k(m) - 1$ **pardo**
 - 2.1 Berechne $rank(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = rank(b_{i \log m} : A)$
3. **for** $0 \leq i \leq k(m) - 1$ **pardo**
 - 3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$
 - 3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



$$i = 2$$



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

$$m = 4$$

$$k_m = 2$$

$$(1) |B_i| = \log m$$

$$(2) \sum_i |A_i| = n \text{ und}$$

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

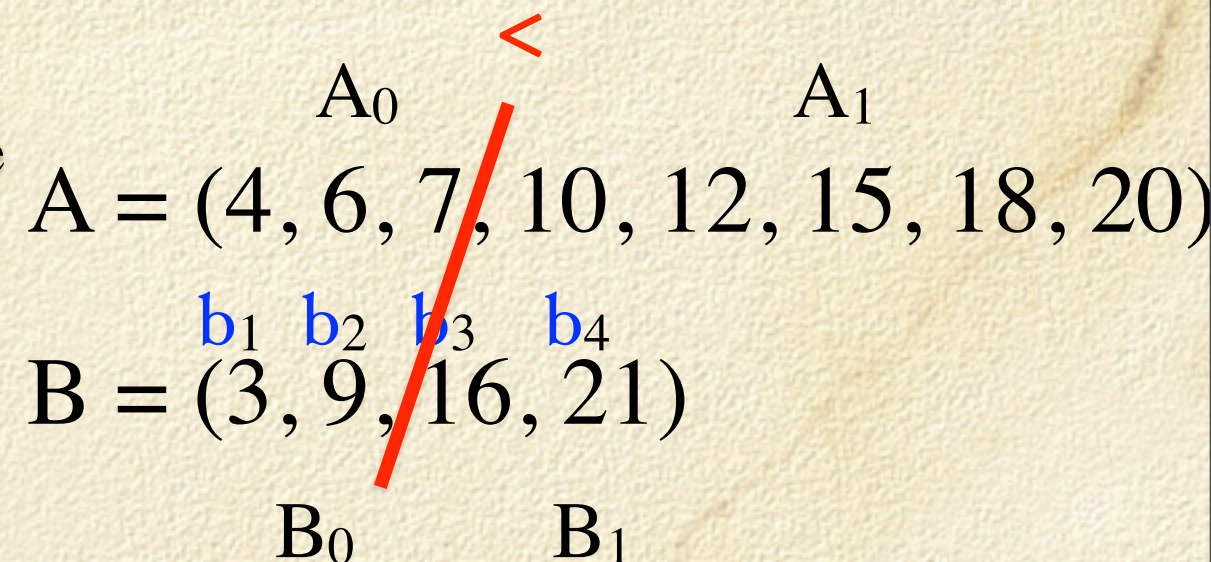
2.1 Berechne $rank(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = rank(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

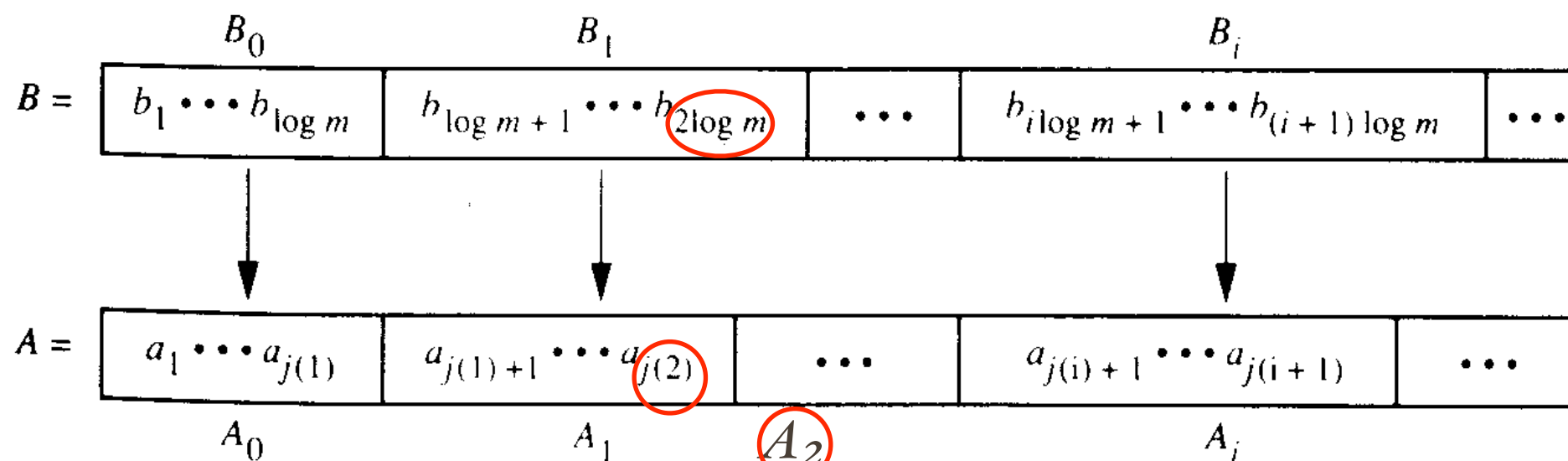
3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



$i = 2$



Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

$$m = 4$$

$$k_m = 2$$

$$(1) |B_i| = \log m$$

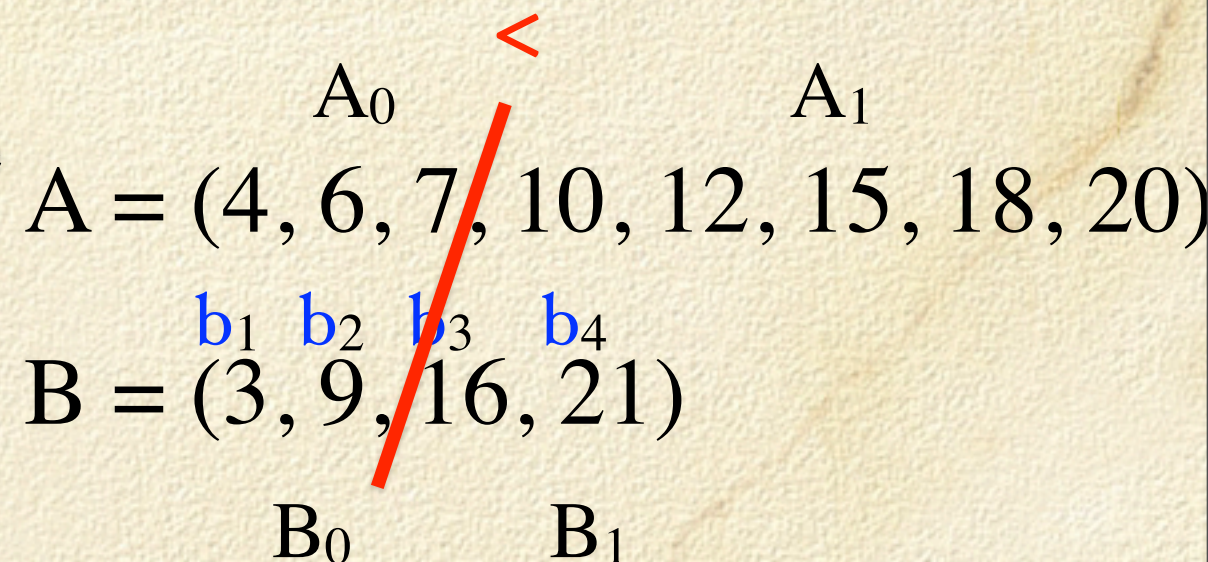
$$(2) \sum_i |A_i| = n \text{ und}$$

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

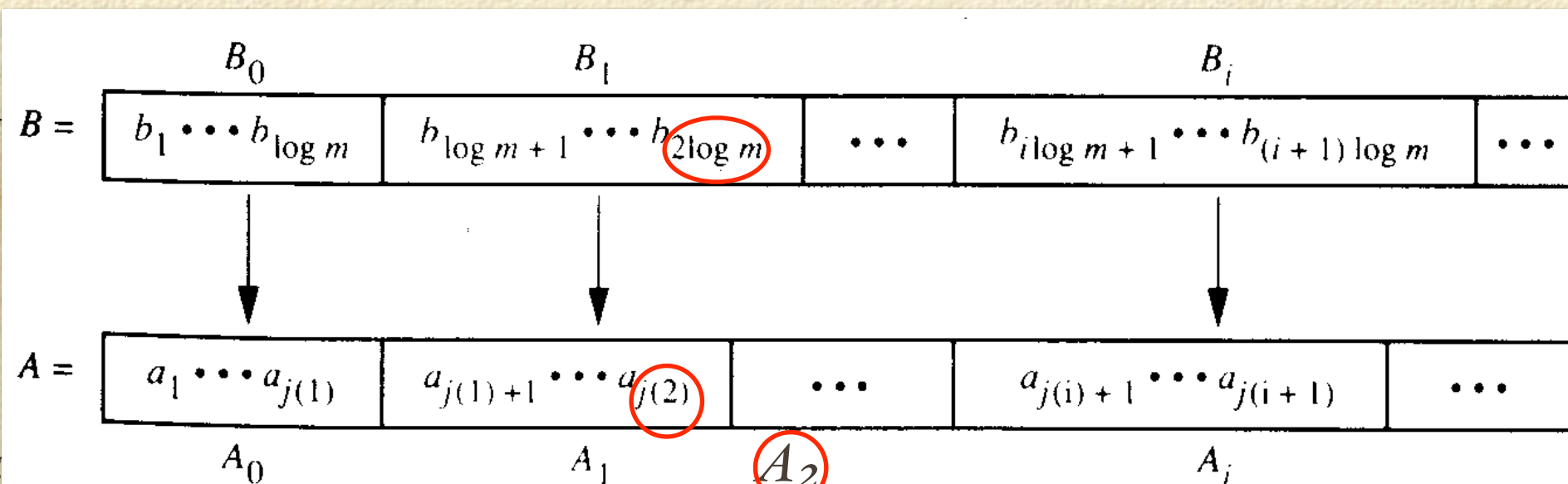
begin

1. Set $j(0) := 0, j(k(m)) := n$
2. **for** $1 \leq i \leq k(m) - 1$ **pardo**
 - 2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$
3. **for** $0 \leq i \leq k(m) - 1$ **pardo**
 - 3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$
 - 3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end



$i = 2$



$$A = \overset{A_0}{(4, 6, 7, 10, 12, 15, 18, 20)} \overset{A_1}{}$$

$$B = \overset{b_1}{(3, 9, 16, 21)} \overset{b_2}{\quad} \overset{b_3}{\quad} \overset{b_4}{\quad} \\ \underset{B_0}{\quad} \underset{B_1}{\quad}$$

$$C = \underset{C_0}{(3, 4, 6, 7, 9, 10, 12, 15, 16, 18, 20, 21)} \underset{C_1}{}$$

$$\begin{array}{c}
 A_0 \qquad \qquad A_1 \\
 A = (4, 6, 7, 10, 12, 15, 18, 20) \\
 \begin{array}{c} b_1 \ b_2 \ b_3 \ b_4 \\
 B = (3, 9, 16, 21) \\
 B_0 \qquad B_1
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 C = (3, 4, 6, 7, 9, 10, 12, 15, 16, 18, 20, 21) \\
 C_0 \qquad \qquad C_1
 \end{array}$$

$$\begin{array}{c}
 A_0 \qquad \qquad \qquad A_1 \\
 A = (4, 6, 7, 10, 12, 15, 18, 20) \\
 \qquad \qquad \qquad b_1 \quad b_2 \quad b_3 \quad b_4 \\
 B = (3, 9, 16, 21) \\
 \qquad \qquad \qquad B_0 \qquad \qquad B_1
 \end{array}$$

$$\begin{array}{c}
 C = (3, 4, 6, 7, 9, 10, 12, 15, 16, 18, 20, 21) \\
 \qquad \qquad \qquad C_0 \qquad \qquad \qquad C_1
 \end{array}$$

$$\begin{array}{c}
 \begin{array}{cc}
 A_0 & A_1 \\
 A = (4, 6, 7, 10, 12, 15, 18, 20)
 \end{array} \\
 \begin{array}{cc}
 b_1 & b_2 & b_3 & b_4 \\
 B = (3, 9, 16, 21)
 \end{array} \\
 \begin{array}{cc}
 B_0 & B_1
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 C = (3, 4, 6, 7, 9, 10, 12, 15, 16, 18, 20, 21) \\
 \begin{array}{cc}
 C_0 & C_1
 \end{array}
 \end{array}$$

Lemma 6.27 Sei C die sortierte Folge, welche man durch Mischen der sortierten Folgen A und B mit den Längen n bzw. m erhält. Dann teilt der Algorithmus 7.1 A und B in Paare von Teilfolgen (A_i, B_i) auf, so dass $|B_i| = O(\log m)$, $\sum_i |A_i| = n$ und $C = (C_0, C_1, \dots)$, wobei C_i die sortierte Folge ist, welche aus A_i und B_i durch Mischen entsteht. Dieser Algorithmus benötigt in $O(\log n)$ Zeit mit insgesamt $O(n + m)$ Operationen.

$$\begin{array}{c}
 \begin{array}{cc}
 A_0 & A_1 \\
 A = (4, 6, 7, 10, 12, 15, 18, 20)
 \end{array} \\
 \begin{array}{cc}
 b_1 & b_2 & b_3 & b_4 \\
 B = (3, 9, 16, 21) \\
 B_0 & B_1
 \end{array}
 \end{array}$$

$$\begin{array}{c}
 C = (3, 4, 6, 7, 9, 10, 12, 15, 16, 18, 20, 21) \\
 C_0 \qquad C_1
 \end{array}$$

Lemma 6.27 Sei C die sortierte Folge, welche man durch Mischen der sortierten Folgen A und B mit den Längen n bzw. m erhält. Dann teilt der Algorithmus 7.1 A und B in Paare von Teilfolgen (A_i, B_i) auf, so dass $|B_i| = O(\log m)$, $\sum_i |A_i| = n$ und $C = (C_0, C_1, \dots)$, wobei C_i die sortierte Folge ist, welche aus A_i und B_i durch Mischen entsteht. Dieser Algorithmus benötigt in $O(\log n)$ Zeit mit insgesamt $O(n + m)$ Operationen.

warum?

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche
und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

*Zeitkomplexität:
 $T(n) = ?$*

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche
und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

*Zeitkomplexität:
 $T(n) = ?$*

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche
und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität:
 $T(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche
und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität:
 $T(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität:
 $T(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche
und setze $j(i) = \text{rank}(b_{i \log m} : A)$

$O(\log n)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität:
 $T(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

$O(\log n)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität:
 $T(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

$O(\log n)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

$O(1)$

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität:
 $T(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

$O(\log n)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m});$

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

$O(1)$

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität:
 $T(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

$O(\log n)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

$O(1)$

end

$O(\log n)$

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(\log n)$

$O(1)$

$O(\log n)$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(1)$

$O(\log n)$

$O(1)$

$O(\log n)$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(1)$

$O(\log n)$

$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right)$

$O(1)$

$O(\log n)$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(1)$

$O(\log n)$

$$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right) = O(m + n)$$

$O(1)$

$O(\log n)$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(1)$

$O(\log n)$

$$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right) = O(m + n)$$

$O(1)$

$O(\log n)$

$$\text{da } \left(\frac{m \log n}{\log m}\right) < \left(\frac{m \log(n+m)}{\log m}\right) \leq n + m \text{ für } n, m \geq 4$$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(1)$

$O(\log n)$

$$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right) = O(m + n)$$

$O(1)$

$O(m)$

$O(\log n)$

$$\text{da } \left(\frac{m \log n}{\log m}\right) < \left(\frac{m \log(n+m)}{\log m}\right) \leq n + m \text{ für } n, m \geq 4$$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(1)$

$O(\log n)$

$$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right) = O(m + n)$$

$O(1)$

$O(m)$

$O(\log n)$

$$\text{da } \left(\frac{m \log n}{\log m}\right) < \left(\frac{m \log(n+m)}{\log m}\right) \leq n + m \text{ für } n, m \geq 4$$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sind.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$$k_m = \frac{m}{\log m} \in \mathbb{N}$$

So viele Blöcke!!

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

$O(1)$

$O(1)$

$O(\log n)$

$$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right) = O(m + n)$$

$O(1)$

$O(m)$

$O(\log n)$

$O(m + n)$

$$\text{da } \left(\frac{m \log n}{\log m}\right) < \left(\frac{m \log(n+m)}{\log m}\right) \leq n + m \text{ für } n, m \geq 4$$

Mit diesem Algorithmus haben wir das Mischproblem der Größe n auf Unterprobleme kleinerer Größe reduziert.

optimaler Algorithmus für Mischen:

- wende Algorithmus **6.1** an

Mit diesem Algorithmus haben wir das Mischproblem der Größe n auf Unterprobleme kleinerer Größe reduziert.

optimaler Algorithmus für Mischen:

- wende Algorithmus **6.1** an
- behandle die Paare (A_i, B_i) getrennt (es gilt $|B_i| = \log n$) parallel

Mit diesem Algorithmus haben wir das Mischproblem der Größe n auf Unterprobleme kleinerer Größe reduziert.

optimaler Algorithmus für Mischen:

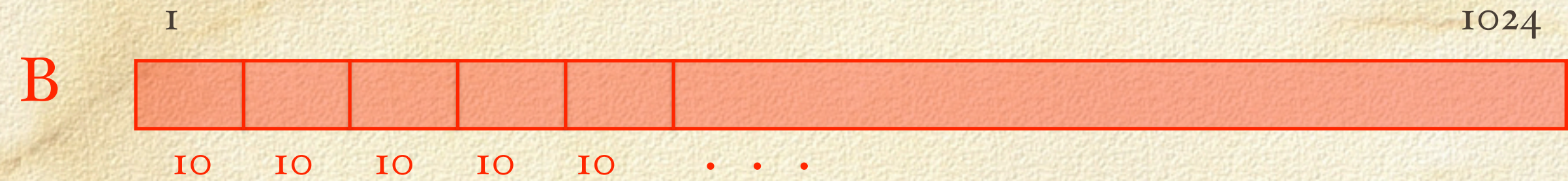
- wende Algorithmus **6.1** an
- behandle die Paare (A_i, B_i) getrennt (es gilt $|B_i| = \log n$) parallel
- falls $|A_i| \leq c \log n$ mische (A_i, B_i) in $O(\log n)$ Zeit mit einem sequentiellen Mischalgorithmus

Mit diesem Algorithmus haben wir das Mischproblem der Größe n auf Unterprobleme kleinerer Größe reduziert.

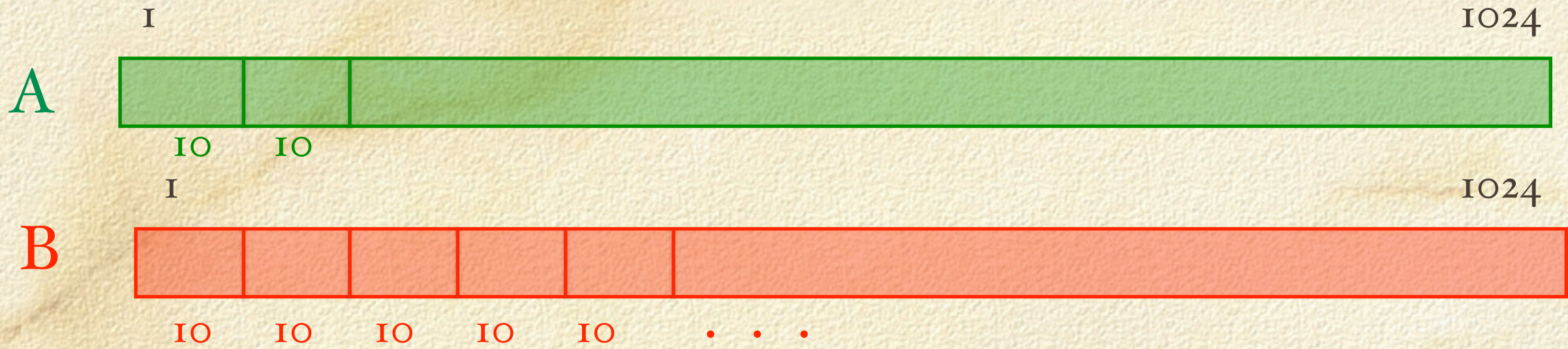
optimaler Algorithmus für Mischen:

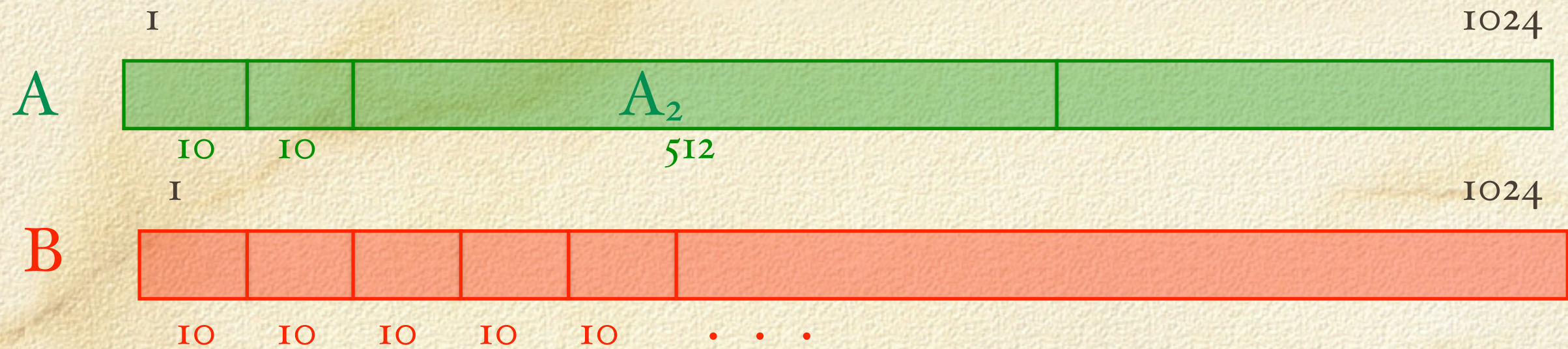
- wende Algorithmus **6.1** an
- handle die Paare (A_i, B_i) getrennt (es gilt $|B_i| = \log n$) parallel
- falls $|A_i| \leq c \log n$ mische (A_i, B_i) in $O(\log n)$ Zeit mit einem sequentiellen Mischalgorithmus
- falls $|A_i| \not\leq c \log n$, dann wende Algorithmus **6.1** an, um A_i in Blöcke der Länge $\log n$ zu zerlegen. Dazu werden $O(\log \log n)$ Zeit und $O(|A_i|)$ Operationen benötigt.

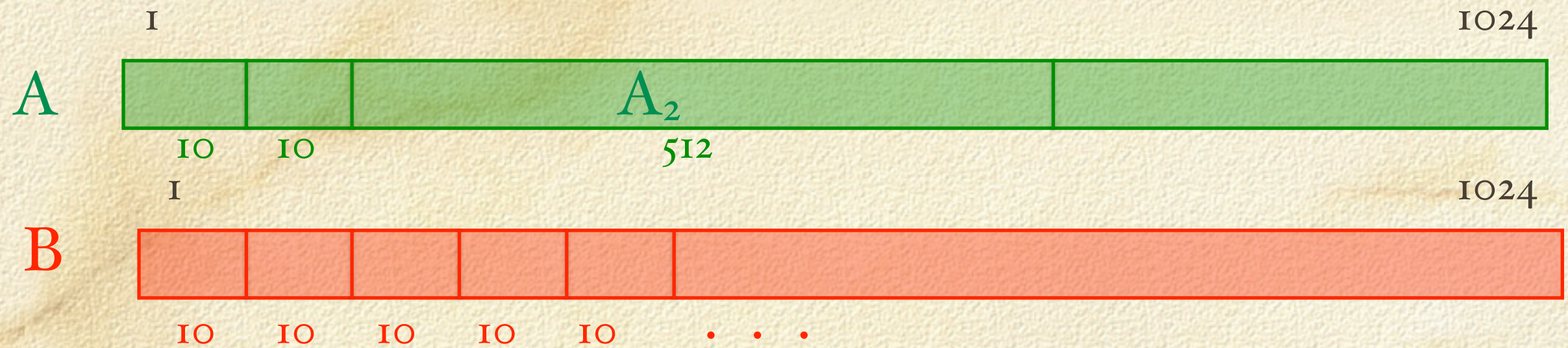






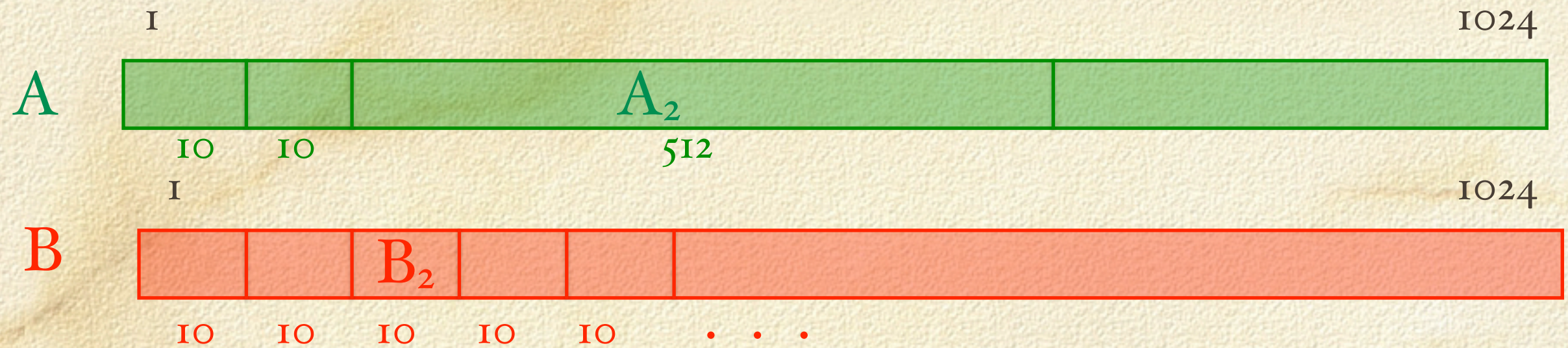






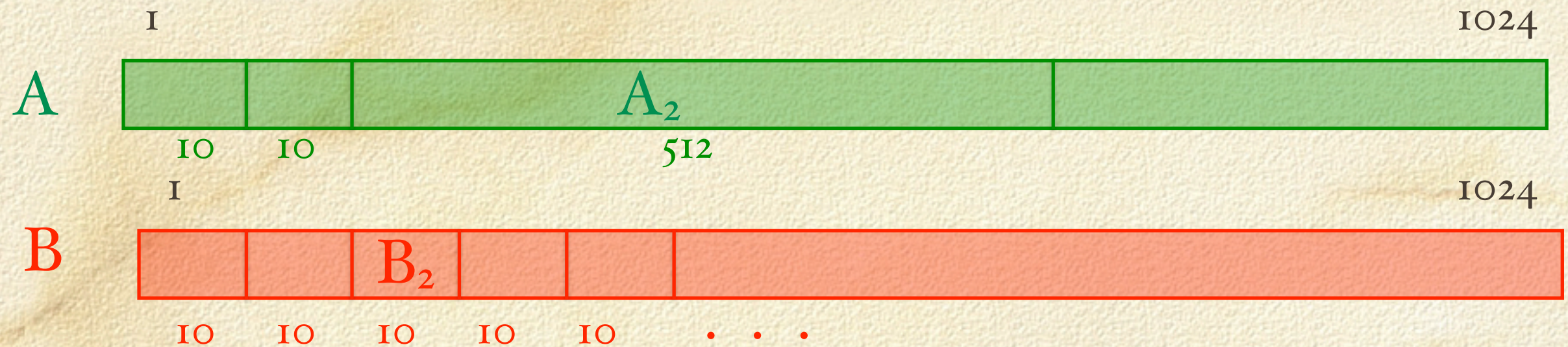
Zeit: $O(\log n)$

Operationen: $O(n)$



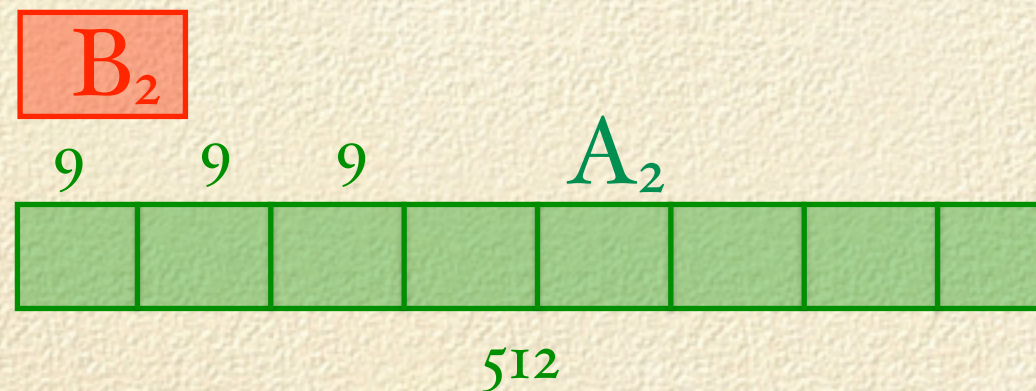
Zeit: $O(\log n)$

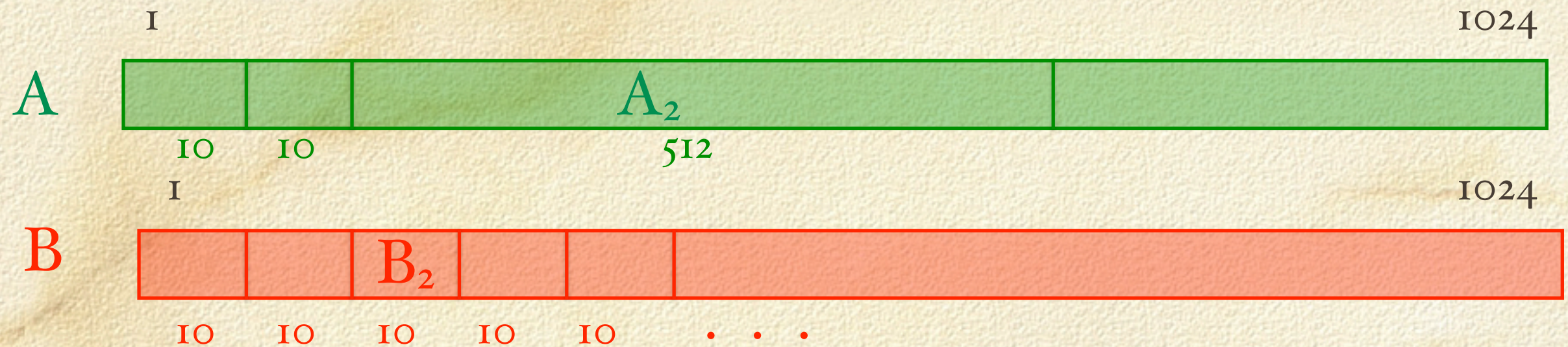
Operationen: $O(n)$



Zeit: $O(\log n)$

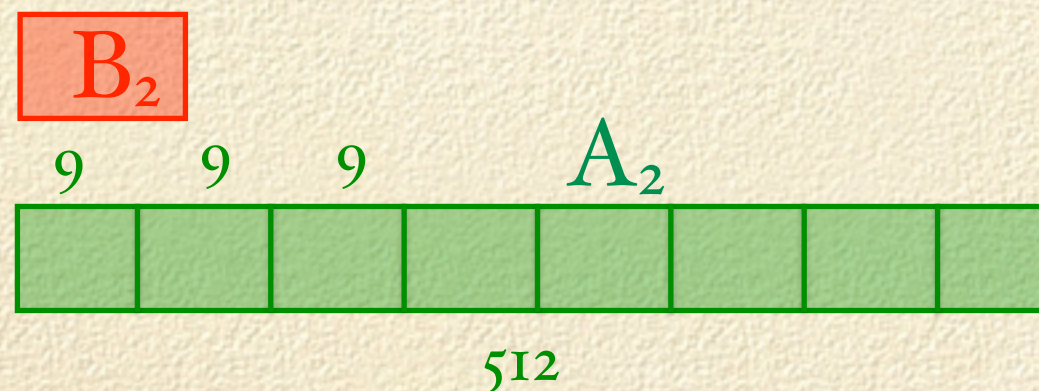
Operationen: $O(n)$



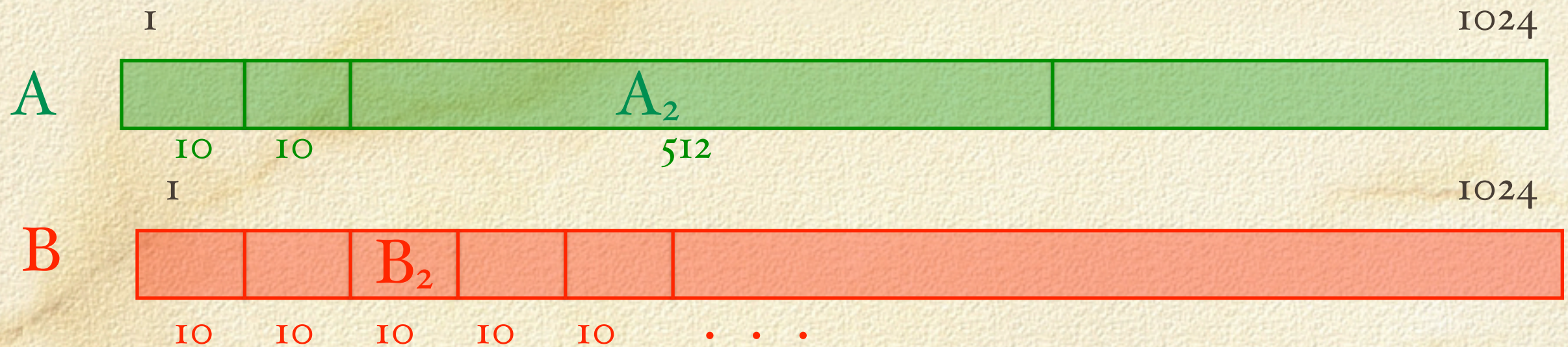


Zeit: $O(\log n)$

Operationen: $O(n)$

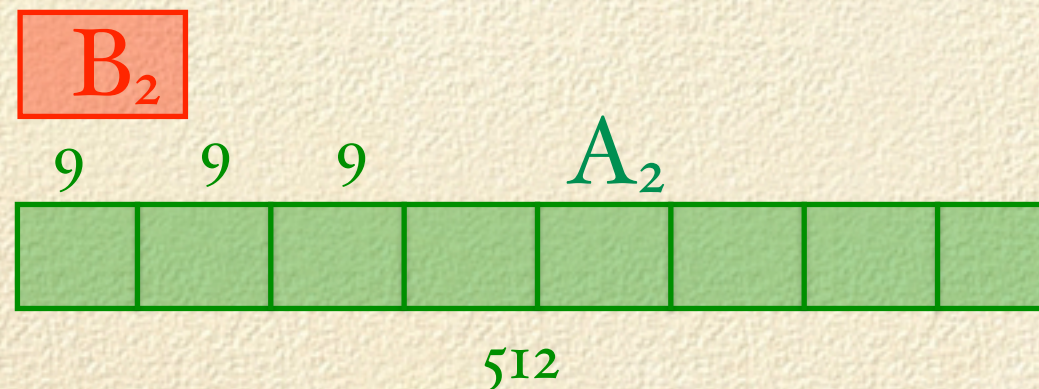


parallele Zeit: $O(\log |B_2|) = O(\log \log n)$



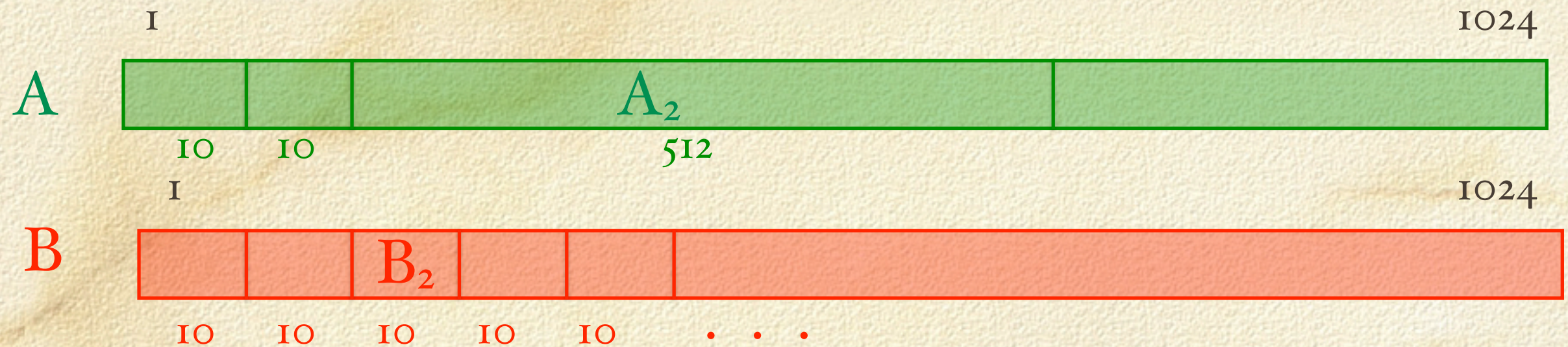
Zeit: $O(\log n)$

Operationen: $O(n)$



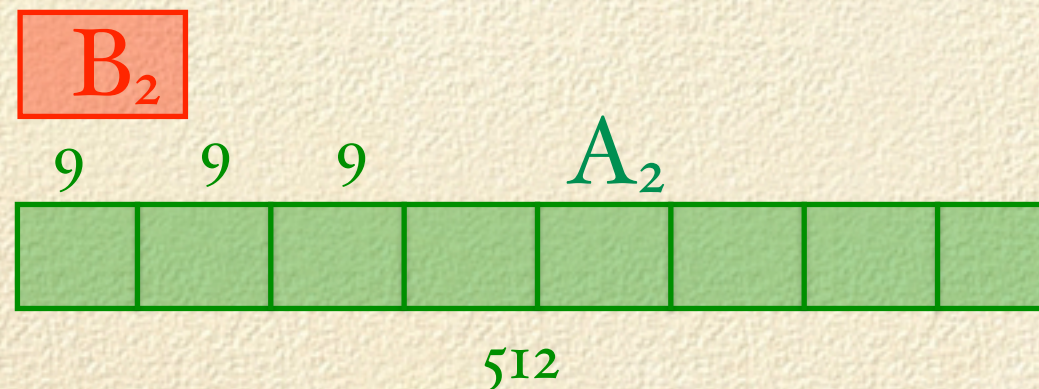
parallele Zeit: $O(\log |B_2|) = O(\log \log n)$

Operationen: $O(|A_2|)$



Zeit: $O(\log n)$

Operationen: $O(n)$



parallele Zeit: $O(\log |B_2|) = O(\log \log n)$

Operationen: $O(|A_2|)$

insgesamt:

$$T(n) = O(\log n)$$

$$W(n) = O(n)$$

Insgesamt ergibt sich:

Satz 6.28 *Das Mischen zweier Folgen A und B der Länge n ist von einem parallelen Algorithmus in $O(\log n)$ Zeit mit $O(n)$ Operationen durchführbar.*

Insgesamt ergibt sich:

Satz 6.28 *Das Mischen zweier Folgen A und B der Länge n ist von einem parallelen Algorithmus in $O(\log n)$ Zeit mit $O(n)$ Operationen durchführbar.*

noch schnellerer Algorithmus?

Insgesamt ergibt sich:

Satz 6.28 *Das Mischen zweier Folgen A und B der Länge n ist von einem parallelen Algorithmus in $O(\log n)$ Zeit mit $O(n)$ Operationen durchführbar.*

noch schnellerer Algorithmus?

Wo ist noch Potenzial für Beschleunigung?

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

$O(1)$

$O(1)$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

$O(\log n)$

$$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right) = O(m + n)$$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

$O(1)$

$O(m)$

end

$O(\log n)$

$O(m + n)$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$O(1)$

$O(\log n)$

$O\left(\frac{\log(n+1)}{\log(p+1)}\right)$

$O(1)$

$O(\log n)$

hier !!

$O(1)$

$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right)$
 $= O(m + n)$

$O(m)$

$O(m + n)$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$O(1)$

$O(1)$

$O(\log n)$

$O\left(\frac{\log(n+1)}{\log(p+1)}\right)$

$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right)$
 $= O(m + n)$

$O(1)$

$O(m)$

$O(\log n)$

$O(m + n)$

hier !!

Paralleles Suchen mit p Prozessoren

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$O(1)$

$O(1)$

$O(\log n)$

$O\left(\frac{\log(n+1)}{\log(p+1)}\right)$

$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right)$
 $= O(m + n)$

$O(1)$

$O(m)$

$O(\log n)$

$O(m + n)$

hier !!

Paralleles Suchen mit p Prozessoren

... führt zu einem Mischalgorithmus mit $T(n) = O(\log \log n)$

Algorithmus 6.1 (Partitionieren)

Eingabe: Zwei Felder $A = (a_1, \dots, a_n)$ und $B = (b_1, \dots, b_m)$ in aufsteigender Reihenfolge sortiert, wobei $\log m$ und $k(m) = \frac{m}{\log m}$ ganze Zahlen sein müssen.

Ausgabe: $k(m)$ Paare (A_i, B_i) von Teilfolgen von A und B , so dass

(1) $|B_i| = \log m$

(2) $\sum_i |A_i| = n$ und

(3) jedes Element von A_i und B_i ist größer als jedes Element von A_{i-1} oder B_{i-1} für alle $1 \leq i \leq k(m) - 1$.

Zeitkomplexität
 $T(n) = ?$

Operationenkomplexität
 $W(n) = ?$

begin

1. Set $j(0) := 0, j(k(m)) := n$

2. **for** $1 \leq i \leq k(m) - 1$ **pardo**

2.1 Berechne $\text{rank}(b_{i \log m} : A)$ durch binäre Suche und setze $j(i) = \text{rank}(b_{i \log m} : A)$

3. **for** $0 \leq i \leq k(m) - 1$ **pardo**

3.1 $B_i := (b_{i \log m + 1}, \dots, b_{(i+1) \log m})$;

3.2 $A_i := (a_{j(i)+1}, \dots, a_{j(i+1)})$
(A_i ist leer, wenn $j(i) = j(i+1)$ gilt.)

end

$O(1)$

$O(1)$

$O(\log n)$

$O\left(\frac{\log(n+1)}{\log(p+1)}\right)$

$O\left((\log n) \times \left(\frac{m}{\log m}\right)\right)$
 $= O(m + n)$

$O(1)$

$O(m)$

$O(\log n)$

$O(m + n)$

hier !!

Paralleles Suchen mit p Prozessoren

... führt zu einem Mischalgorithmus mit $T(n) = O(\log \log n)$

und $W(n) = O(n \log \log n)$

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Zeitkomplexität

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Zeitkomplexität $T(n) =$

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Zeitkomplexität $T(n) = O(\log n)$

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Zeitkomplexität $T(n) = O(\log n)$

paralleles Suchen mit p Prozessoren:

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Zeitkomplexität $T(n) = O(\log n)$

paralleles Suchen mit p Prozessoren: $T(n) =$

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Zeitkomplexität $T(n) = O(\log n)$

paralleles Suchen mit p Prozessoren: $T(n) = O\left(\frac{\log(n+1)}{\log(p+1)}\right)$

Paralleles Suchen

- *Gegeben:* Eine linear geordnete Folge $X = (x_1, \dots, x_n)$ von verschiedenen Elementen einer linear geordneten Menge $(S, \leq)$
- $y \in S$
- *Suchproblem:* Finde Index $i \in \{0, 1, \dots, n\}$ mit $x_i \leq y < x_{i+1}$

binäres Suchen:

Zeitkomplexität $T(n) = O(\log n)$

paralleles Suchen mit p Prozessoren: $T(n) = O\left(\frac{\log(n+1)}{\log(p+1)}\right)$

Der Algorithmus ist optimal, wenn p konstant ist.

Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

```
1. if( $j=1$ ) then do
    1.1 Set  $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$ 
    1.2. Set  $c_0 := 0; c_{p+1} := 1$ 
2. while( $r - l > p$ ) do
    2.1. if( $j=1$ ) then {set  $q_0 := l; q_{p+1} := r$ }
    2.2. Set  $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$ 
    2.3. if( $y = x_{q_j}$ ) then {return( $q_j$ ); exit}
        else {set  $c_j := 0$  if  $y > x_{q_j}$  and  $c_j := 1$ 
            if  $y < x_{q_j}$ }
    2.4. if( $c_j < c_{j+1}$ ) then {set  $l := q_j; r := q_{j+1}$ }
    2.5. if( $j = 1$  and  $c_0 < c_1$ ) then {set  $l := q_0; r := q_1$ }
3. if( $j \leq r - l$ ) then do
    3.1. Case statement:
         $y = x_{l+j}$ : {return( $l + j$ ); exit}
         $y > x_{l+j}$ : set  $c_j := 0$ 
         $y < x_{l+j}$ : set  $c_j := 1$ 
    3.2. if( $c_{j-1} < c_j$ ) then return( $l + j - 1$ )
```

end

Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end

X_0

X_{n+1}

Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

```
1. if(j=1) then do
    1.1 Set  $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$ 
    1.2. Set  $c_0 := 0; c_{p+1} := 1$ 
2. while( $r - l > p$ ) do
    2.1. if(j=1) then {set  $q_0 := l; q_{p+1} := r$ }
    2.2. Set  $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$ 
    2.3. if( $y = x_{q_i}$ ) then {return( $q_j$ ); exit}
        else {set  $c_j := 0$  if  $y > x_{q_j}$  and  $c_j := 1$ 
              if  $y < x_{q_j}$ }
    2.4. if( $c_j < c_{j+1}$ ) then {set  $l := q_j; r := q_{j+1}$ }
    2.5. if( $j = 1$  and  $c_0 < c_1$ ) then {set  $l := q_0; r := q_1$ }
3. if( $j \leq r - l$ ) then do
    3.1. Case statement:
         $y = x_{l+j}$ : {return( $l + j$ ); exit}
         $y > x_{l+j}$ : set  $c_j := 0$ 
         $y < x_{l+j}$ : set  $c_j := 1$ 
    3.2. if( $c_{j-1} < c_j$ ) then return( $l + j - 1$ )
```

end

X_0

X_{n+1}

Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end

X_0

X_{n+1}

Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end

X_0

X_{n+1}

Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end

X_0

$+\infty$

X_{n+1}

Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

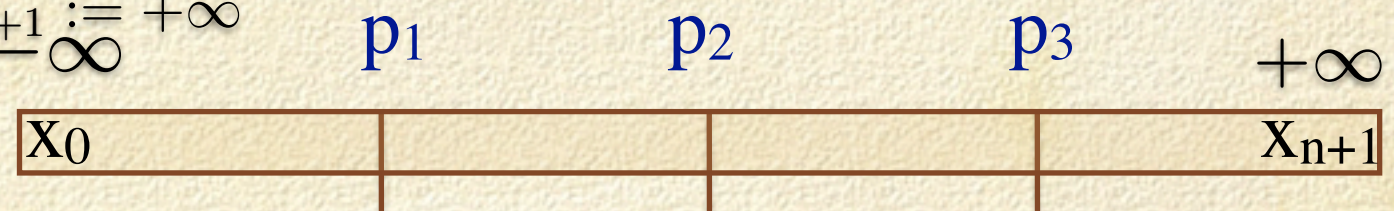
$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end



Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

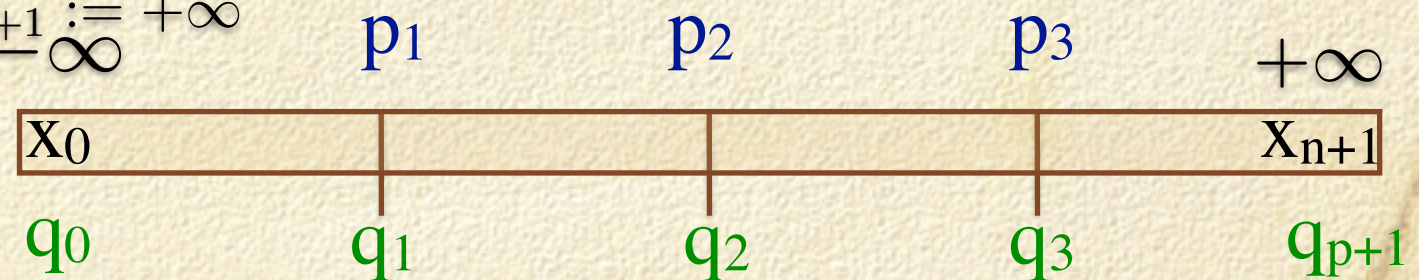
$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end



Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

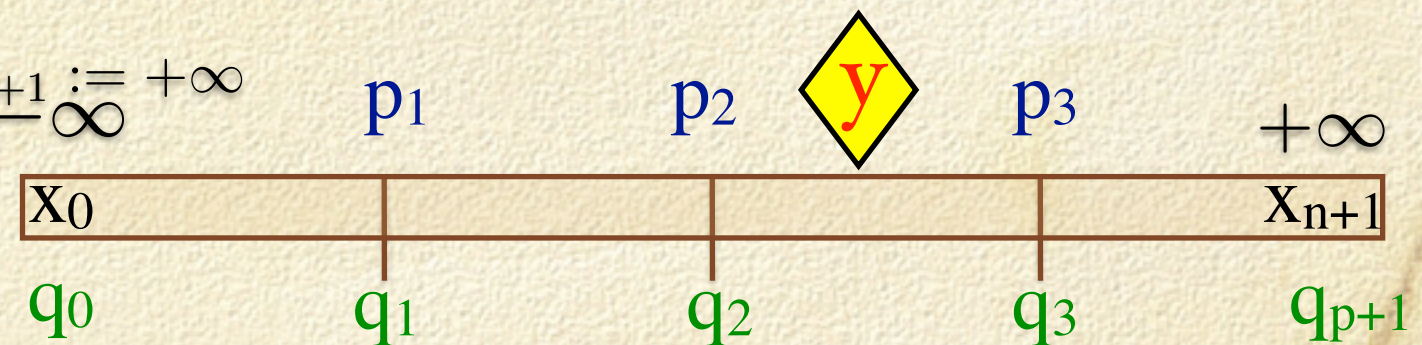
$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end



Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

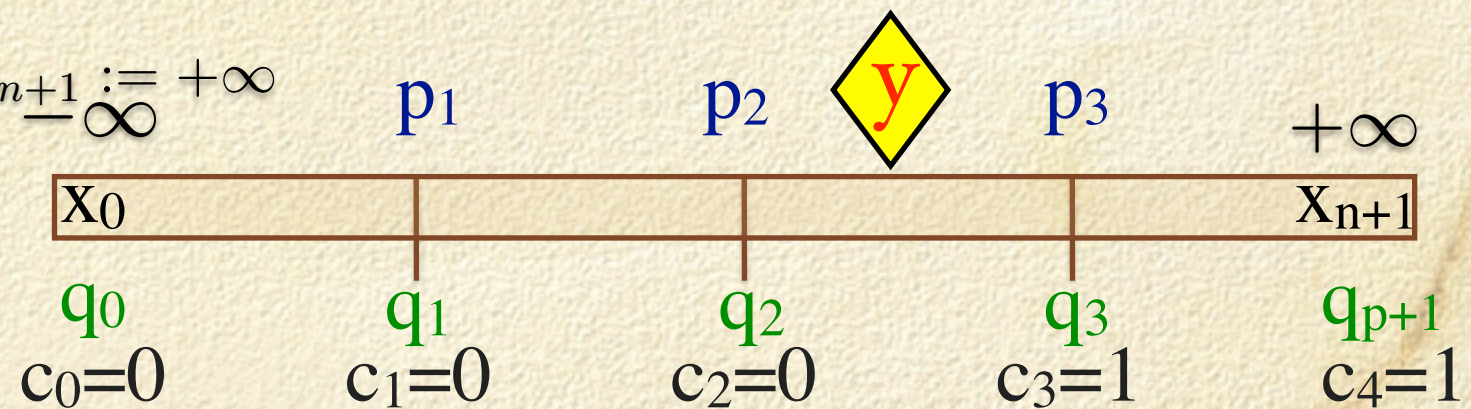
$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end



Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}

else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

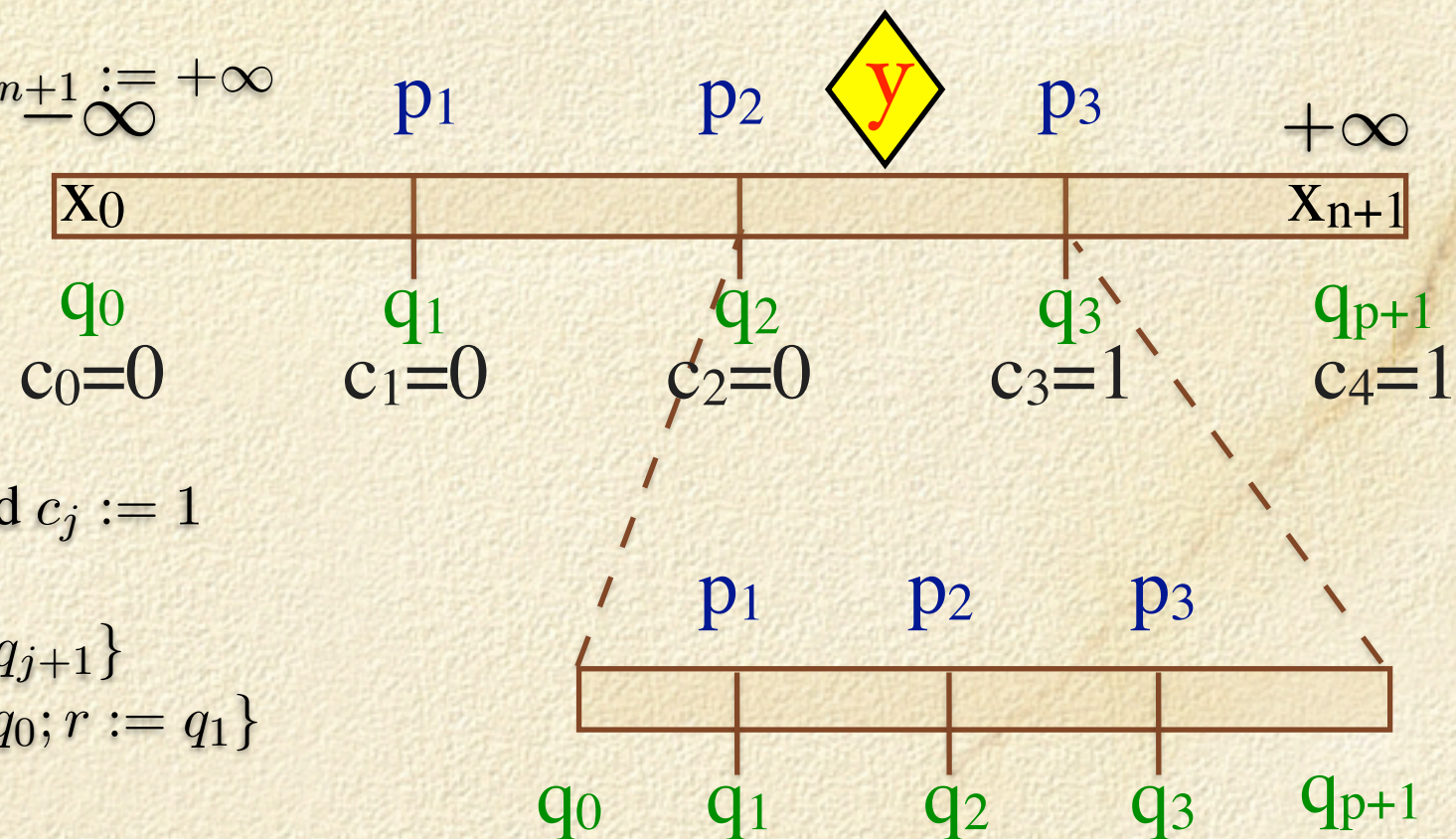
$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end



Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

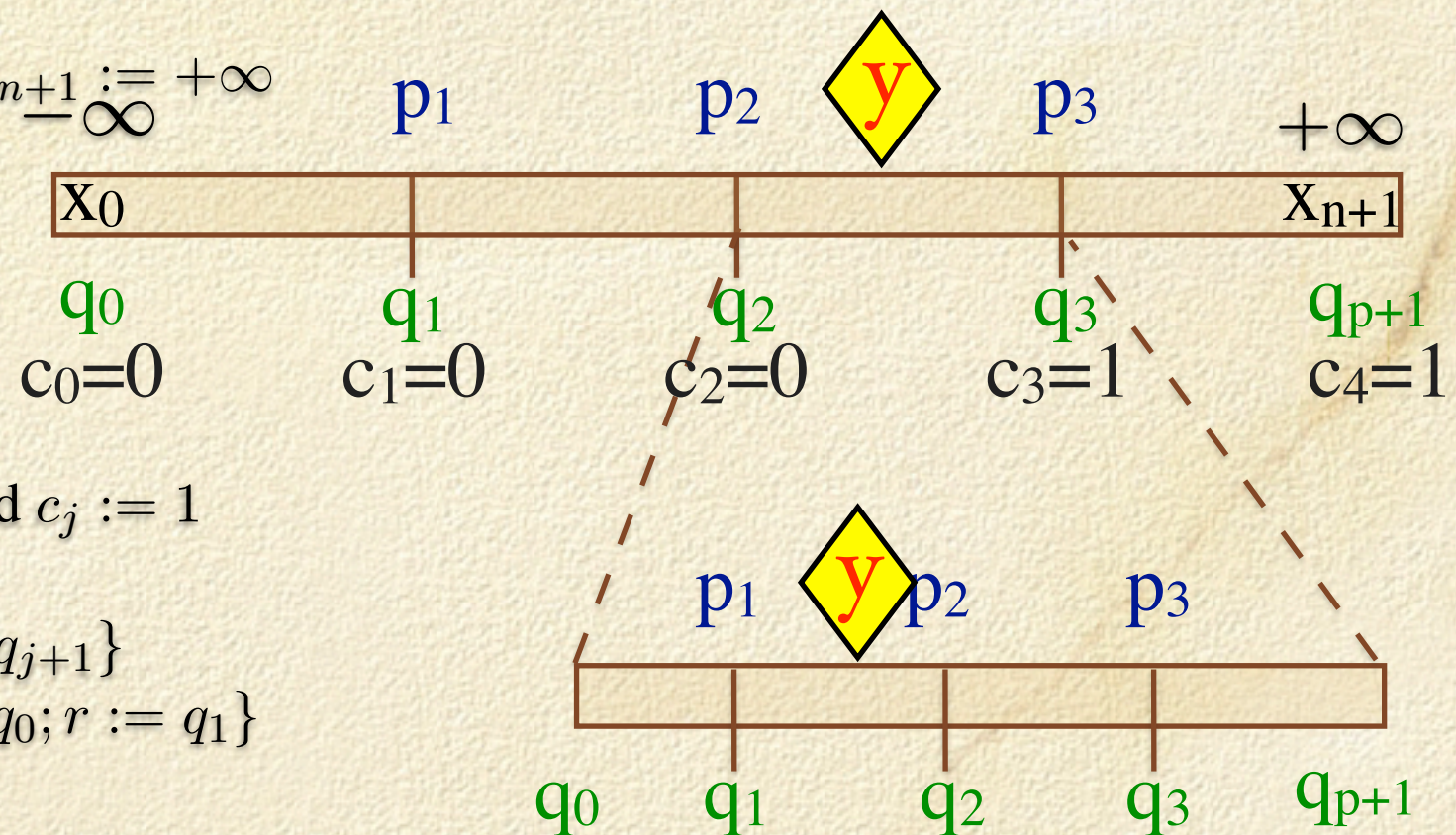
$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end



Algorithmus 6.2 (Paralleles Suchen für Prozessor P_j)

Eingabe: (1) Ein Feld $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$
(2) ein Element y
(3) die Anzahl p der Prozessoren mit $p \leq n$
(4) die Prozessornummer j mit $1 \leq j \leq p$

Ausgabe: ein Index i mit $x_i \leq y < x_{i+1}$

begin

3 Prozessoren: $p = 3$

1. **if**($j=1$) **then do**

1.1 Set $l := 0; r := n + 1; x_0 := -\infty; x_{n+1} := +\infty$

1.2. Set $c_0 := 0; c_{p+1} := 1$

2. **while**($r - l > p$) **do**

2.1. **if**($j=1$) **then** {set $q_0 := l; q_{p+1} := r$ }

2.2. Set $q_j := l + j \lfloor \frac{r-l}{p+1} \rfloor$

2.3. **if**($y = x_{q_i}$) **then** {**return**(q_j); **exit**}
 else {set $c_j := 0$ if $y > x_{q_j}$ and $c_j := 1$
 if $y < x_{q_j}$ }

2.4. **if**($c_j < c_{j+1}$) **then** {set $l := q_j; r := q_{j+1}$ }

2.5. **if**($j = 1$ and $c_0 < c_1$) **then** {set $l := q_0; r := q_1$ }

3. **if**($j \leq r - l$) **then do**

3.1. Case statement:

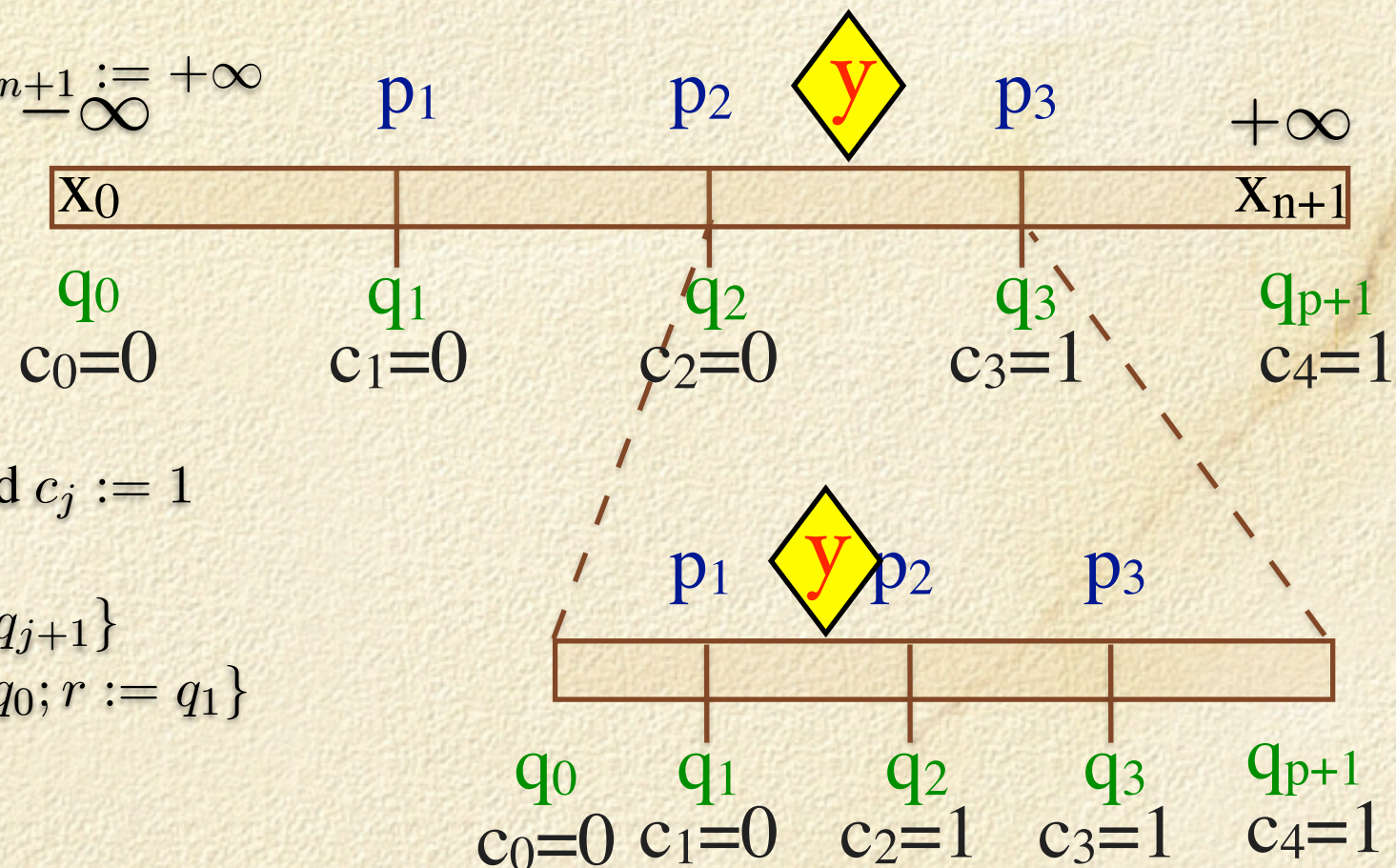
$y = x_{l+j}$: {**return**($l + j$); **exit**}

$y > x_{l+j}$: set $c_j := 0$

$y < x_{l+j}$: set $c_j := 1$

3.2. **if**($c_{j-1} < c_j$) **then return**($l + j - 1$)

end



paralleles Suchen mit $p \leq n$ Prozessoren: Prozessor P_1 : zuständig für Initialisierung und Randsingularitäten. Aufteilen von X in $p + 1$ Blöcke von etwa gleicher Länge. Jede parallele Runde findet $x_i = y$ oder einen y enthaltenden Block.

$n = 15$

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
$-\infty$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	$+\infty$

$p = 2$ Prozessoren

$y=19$

$n = 15$

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
$-\infty$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	$+\infty$

n = 15

[illegible]

p = 2 Prozessoren

y=19

n = 15

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
$-\infty$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	$+\infty$
ing $l=0$ $c_0=0$																

p = 2 Prozessoren

y=19

n = 15

	x ₀	x ₁	x ₂	x ₃	x ₄	x ₅	x ₆	x ₇	x ₈	x ₉	x ₁₀	x ₁₁	x ₁₂	x ₁₃	x ₁₄	x ₁₅	x ₁₆
	-∞	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	+∞
Initialisierung																	
1. Iteration																	
2. Iteration																	

Initialisierung

1. Iteration

2. Iteration

p = 2 Prozessoren

y=19

n = 15

x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}	x_{12}	x_{13}	x_{14}	x_{15}	x_{16}
$-\infty$	2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	$+\infty$

Initialisierung

$l=0$
 $c_0=0$

$r=n+1=16$
 $c_3=1$

1. Iteration

$q_0=0$

$q_1=5$
 $c_1=0$
 $l=5$

$q_2=10$
 $c_2=1$
 $r=10$

$q_3=16$

2. Iteration

$q_0=5$ $q_1=6$
 $c_1=0$

$q_2=7$
 $c_2=0$
 $l=7$

$q_3=10$
 $r=10$

3. Iteration

$q_0=7$ $q_1=8$
 $c_1=0$

$q_2=9$
 $c_2=0$
 $l=9$

$q_3=10$
 $r=10$

Terminierung

$y < x_{10}$
 $c_1=1$

if $c_0 < c_1$ then return($l+j-1$)
= return($9+1-1$)

Satz 6.29 Zu der Folge $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$ und $y \in S$ berechnet der Algorithmus 7.2 einen Index i mit $x_i \leq y < x_{i+1}$ in der Zeitkomplexität $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$, wobei $p \leq n$ die Anzahl der Prozessoren ist.

Satz 6.29 Zu der Folge $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$ und $y \in S$ berechnet der Algorithmus 7.2 einen Index i mit $x_i \leq y < x_{i+1}$ in der Zeitkomplexität $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$, wobei $p \leq n$ die Anzahl der Prozessoren ist.

Beweis:

Zur Komplexität: In der $i+1$ -ten Iteration der while-Schleife wird die Länge der zu durchsuchenden Unterfolge von $s_i = r - l$ auf $s_{i+1} \leq \frac{r-l}{p+1} + p = \frac{s_i}{p+1} + p$ gesetzt, was die Länge des $(p+1)$ -ten Blockes beschränkt. Mit $s_0 = n+1$ löst man die rekurrente Ungleichung zu $s_i \leq \frac{n+1}{(p+1)^i} + p+1$.

Also werden $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$ Iterationen der Länge $O(1)$ benötigt. Schritt 3 benötigt $O(1)$ Zeit. $\square$

Satz 6.29 Zu der Folge $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$ und $y \in S$ berechnet der Algorithmus 7.2 einen Index i mit $x_i \leq y < x_{i+1}$ in $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$, wobei $p \leq n$ die Anzahl der Prozessoren ist.

$$s_i \leq \frac{n+1}{(p+1)^i} + p + 1 = p + 2$$

Beweis:

Zur Komplexitätsanalyse der while-Schleife wird die Länge der i -ten Iteration von $s_i = r - l$ auf $s_{i+1} \leq \frac{r-l}{p+1} + p = \frac{s_i}{p+1} + p$ gesetzt, was die Länge des $(p+1)$ -ten Blockes beschränkt. Mit $s_0 = n+1$ löst man die rekurrente Ungleichung zu $s_i \leq \frac{n+1}{(p+1)^i} + p + 1$. Also werden $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$ Iterationen der Länge $O(1)$ benötigt. Schritt 3 benötigt $O(1)$ Zeit. $\square$

Satz 6.29 Zu der Folge $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$ und $y \in S$ berechnet der Algorithmus 7.2 einen Index i mit $x_i \leq y < x_{i+1}$ in $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$, wobei $p \leq n$ die Anzahl der Prozessoren ist.

$$s_i \leq \frac{n+1}{(p+1)^i} + p + 1 = p + 2$$

für welches i ?

Beweis:

Zur Komplexitätsanalyse der while-Schleife wird die Länge der Blocke von $s_i = r - l$ auf $s_{i+1} \leq \frac{r-l}{p+1} + p = \frac{s_i}{p+1} + p$ gesetzt, was die Länge des $(p+1)$ -ten Blockes beschränkt. Mit $s_0 = n+1$ löst man die rekurrente Ungleichung zu $s_i \leq \frac{n+1}{(p+1)^i} + p + 1$. Also werden $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$ Iterationen der Länge $O(1)$ benötigt. Schritt 3 benötigt $O(1)$ Zeit. $\square$

Satz 6.29 Zu der Folge $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$ und $y \in S$ berechnet der Algorithmus 7.2 einen Index i mit $x_i \leq y < x_{i+1}$ in $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$, wobei $p \leq n$ die Anzahl der Prozessoren ist.

$$s_i \leq \frac{n+1}{(p+1)^i} + p + 1 = p + 2$$

für welches i ?

$$\frac{n+1}{(p+1)^i} = 1$$

Beweis:

Zur Komplexitätsanalyse der while-Schleife wird die Länge der while-Schleife von $s_i = r - l$ auf $s_{i+1} \leq \frac{r-l}{p+1} +$

$p = \frac{s_i}{p+1} + p$ gesetzt, was die Länge des $(p+1)$ -ten Blockes beschränkt.

Mit $s_0 = n + 1$ löst man die rekurrente Ungleichung zu $s_i \leq \frac{n+1}{(p+1)^i} + p + 1$.

Also werden $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$ Iterationen der Länge $O(1)$ benötigt. Schritt 3 benötigt $O(1)$ Zeit. $\square$

Satz 6.29 Zu der Folge $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$ und $y \in S$ berechnet der Algorithmus 7.2 einen Index i mit $x_i \leq y < x_{i+1}$ in $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$, wobei $p \leq n$ die Anzahl der Prozessoren ist.

$$s_i \leq \frac{n+1}{(p+1)^i} + p + 1 = p + 2$$

für welches i ?

$$\frac{n+1}{(p+1)^i} = 1$$

$$(p+1)^i = n+1$$

Beweis:

Zur Komplexitätsanalyse der while-Schleife wird die Länge der Schleife von $s_i = r - l$ auf $s_{i+1} \leq \frac{r-l}{p+1} +$

$p = \frac{s_i}{p+1} + p$ gesetzt, was die Länge des $(p+1)$ -ten Blockes beschränkt.

Mit $s_0 = n+1$ löst man die rekurrente Ungleichung zu $s_i \leq \frac{n+1}{(p+1)^i} + p + 1$.

Also werden $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$ Iterationen der Länge $O(1)$ benötigt. Schritt 3 benötigt $O(1)$ Zeit. $\square$

Satz 6.29 Zu der Folge $X = (x_1, x_2, \dots, x_n)$ mit $x_1 < x_2 < \dots < x_n$ und $y \in S$ berechnet der Algorithmus 7.2 einen Index i mit $x_i \leq y < x_{i+1}$ in $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$, wobei $p \leq n$ die Anzahl der Prozessoren ist.

$$s_i \leq \frac{n+1}{(p+1)^i} + p + 1 = p + 2$$

für welches i ?

$$\frac{n+1}{(p+1)^i} = 1$$

$$(p+1)^i = n+1$$

Beweis:

Zur Komplexitätsanalyse der while-Schleife wird die Länge der Schleife von $s_i = r - l$ auf $s_{i+1} \leq \frac{r-l}{p+1} + p = \frac{s_i}{p+1} + p$ gesetzt, was die Länge des $(p+1)$ -ten Blockes beschränkt. Mit $s_0 = n+1$ löst man die rekurrente Ungleichung zu $s_i \leq \frac{n+1}{(p+1)^i} + p + 1$. Also werden $O\left(\frac{\log(n+1)}{\log(p+1)}\right)$ Iterationen der Länge $O(1)$ benötigt. Schritt 3 benötigt $O(1)$ Zeit. $\square$

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

	$T(n)$	$W(n)$	

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

	$T(n)$	$W(n)$	
Satz 6.27	$O(\log n)$	$O(n)$ optimal	1. Ergebnis

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

	$T(n)$	$W(n)$	
Satz 6.27	$O(\log n)$	$O(n)$ optimal	1. Ergebnis
Korollar 6.31	$O(\log \log n)$	$O(n \log \log n)$	mit schnellem Suchen

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

	$T(n)$	$W(n)$	
Satz 6.27	$O(\log n)$	$O(n)$ optimal	1. Ergebnis
Korollar 6.31	$O(\log \log n)$	$O(n \log \log n)$	mit schnellem Suchen
Satz 6.32	$O(\log \log n)$	$O(n)$ optimal	Zitat

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

binäres Suchen

	$T(n)$	$W(n)$	
Satz 6.27	$O(\log n)$	$O(n)$ optimal	I. Ergebnis
Korollar 6.31	$O(\log \log n)$	$O(n \log \log n)$	mit schnellem Suchen
Satz 6.32	$O(\log \log n)$	$O(n)$ optimal	Zitat

Rückblick und Ausblick

Mischen von Folgen A und B der Länge n

	$T(n)$	$W(n)$	
Satz 6.27	$O(\log n)$	$O(n)$ optimal	binäres Suchen I. E.
Korollar 6.31	$O(\log \log n)$	$O(n \log \log n)$	paralleles Suchen schnellem Suchen
Satz 6.32	$O(\log \log n)$	$O(n)$ optimal	Zitat

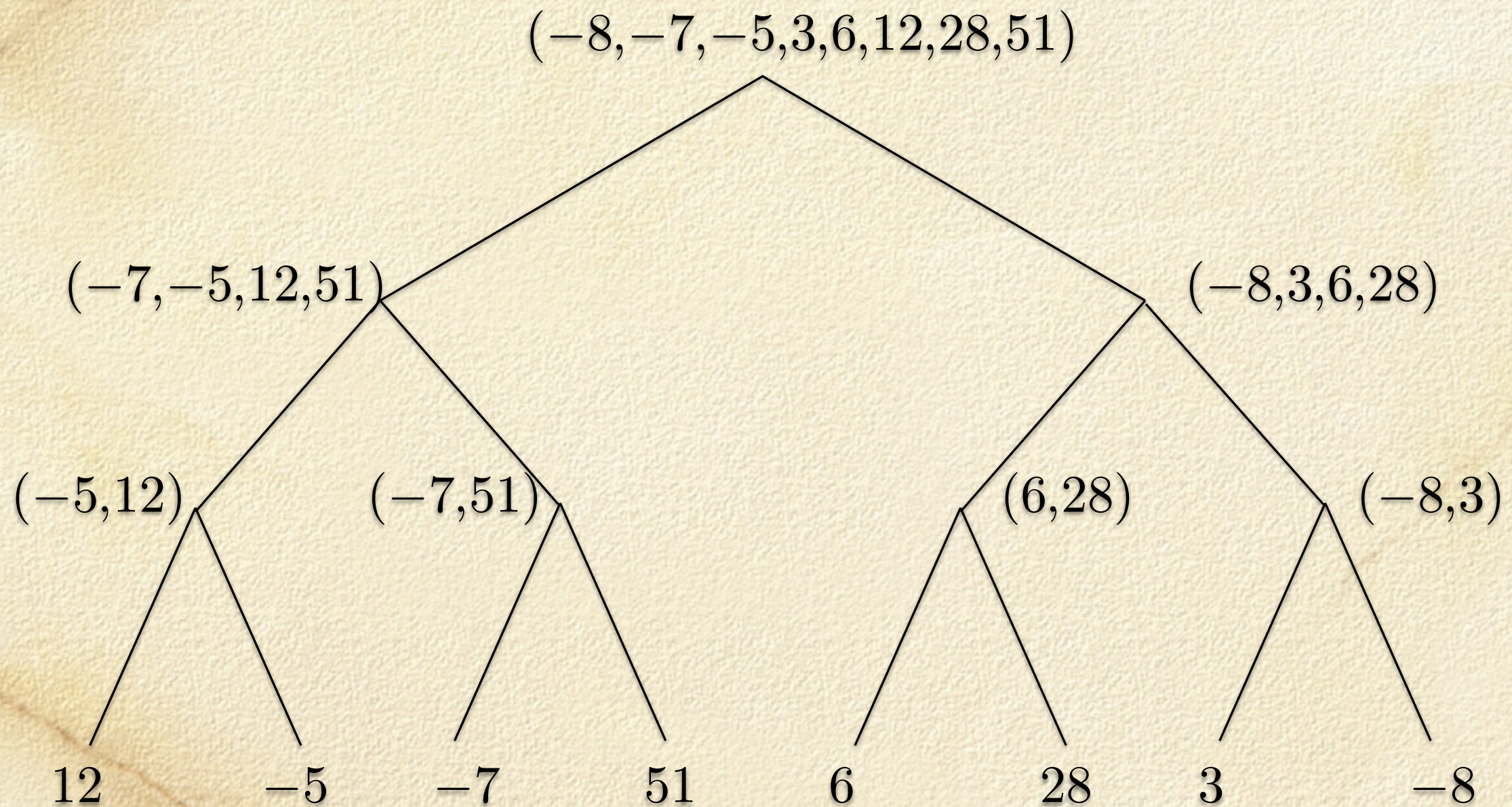
Paralleles Sortieren

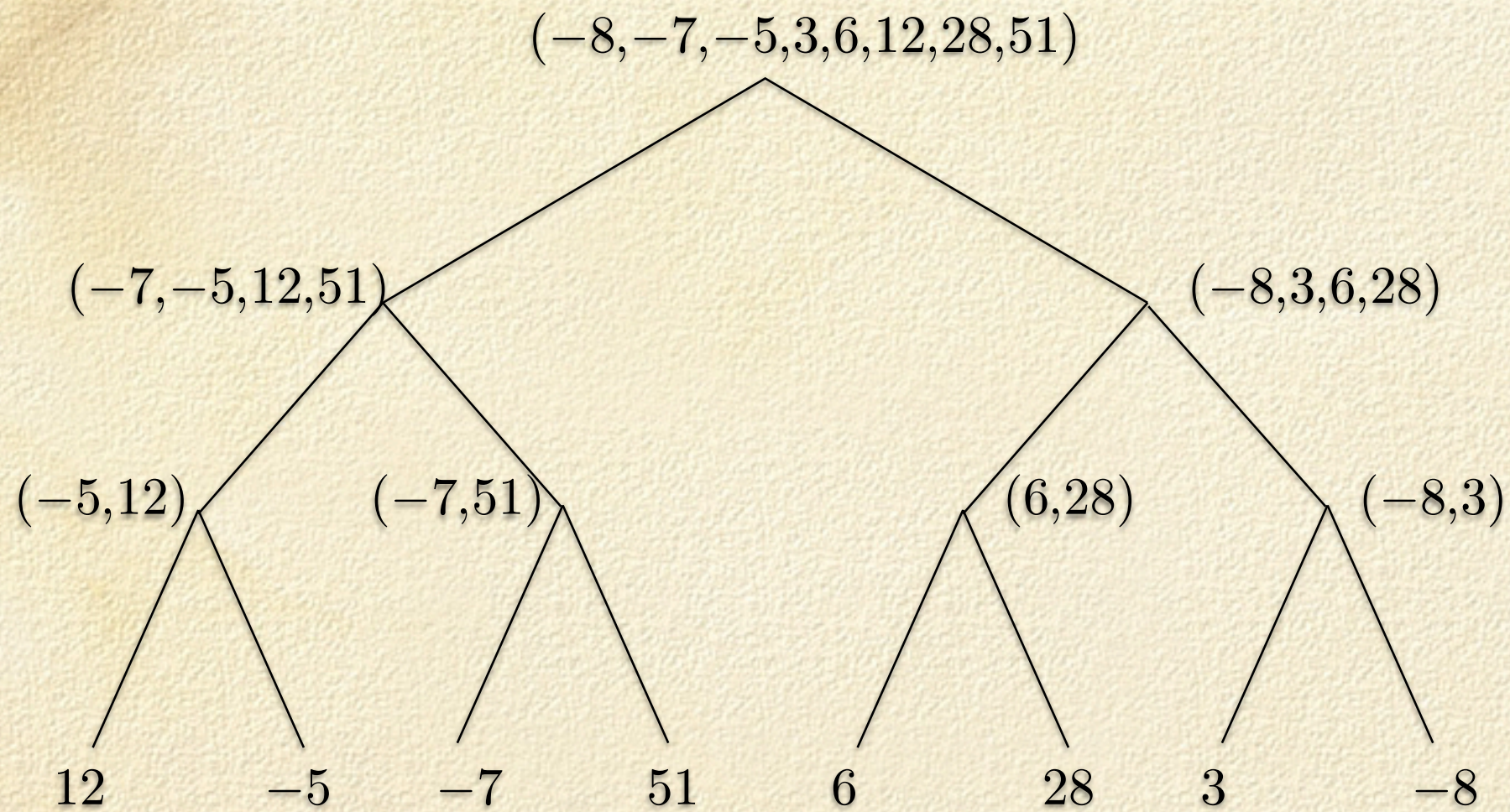
$$P(n) = n/2$$

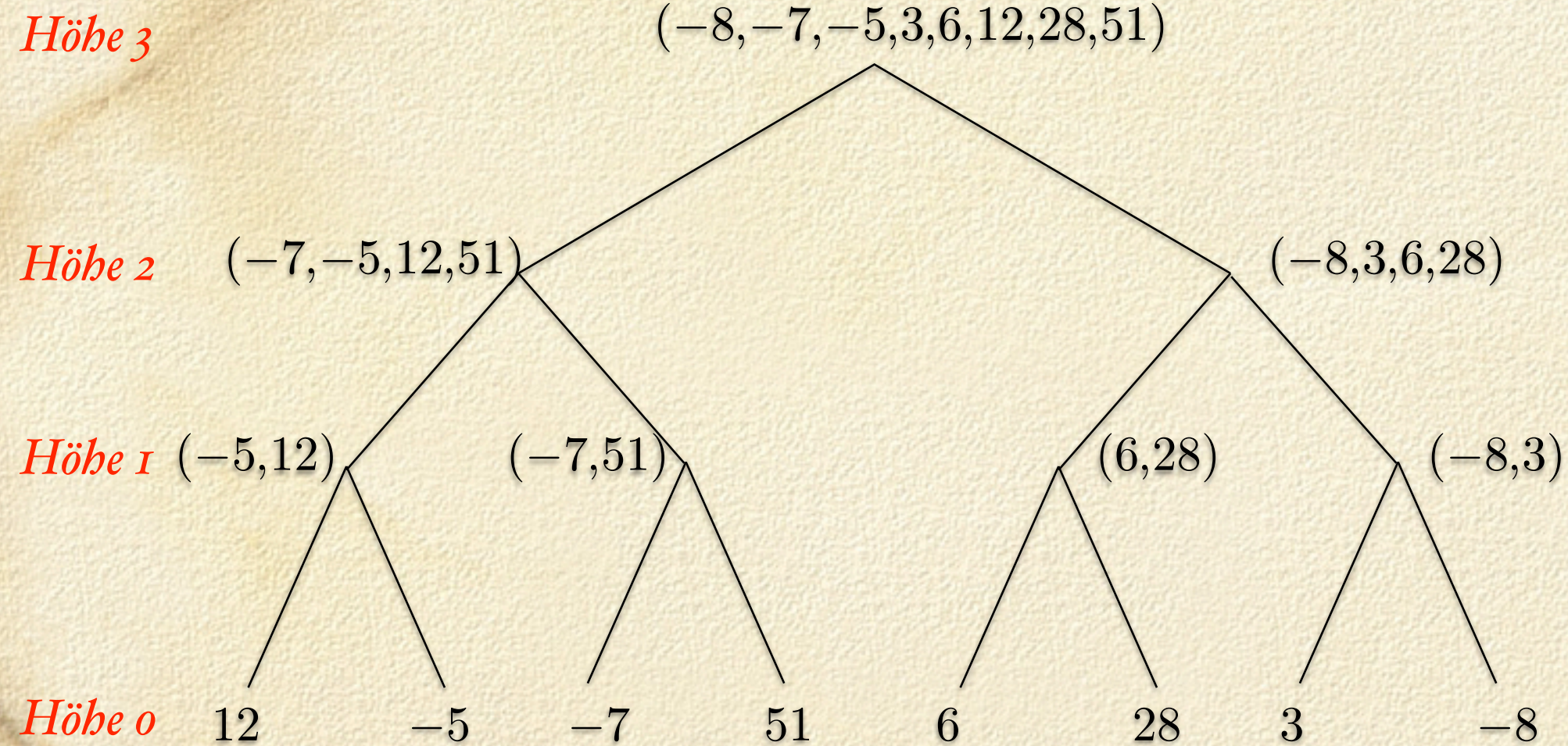
$$T(n) = \log n \log \log n$$

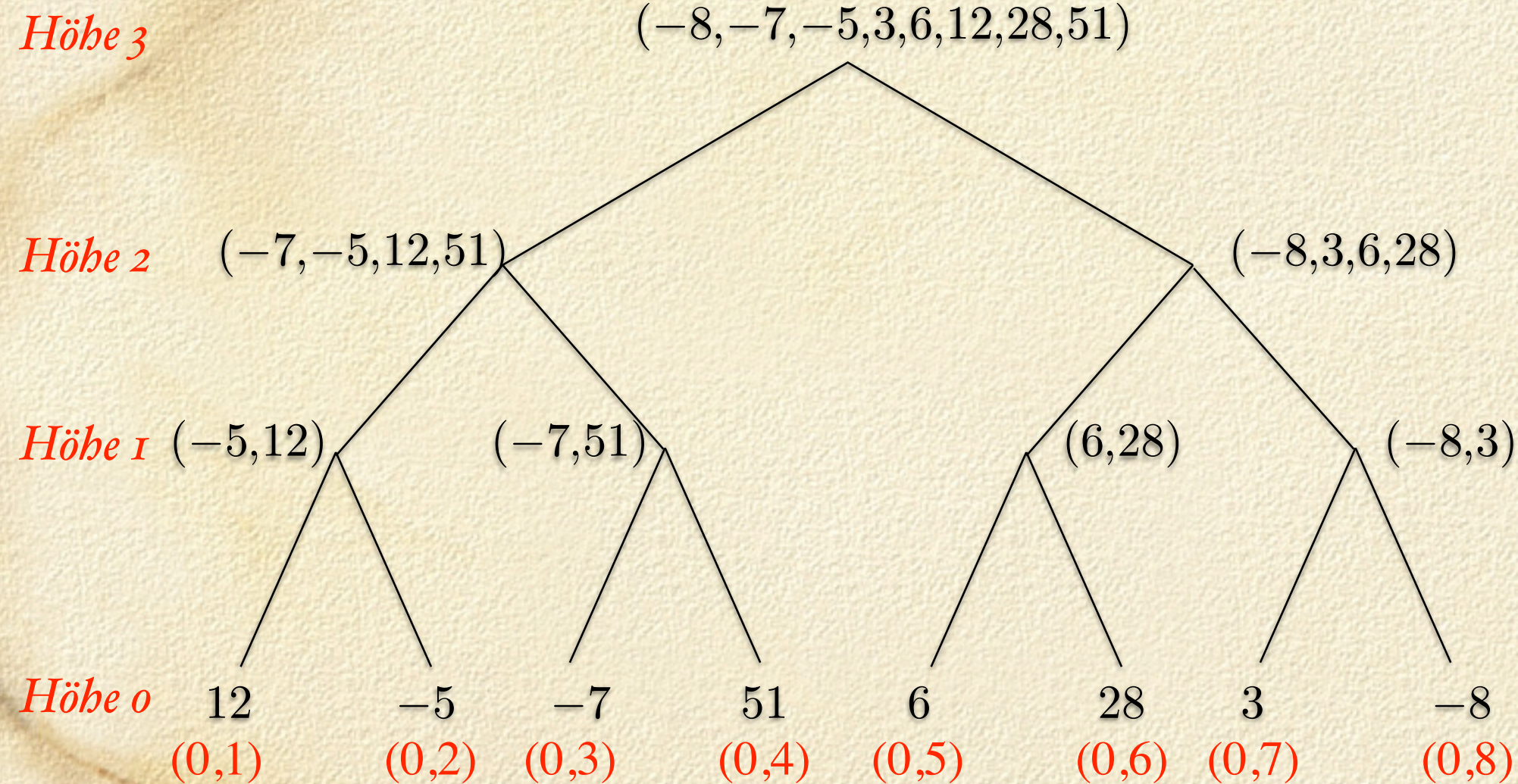
$$W(n) = n \log n$$

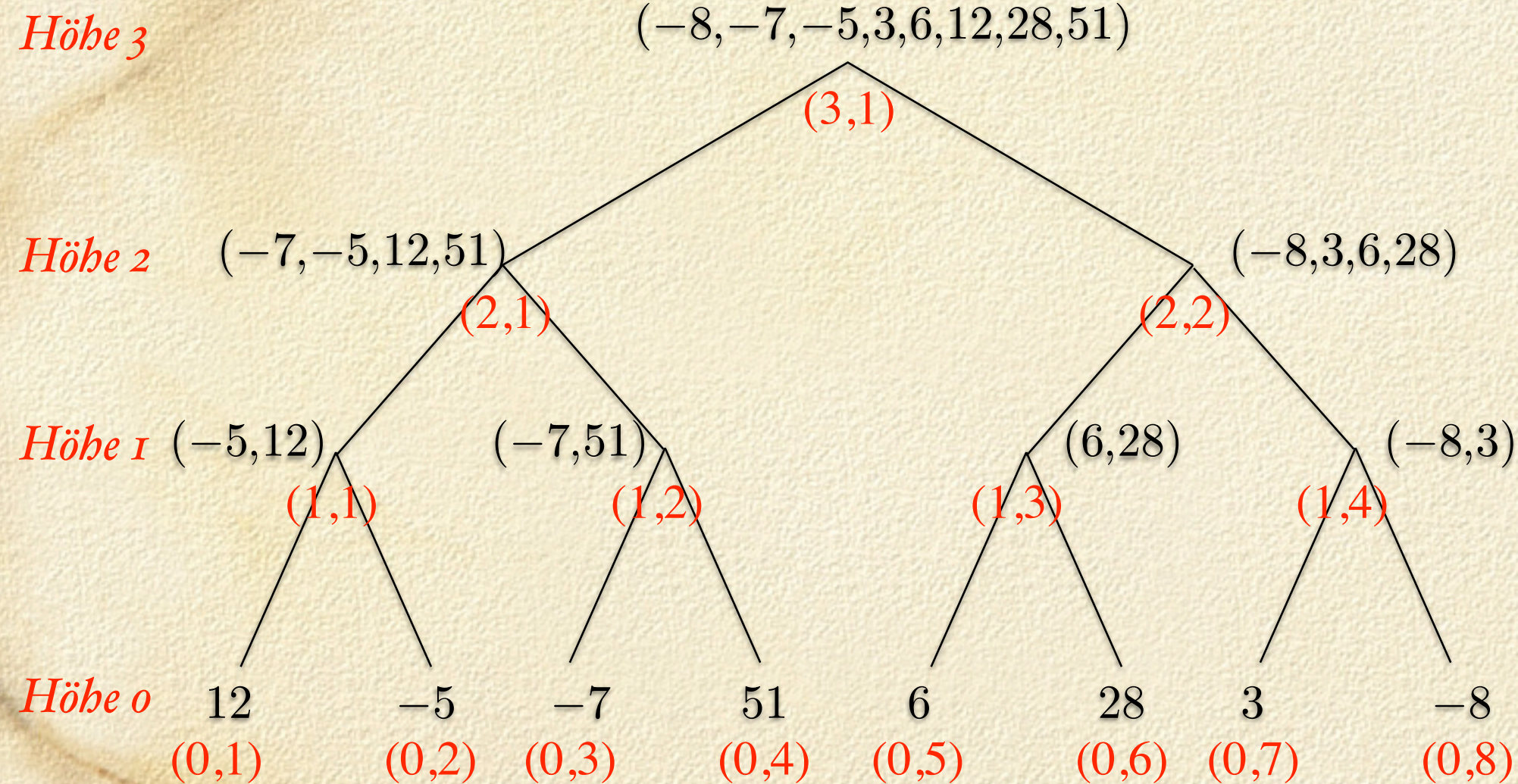
optimal

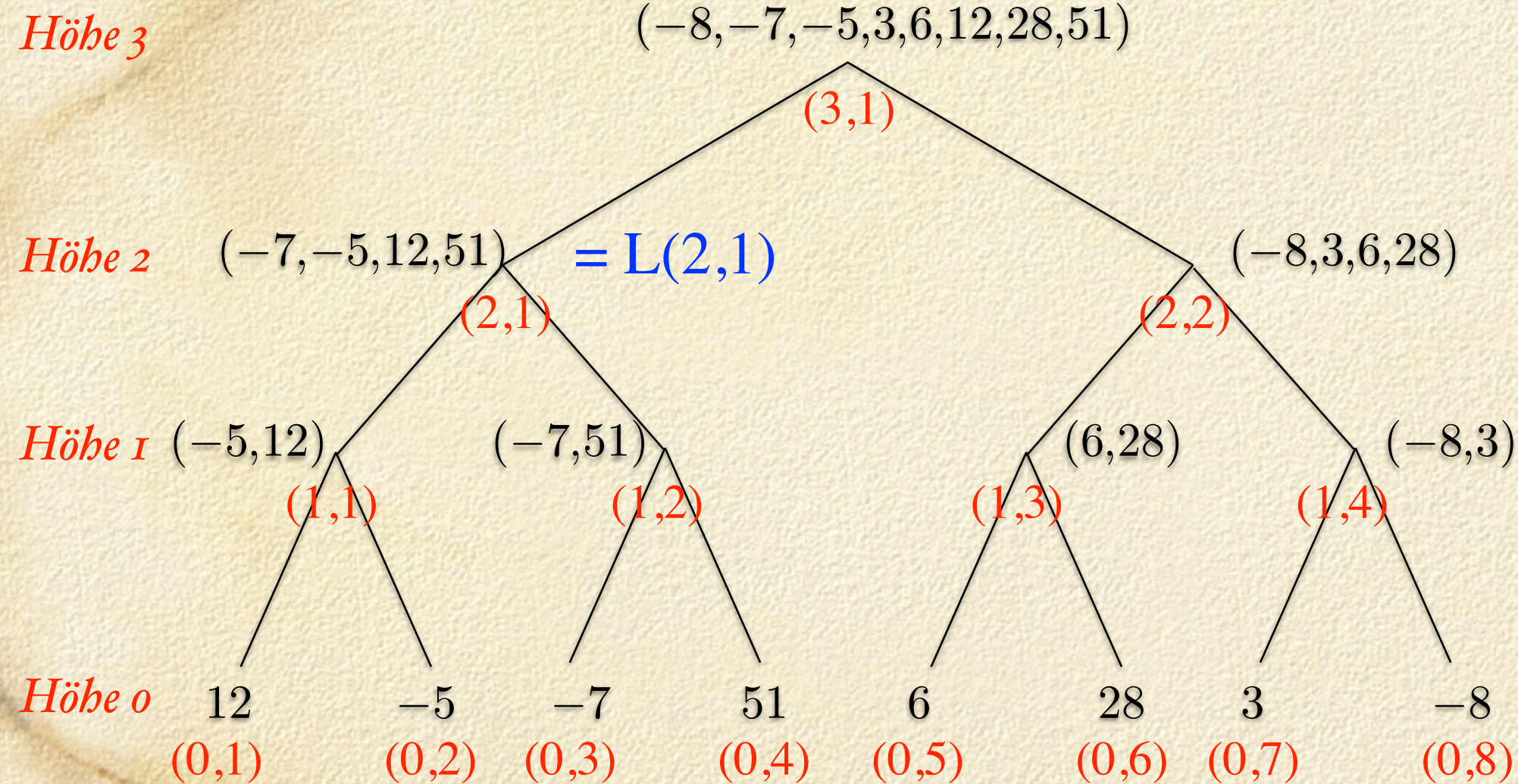












Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3, 1)$

Höhe 2

$(-7, -5, 12, 51)$

$= L(2, 1)$

$(-8, 3, 6, 28)$

$(2, 1)$

$(2, 2)$

Höhe 1

$(-5, 12)$

$(-7, 51)$

$(6, 28)$

$(-8, 3)$

$(1, 1)$

$(1, 2)$

$(1, 3)$

$(1, 4)$

Höhe 0

12

-5

-7

51

6

28

3

-8

$(0, 1)$

$(0, 2)$

$(0, 3)$

$(0, 4)$

$(0, 5)$

$(0, 6)$

$(0, 7)$

$(0, 8)$

Eingabe: Ein Feld X der Ordnung n , wobei $n = 2^l$ für eine ganze Zahl l ist.

Ausgabe: Ein balancierter binärer Baum mit n Blättern derart, dass $L(h, j)$ für jedes $0 \leq h \leq \log n$ die sortierte Teilfolge der Elemente im Teilbaum mit Wurzel (h, j) enthält (mit $1 \leq j \leq \frac{n}{2^h}$). Damit enthält der Knoten (h, j) die sortierte Liste der Elemente $X(2^h(j-1)+1), X(2^h(j-1)+2), \dots, X(2^h j)$.

begin

1. **for** $1 \leq j \leq n$ **pardo**

$L(0, j) := X(j)$

2. **for** $h = 1$ **to** $\log n$ **do**

for $1 \leq j \leq \frac{n}{2^h}$ **pardo**

Mische $L(h-1, 2j-1)$ und $L(h-1, 2j)$ zur sortierten Liste $L(h, j)$

end

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3, 1)$

Höhe 2

$(-7, -5, 12, 51)$

$= L(2, 1)$

$(-8, 3, 6, 28)$

$(2, 1)$

$(2, 2)$

Höhe 1

$(-5, 12)$

$(-7, 51)$

$(6, 28)$

$(-8, 3)$

$(1, 1)$

$(1, 2)$

$(1, 3)$

$(1, 4)$

Höhe 0

12

-5

-7

51

6

28

3

-8

$(0, 1)$

$(0, 2)$

$(0, 3)$

$(0, 4)$

$(0, 5)$

$(0, 6)$

$(0, 7)$

$(0, 8)$

Eingabe: Ein Feld X der Ordnung n , wobei $n = 2^l$ für eine ganze Zahl l ist.

Ausgabe: Ein balancierter binärer Baum mit n Blättern derart, dass $L(h, j)$

für jedes $0 \leq h \leq \log n$ die sortierte Teilfolge der Elemente im Teilbaum mit Wurzel (h, j) enthält (mit $1 \leq j \leq \frac{n}{2^h}$). Damit enthält der Knoten (h, j) die sortierte Liste der Elemente $X(2^h(j-1)+1), X(2^h(j-1)+2), \dots, X(2^h j)$.

begin *also für die Wurzel* $(h, j) = (\log n, 1): X(1, \dots, n)$

1. **for** $1 \leq j \leq n$ **pardo**

$L(0, j) := X(j)$

2. **for** $h = 1$ **to** $\log n$ **do**

for $1 \leq j \leq \frac{n}{2^h}$ **pardo**

Mische $L(h-1, 2j-1)$ und $L(h-1, 2j)$ zur sortierten Liste $L(h, j)$

end

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3, 1)$

Höhe 2

$(-7, -5, 12, 51)$

$= L(2, 1)$

$(2, 1)$

$(-8, 3, 6, 28)$

$(2, 2)$

Höhe 1

$(-5, 12)$

$(1, 1)$

$(-7, 51)$

$(1, 2)$

$(6, 28)$

$(1, 3)$

$(-8, 3)$

$(1, 4)$

Höhe 0

12

-5

-7

51

6

28

3

-8

$(0, 1)$

$(0, 2)$

$(0, 3)$

$(0, 4)$

$(0, 5)$

$(0, 6)$

$(0, 7)$

$(0, 8)$

Satz 6.34 Der Algorithmus 7.4 ist mit einer Zeitkomplexität von $O(\log n \cdot \log \log n)$ und $O(n \cdot \log n)$ Operationen optimal.

Algorithmus 6.4 (Parallel Merge Sort)

Eingabe: Ein Feld X der Ordnung n , wobei $n = 2^l$ für eine ganze Zahl l ist.

Ausgabe: Ein balancierter binärer Baum mit n Blättern derart, dass $L(h, j)$ für jedes $0 \leq h \leq \log n$ die sortierte Teilfolge der Elemente im Teilbaum mit Wurzel (h, j) enthält (mit $1 \leq j \leq \frac{n}{2^h}$). Damit enthält der Knoten (h, j) die sortierte Liste der Elemente $X(2^h(j-1) + 1), X(2^h(j-1) + 2), \dots, X(2^h j)$.

begin

1. **for** $1 \leq j \leq n$ **pardo**

$L(0, j) := X(j)$

2. **for** $h = 1$ **to** $\log n$ **do**

for $1 \leq j \leq \frac{n}{2^h}$ **pardo**

Mische $L(h-1, 2j-1)$ und $L(h-1, 2j)$ zur sortierten Liste $L(h, j)$

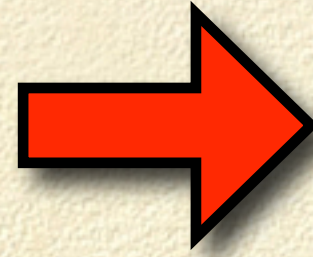
end

schnelles Mischen von Folgen

der Länge m

$T(m) = O(\log \log m)$

$W(m) = O(m)$



Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3,1)$

Höhe 2

$(-7, -5, 12, 51)$

$(2,1)$

$(-8, 3, 6, 28)$

$(2,2)$

Höhe 1

$(-5, 12)$

$(1,1)$

$(-7, 51)$

$(1,2)$

$(6, 28)$

$(1,3)$

$(-8, 3)$

$(1,4)$

Höhe 0

12

$(0,1)$

-5

$(0,2)$

-7

$(0,3)$

51

$(0,4)$

6

$(0,5)$

28

$(0,6)$

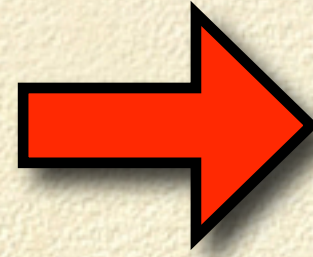
3

$(0,7)$

-8

$(0,8)$

schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3,1)$

Höhe 2

$(-7, -5, 12, 51)$

$(2,1)$

$(-8, 3, 6, 28)$

$(2,2)$

Höhe 1

$(-5, 12)$

$(1,1)$

$(-7, 51)$

$(1,2)$

$(6, 28)$

$(1,3)$

$(-8, 3)$

$(1,4)$

Höhe 0

12

$(0,1)$

-5

$(0,2)$

-7

$(0,3)$

51

$(0,4)$

6

$(0,5)$

28

$(0,6)$

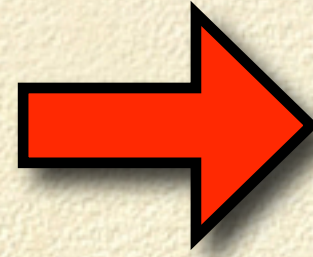
3

$(0,7)$

-8

$(0,8)$

schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3,1)$

Höhe 2

$(-7, -5, 12, 51)$

$(2,1)$

$(-8, 3, 6, 28)$

$(2,2)$

Höhe 1

$(-5, 12)$

$(1,1)$

$(-7, 51)$

$(1,2)$

$(6, 28)$

$(1,3)$

$(-8, 3)$

$(1,4)$

Höhe 0

12

$(0,1)$

-5

$(0,2)$

-7

$(0,3)$

51

$(0,4)$

6

$(0,5)$

28

$(0,6)$

3

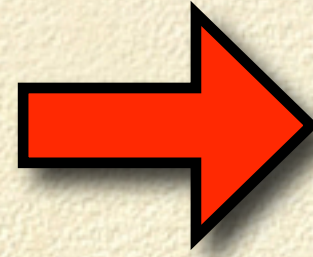
$(0,7)$

-8

$(0,8)$

in Höhe h: Folgen haben Länge 2^h , *d.h.* $T(n) =$

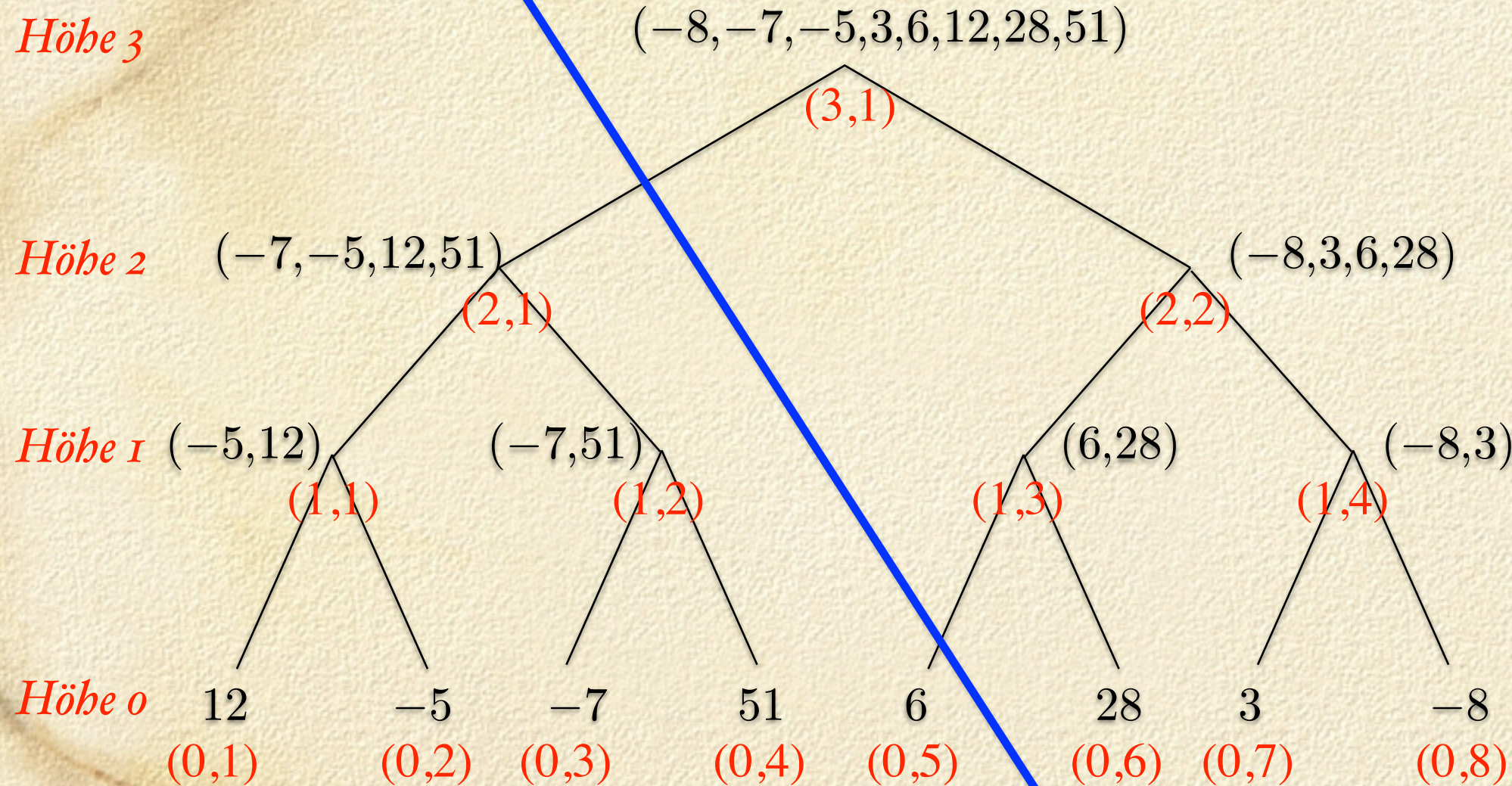
schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

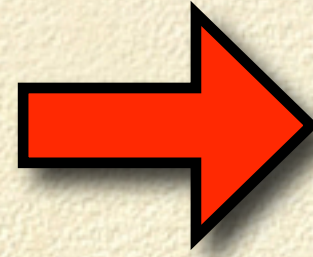
$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$



in Höhe h: Folgen haben Länge 2^h , *d.h.* $T(n) =$

schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3,1)$

Höhe 2

$(-7, -5, 12, 51)$

$(2,1)$

$(-8, 3, 6, 28)$

$(2,2)$

Höhe 1

$(-5, 12)$

$(1,1)$

$(-7, 51)$

$(1,2)$

$(6, 28)$

$(1,3)$

$(-8, 3)$

$(1,4)$

Höhe 0

12

$(0,1)$

-5

$(0,2)$

-7

$(0,3)$

51

$(0,4)$

6

$(0,5)$

28

$(0,6)$

3

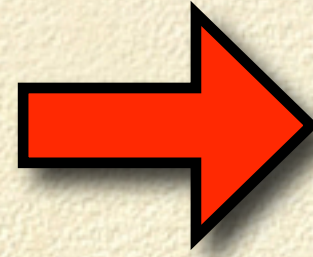
$(0,7)$

-8

$(0,8)$

in Höhe h: Folgen haben Länge 2^h , *d.h.* $T(n) = O(\log \log 2^h) = O(\log h) \leq O(\log \log n)$

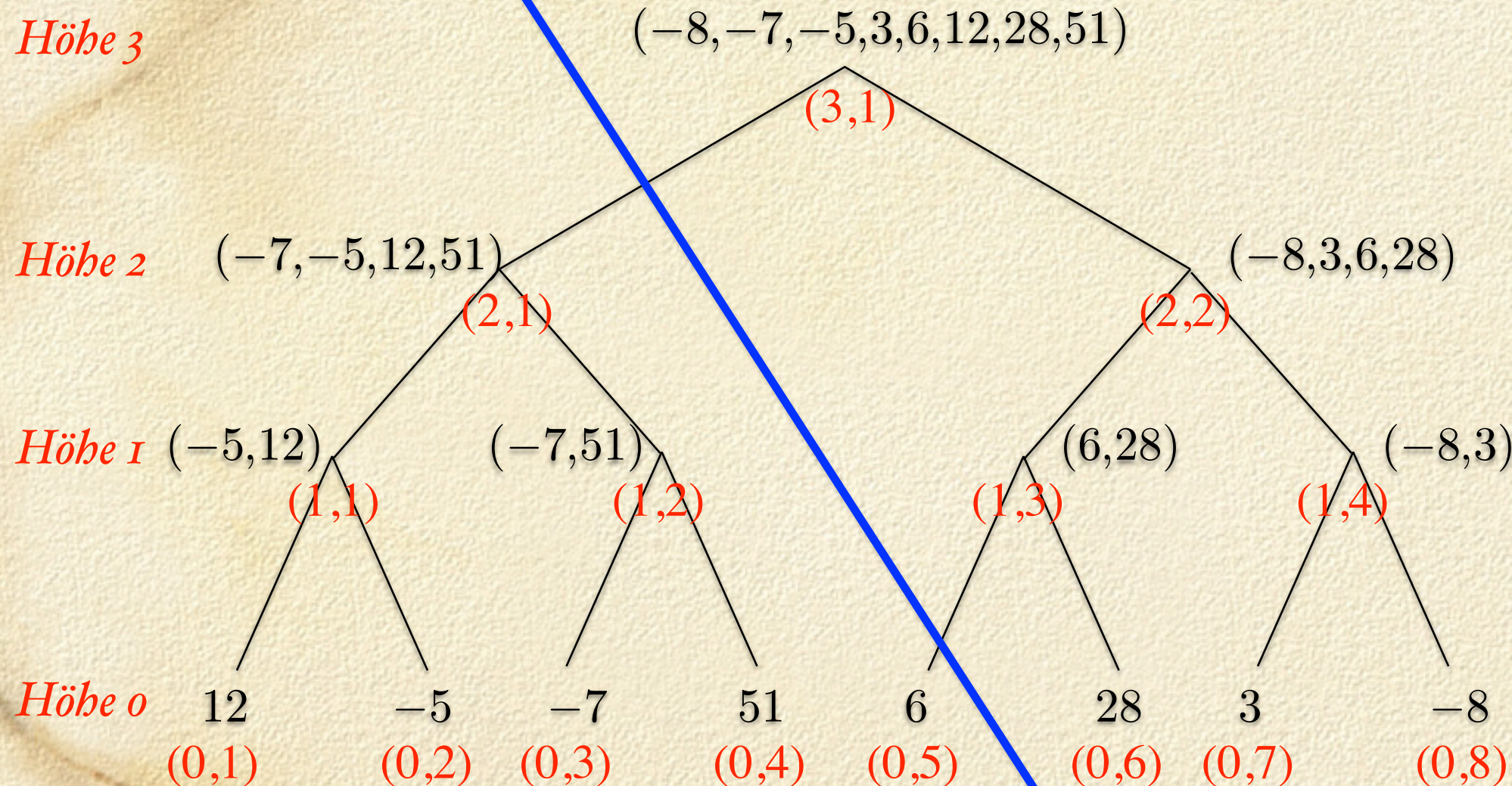
schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

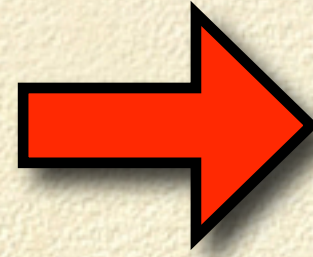
$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$



in Höhe h: Folgen haben Länge 2^h , d.h. $T(n) = O(\log \log 2^h) = O(\log h) \leq O(\log \log n)$ *insgesamt* $O(\log n \log \log n)$

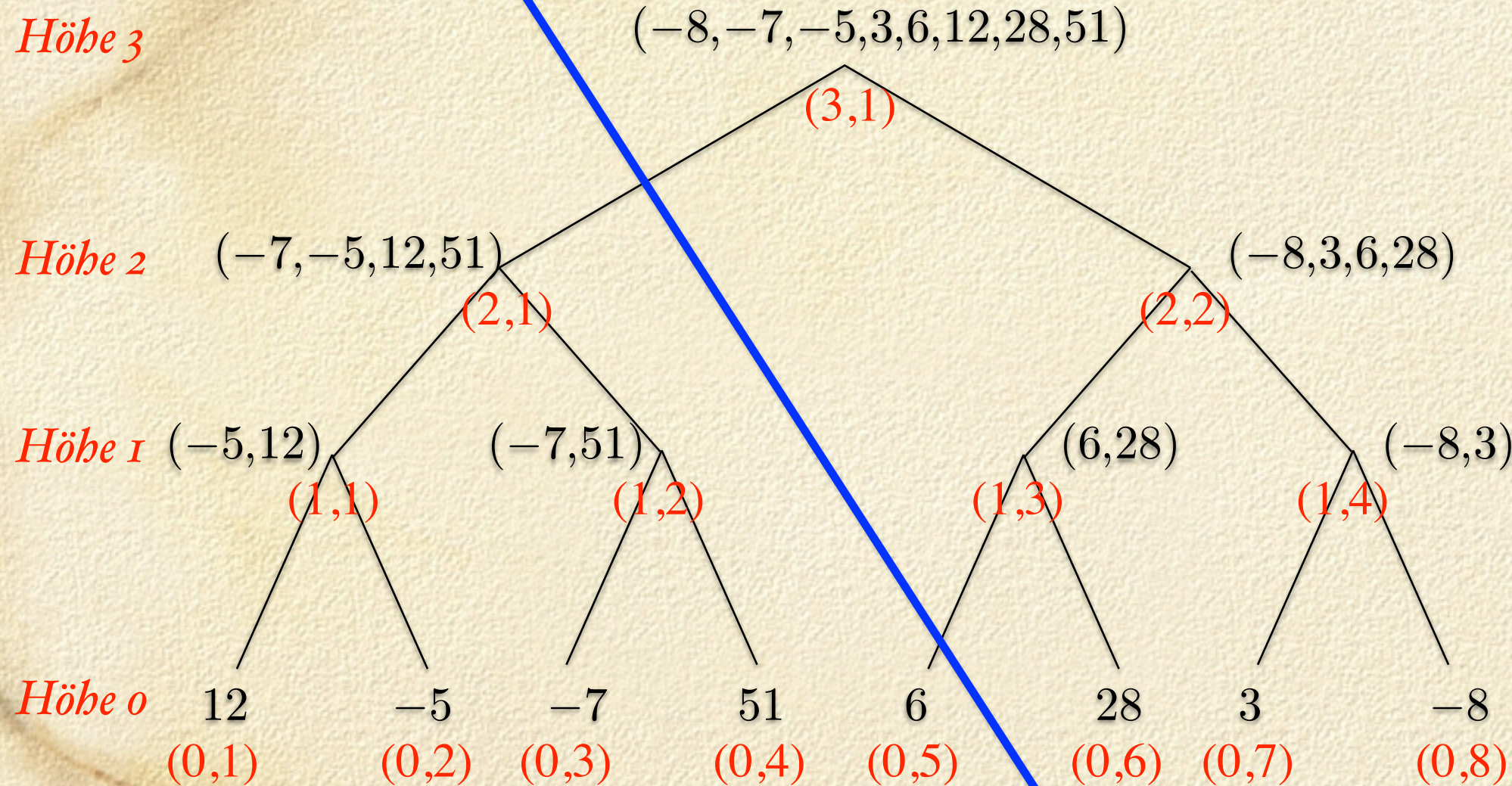
schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

$$T(n) = O(\log n \log \log n)$$

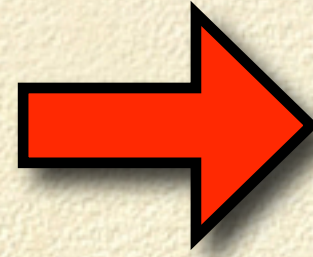
$$W(n) = O(n \log n)$$



in Höhe h: Folgen haben Länge 2^h , d.h. $T(n) = O(\log \log 2^h) = O(\log h) \leq O(\log \log n)$ *insgesamt* $O(\log n \log \log n)$

in Höhe h: Anzahl der Folgen: $n/2^h$, d.h. $W(n) =$

schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3,1)$

Höhe 2

$(-7, -5, 12, 51)$

$(2,1)$

$(-8, 3, 6, 28)$

$(2,2)$

Höhe 1

$(-5, 12)$

$(1,1)$

$(-7, 51)$

$(1,2)$

$(6, 28)$

$(1,3)$

$(-8, 3)$

$(1,4)$

Höhe 0

12

$(0,1)$

-5

$(0,2)$

-7

$(0,3)$

51

$(0,4)$

6

$(0,5)$

28

$(0,6)$

3

$(0,7)$

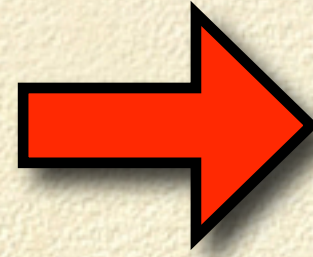
-8

$(0,8)$

in Höhe h : Folgen haben Länge 2^h , d.h. $T(n) = O(\log \log 2^h) = O(\log h) \leq O(\log \log n)$ insgesamt $O(\log n \log \log n)$

in Höhe h : Anzahl der Folgen: $n/2^h$, d.h. $W(n) =$

schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3,1)$

Höhe 2

$(-7, -5, 12, 51)$

$(2,1)$

$(-8, 3, 6, 28)$

$(2,2)$

Höhe 1

$(-5, 12)$

$(1,1)$

$(-7, 51)$

$(1,2)$

$(6, 28)$

$(1,3)$

$(-8, 3)$

$(1,4)$

Höhe 0

12

$(0,1)$

-5

$(0,2)$

-7

$(0,3)$

51

$(0,4)$

6

$(0,5)$

28

$(0,6)$

3

$(0,7)$

-8

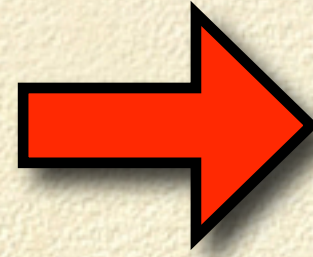
$(0,8)$

in Höhe h : Folgen haben Länge 2^h , d.h. $T(n) = O(\log \log 2^h) = O(\log h) \leq O(\log \log n)$ insgesamt $O(\log n \log \log n)$

in Höhe h : Anzahl der Folgen: $n/2^h$, d.h. $W(n) = O(2^h \cdot \frac{n}{2^h}) = O(n)$

Länge parallele Algorithmen (Teil 2)

schnelles Mischen von Folgen
der Länge m
 $T(m) = O(\log \log m)$
 $W(m) = O(m)$



$$P(n) = n/2$$

$$T(n) = O(\log n \log \log n)$$

$$W(n) = O(n \log n)$$

Höhe 3

$(-8, -7, -5, 3, 6, 12, 28, 51)$

$(3,1)$

Höhe 2

$(-7, -5, 12, 51)$

$(2,1)$

$(-8, 3, 6, 28)$

$(2,2)$

Höhe 1

$(-5, 12)$

$(1,1)$

$(-7, 51)$

$(1,2)$

$(6, 28)$

$(1,3)$

$(-8, 3)$

$(1,4)$

Höhe 0

12

$(0,1)$

-5

$(0,2)$

-7

$(0,3)$

51

$(0,4)$

6

$(0,5)$

28

$(0,6)$

3

$(0,7)$

-8

$(0,8)$

in Höhe h : Folgen haben Länge 2^h , d.h. $T(n) = O(\log \log 2^h) = O(\log h) \leq O(\log \log n)$ insgesamt $O(\log n \log \log n)$

in Höhe h : Anzahl der Folgen: $n/2^h$, d.h. $W(n) = O(2^h \cdot \frac{n}{2^h}) = O(n)$ insgesamt $O(n \log n)$

Länge parallele Algorithmen (Teil 2)

