Algoritmer og Datastrukturer 1

Quicksort [CLRS, kapitel 7]

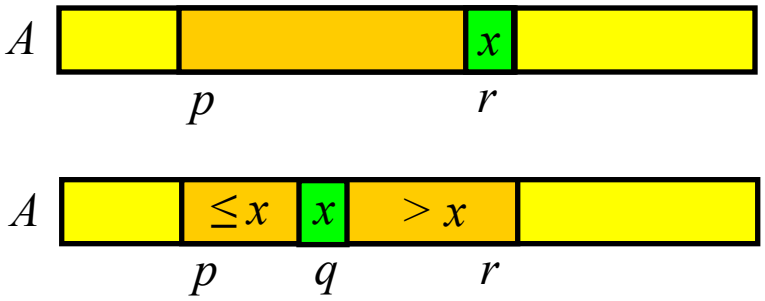
Gerth Stølting Brodal

Aarhus Universitet

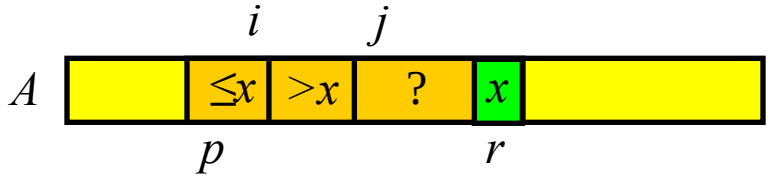
Quicksort:

Sorter *A*[*p*..*r*]

```
QUICKSORT(A, p, r)
1  if p < r
2     then q ← PARTITION(A, p, r)
3          QUICKSORT(A, p, q − 1)
4          QUICKSORT(A, q + 1, r)
```



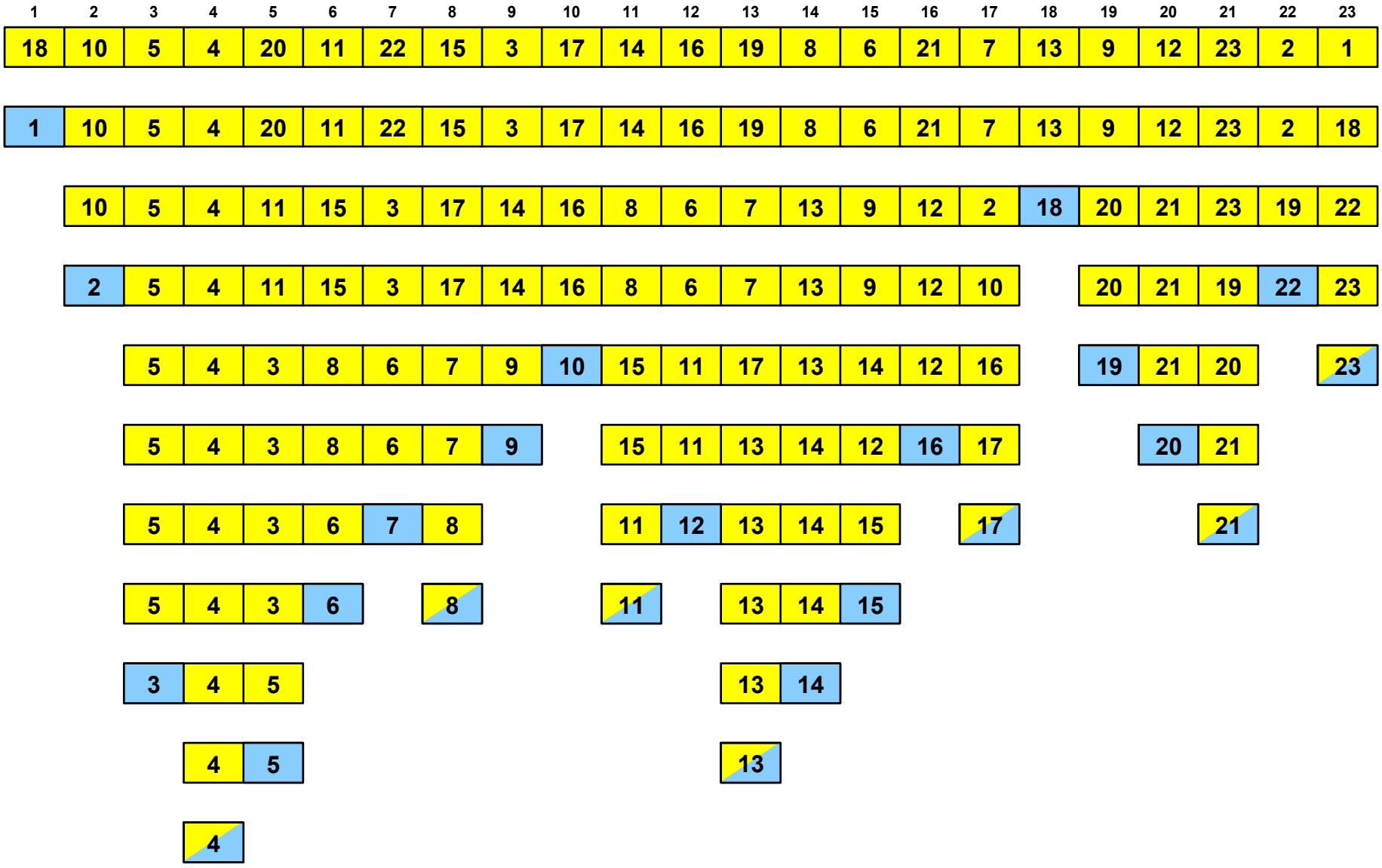

```
PARTITION(A, p, r)
1  x ← A[r]
2  i ← p − 1
3  for j ← p to r − 1
4       do if A[j] ≤ x
5             then i ← i + 1
6                  exchange A[i] ↔ A[j]
7  exchange A[i + 1] ↔ A[r]
8  return i + 1
```



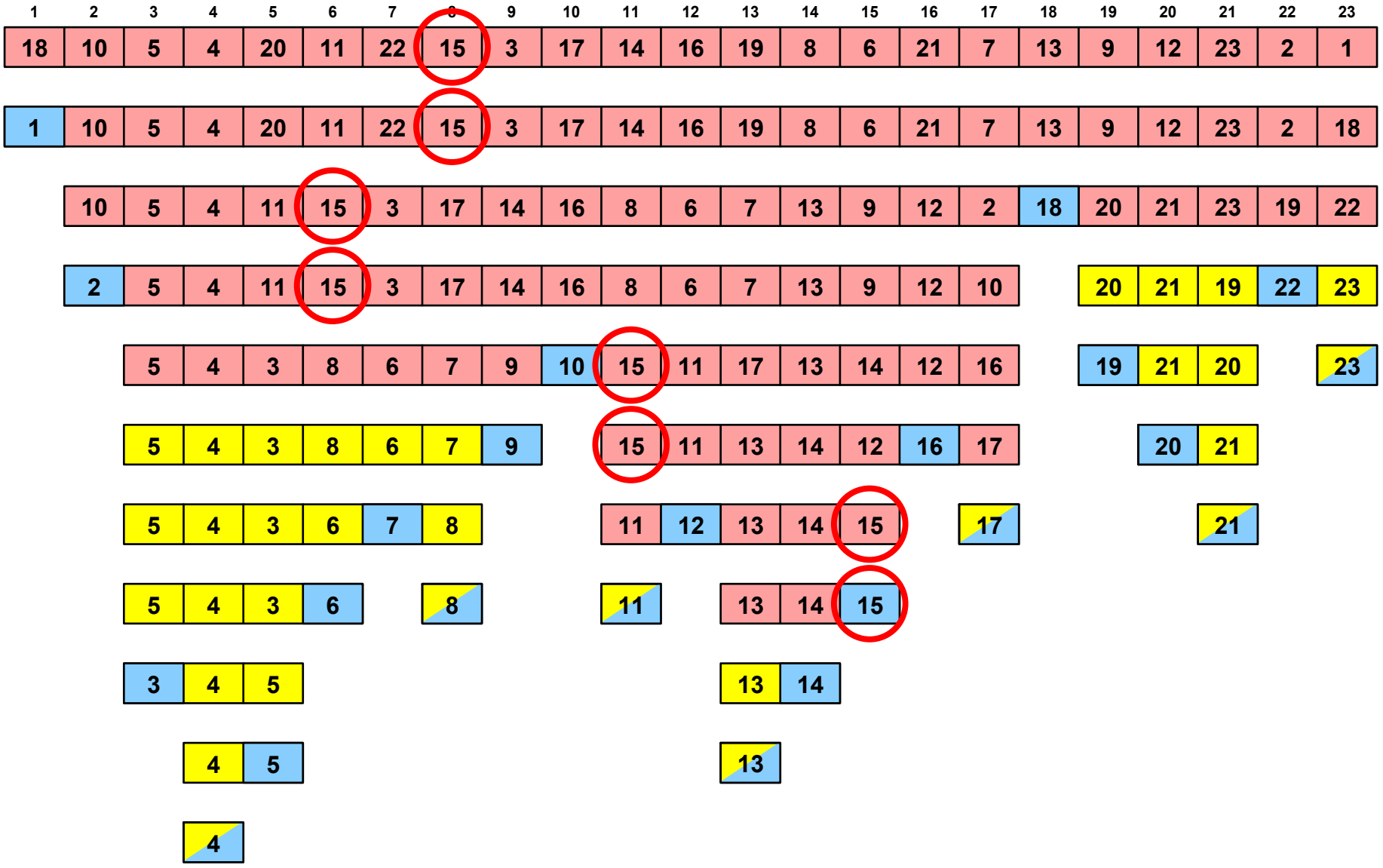
Worst-case tid O(n^2)

Hoare, 1961

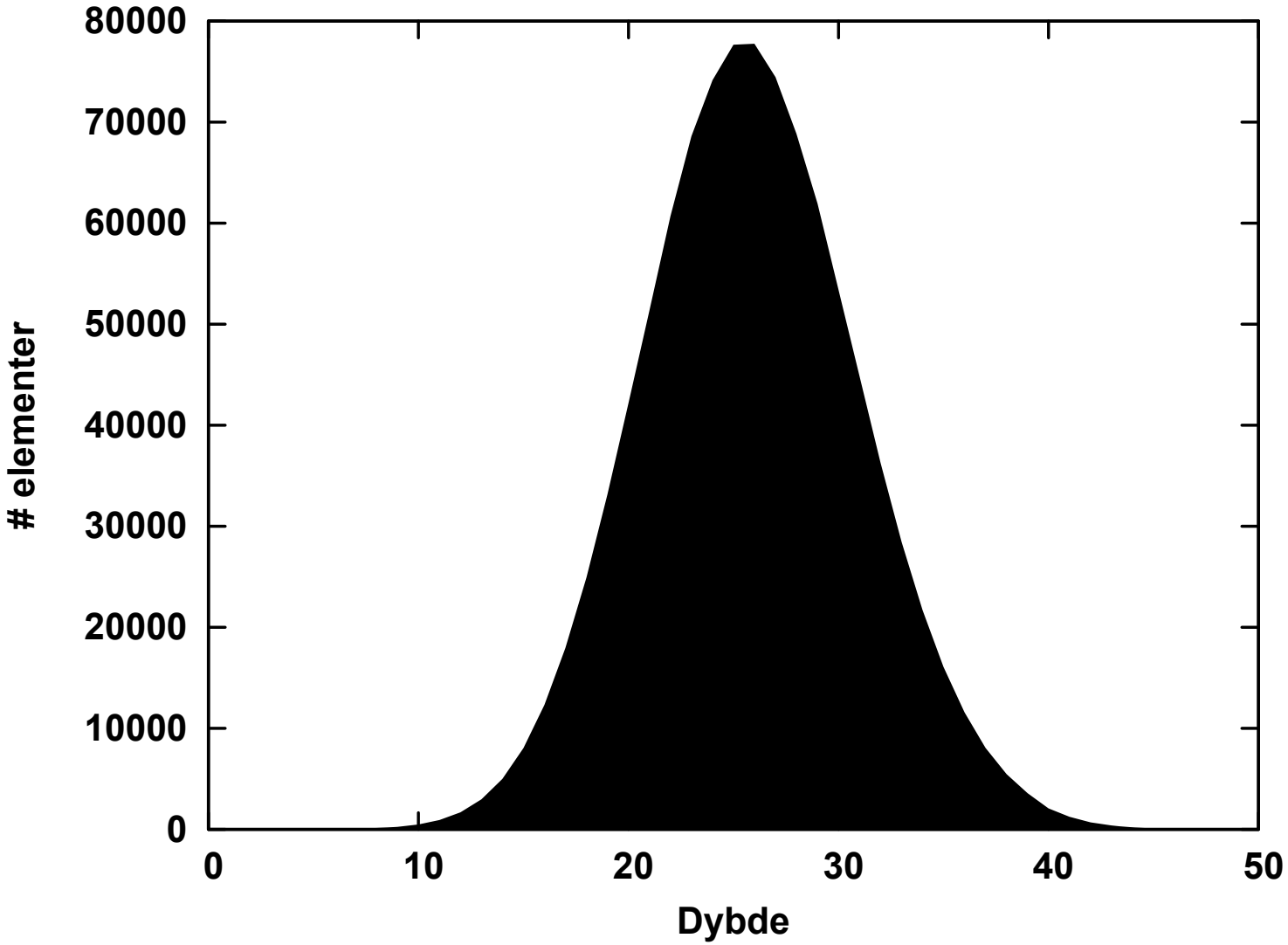
Quicksort på 23 elementer



Quicksort : Rekursionen for 15



Quicksort : Dybde ved $n \approx 2^{20}$



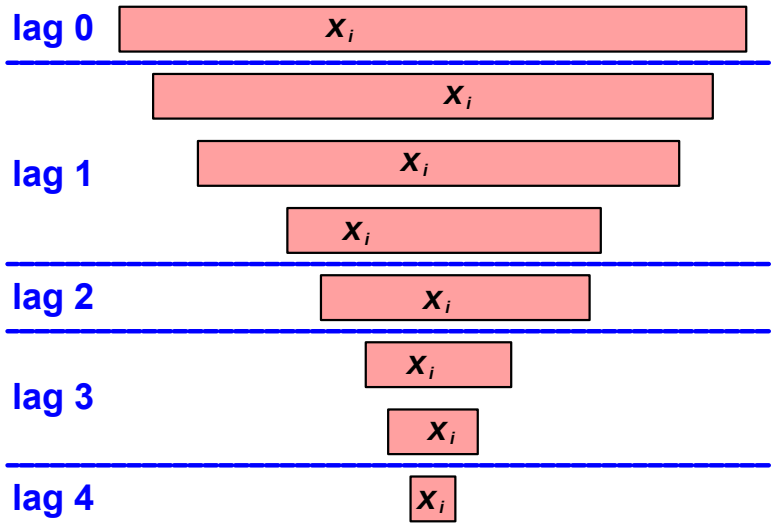
Randomized Quicksort

```
RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2     then q ← RANDOMIZED-PARTITION(A, p, r)
3          RANDOMIZED-QUICKSORT(A, p, q − 1)
4          RANDOMIZED-QUICKSORT(A, q + 1, r)
```

```
RANDOMIZED-PARTITION(A, p, r)
1  i ← RANDOM(p, r)
2  exchange A[r] ↔ A[i]
3  return PARTITION(A, p, r)
```

Forventet tid O(n·log n)

Randomized Quicksort : Analyse



- Et array er i lag j hvis længde $n(3/4)^{j+1}$.. $n(3/4)^j$
- En opdeling er **god** hvis hver del ≤ ¾ elementer (mindst +1 lag) – sker med sandsynlighed ≥ 0.5
- x_i forventes ≤ 2 gange i hvert lag
- **Forventede dybde af x_i ≤ $2 \cdot \log_{4/3} n$**

Randomized Quicksort : Analyse

Forventede tid for randomized quicksort

= O(**Σ**$_i$ **forventede dybde** af input x_i)

= O(**Σ**$_i$ log n)

= **O(n·log n)**

Sorterings-algoritmer

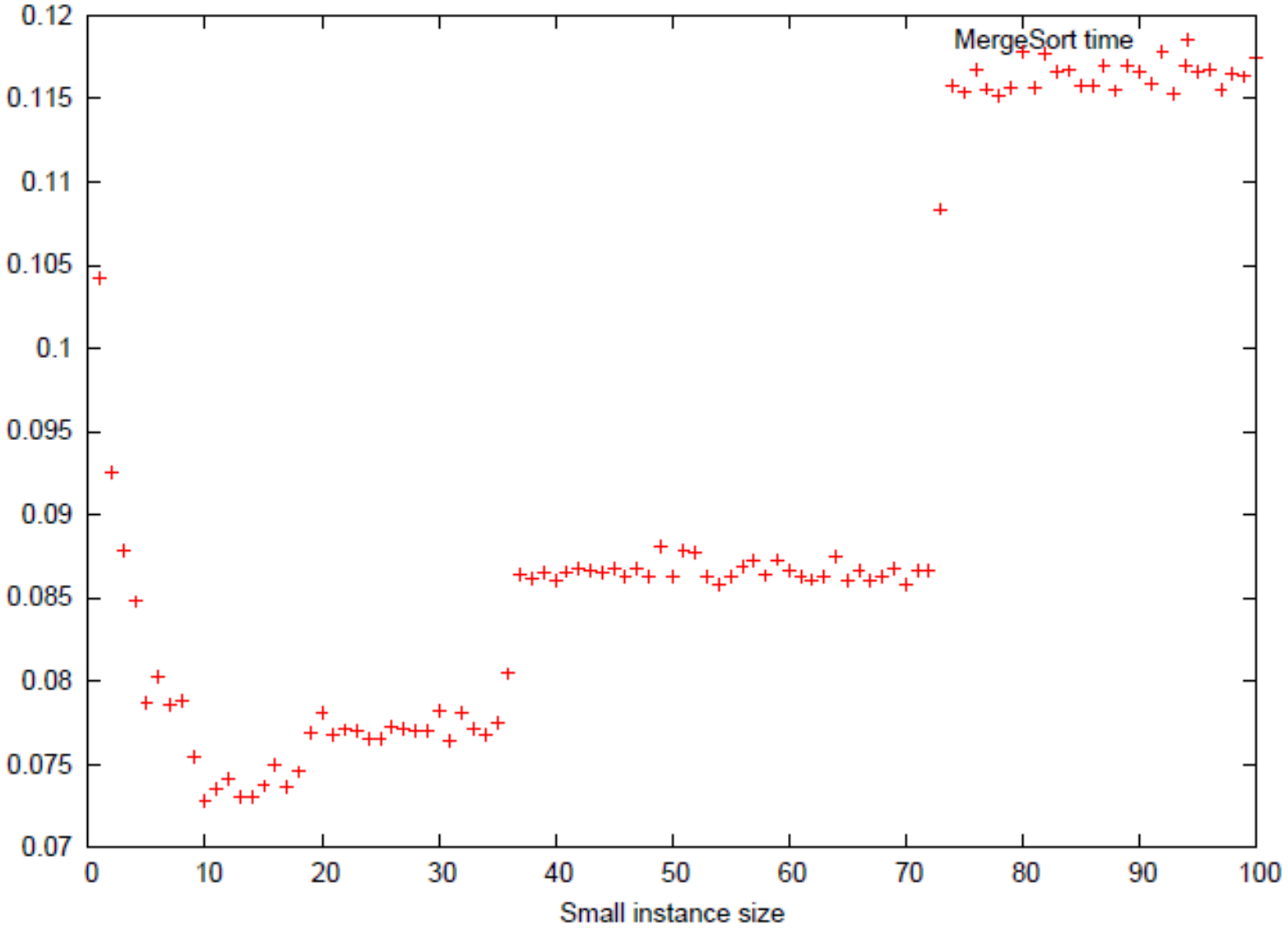
Algoritme	Worst-Case Tid
Heap-Sort	$O(n \cdot \log n)$
Merge-Sort	
Insertion-Sort	$O(n^2)$
QuickSort (Deterministic og randomiseret)	$O(n^2)$

Algoritme	Forventet tid
Randomiseret QuickSort	$O(n \cdot \log n)$

Sortering:

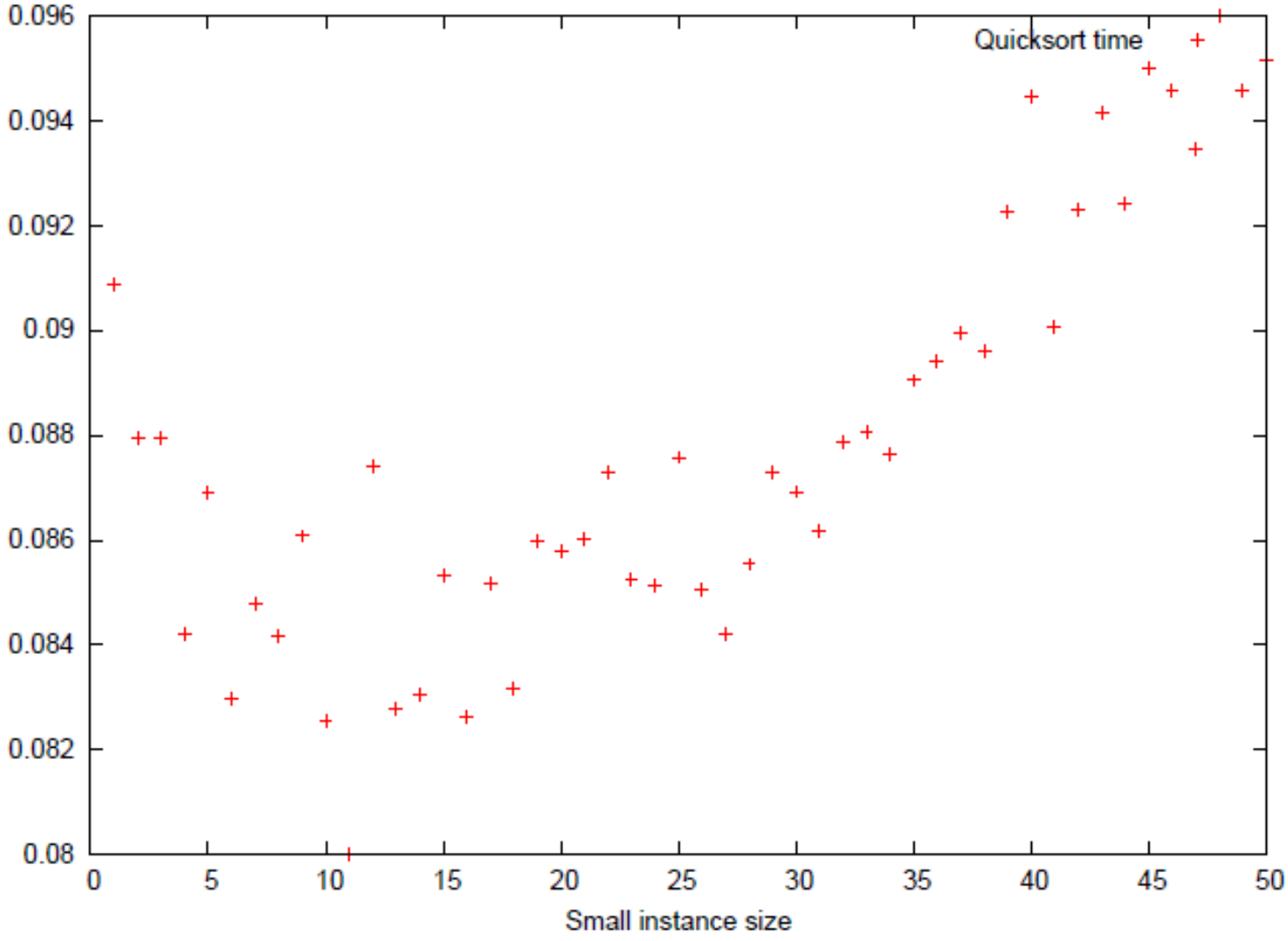
Eksperimentelle resultater

Mergesort med skift til Insertion-sort



Skift til insertion-sort ved små problemstørrelser n = 300.000 elementer

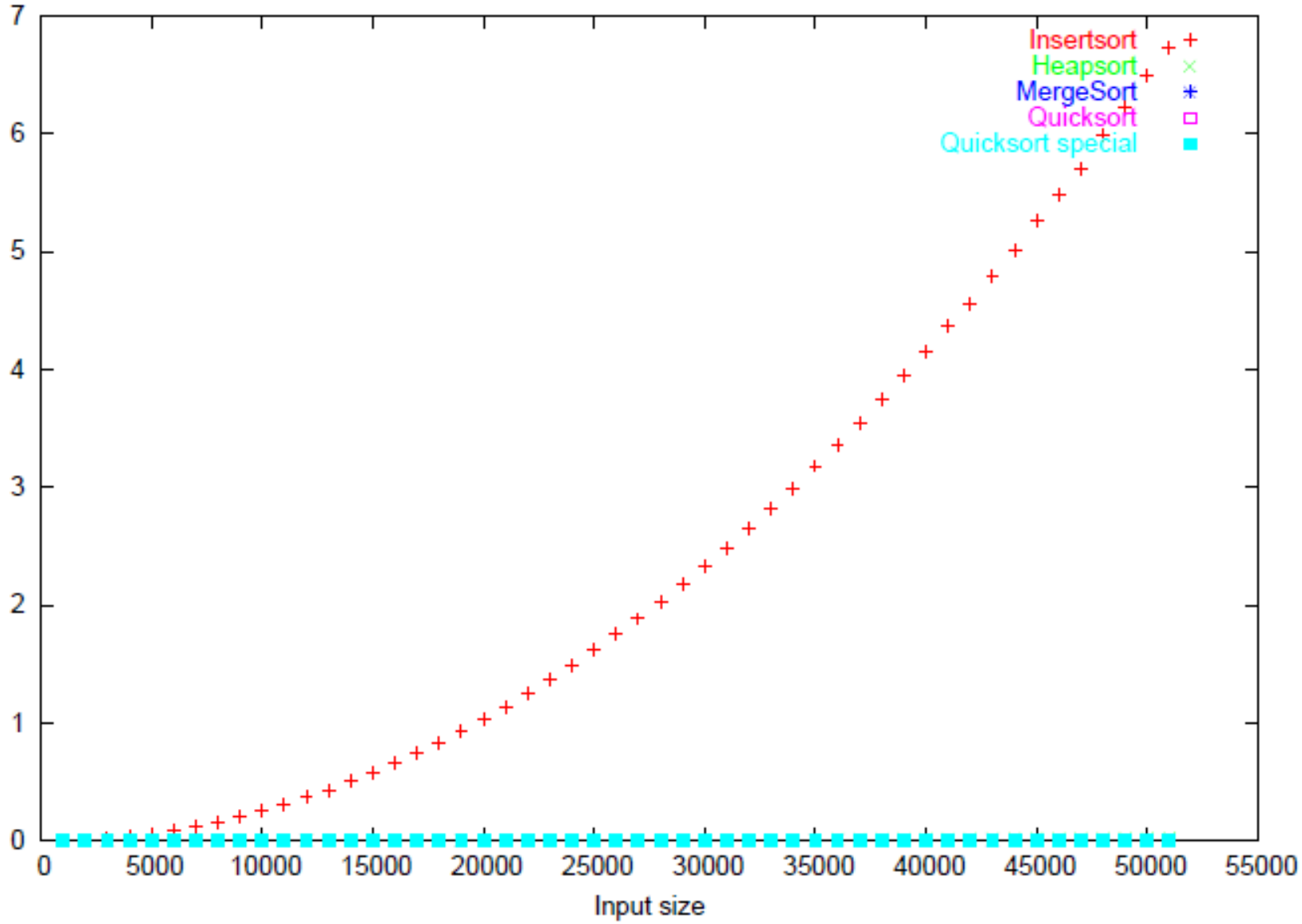
Quicksort med skift til Insertion-sort



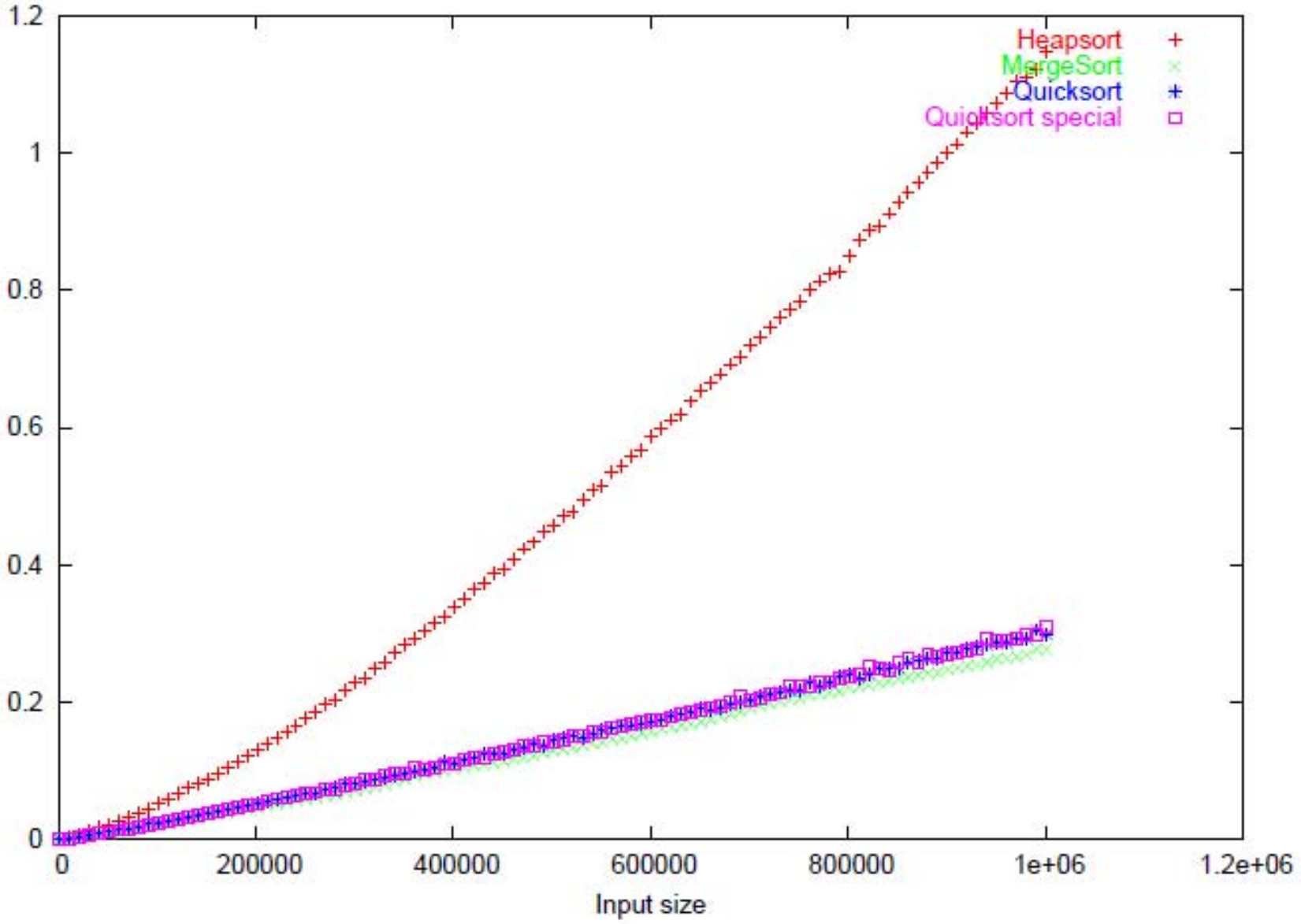
Skift til insertion-sort ved små problemstørrelser

n = 300.000 elementer

Tiden for Sorterings Algoritmer



Tiden for Sorterings Algoritmer



Algoritmer og Datastrukturer 1

Randomized-select
[CLRS, kapitel 9.1-9.2]

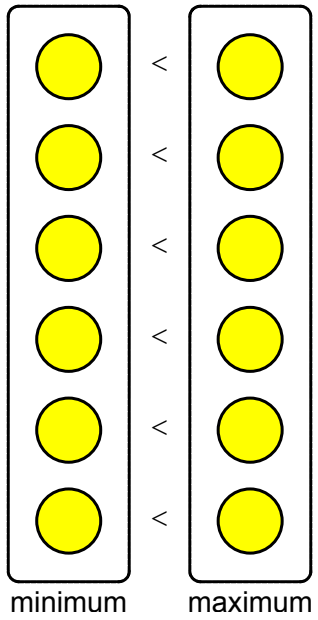
Gerth Stølting Brodal

Aarhus Universitet

Beregning af Minimum of Maximum

- At finde *minimum* af ***n*** elementer kræver ***n*-1** sammenligninger

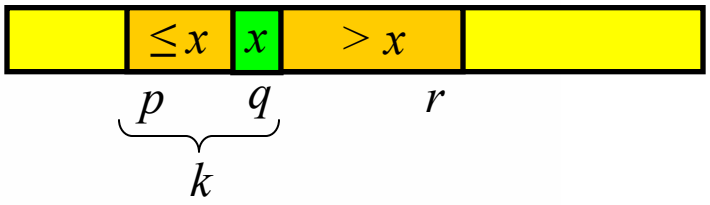
- At finde *minimum og maximum* af ***n*** elementer kræver **3/2·*n*-2** sammenligninger



Randomized Select:

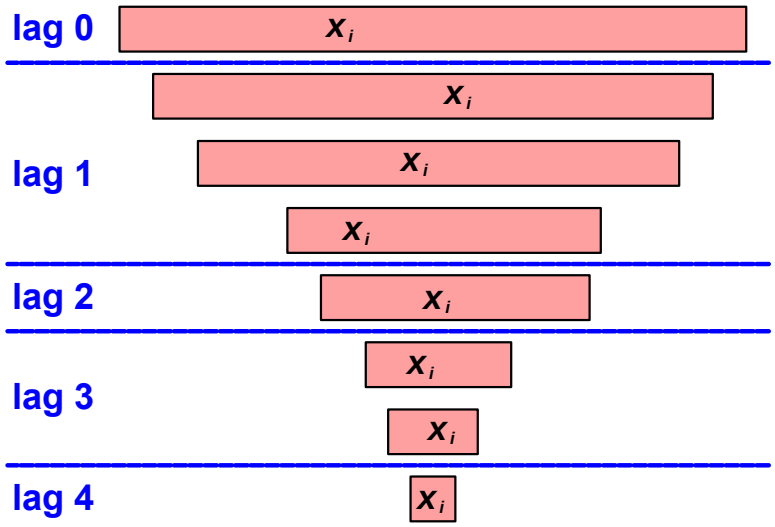
Find det *i*'te mindste element i *A*[*p*..*r*]

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p = r
2     then return A[p]
3  q ← RANDOMIZED-PARTITION(A, p, r)
4  k ← q − p + 1
5  if i = k                ▷ the pivot value is the answer
6     then return A[q]
7  elseif i < k
8     then return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```



Forventet tid O(*n*)

Randomized Select : Analyse



Forventede tid for randomized select

$= O(\Sigma_j$ **forventede tid** i lag $j)$

$\leq O(\Sigma_j\, n \cdot (3/4)^j \cdot$ # **forventede arrays** i lag $j)$

$\leq O(\Sigma_j\, n \cdot (3/4)^j \cdot 2)$

$=$ **$O(n)$**