

# Algoritmer og Datastrukturer 1

Gerth Stølting Brodal

**...mere Sortering**  
**[CLRS, kapitel 8]**



# Sorterings-algoritmer

(sammenligningsbaserede)

| Algoritme                                     | Worst-Case Tid      |
|---|---------------------|
| Heap-Sort                                     | $O(n \cdot \log n)$ |
| Merge-Sort                                    |                     |
| Insertion-Sort                                | $O(n^2)$            |
| QuickSort<br>(Deterministisk og randomiseret) | $O(n^2)$            |

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

# Hvad er en Sorterings algoritme?

## Deterministisk algoritme

- For et givet input gør algoritmen altid det samme

## Randomiseret sorterings algoritme

- Algoritmen kan lave tilfældige valg, algoritmens udførelse afhænger af både input og de tilfældige valg

## Output

- en **permutation** af input

## Sammenligningsbaseret algoritme

- output afhænger kun af sammenligninger af input elementer

# Sammenligninger for Insertion-Sort

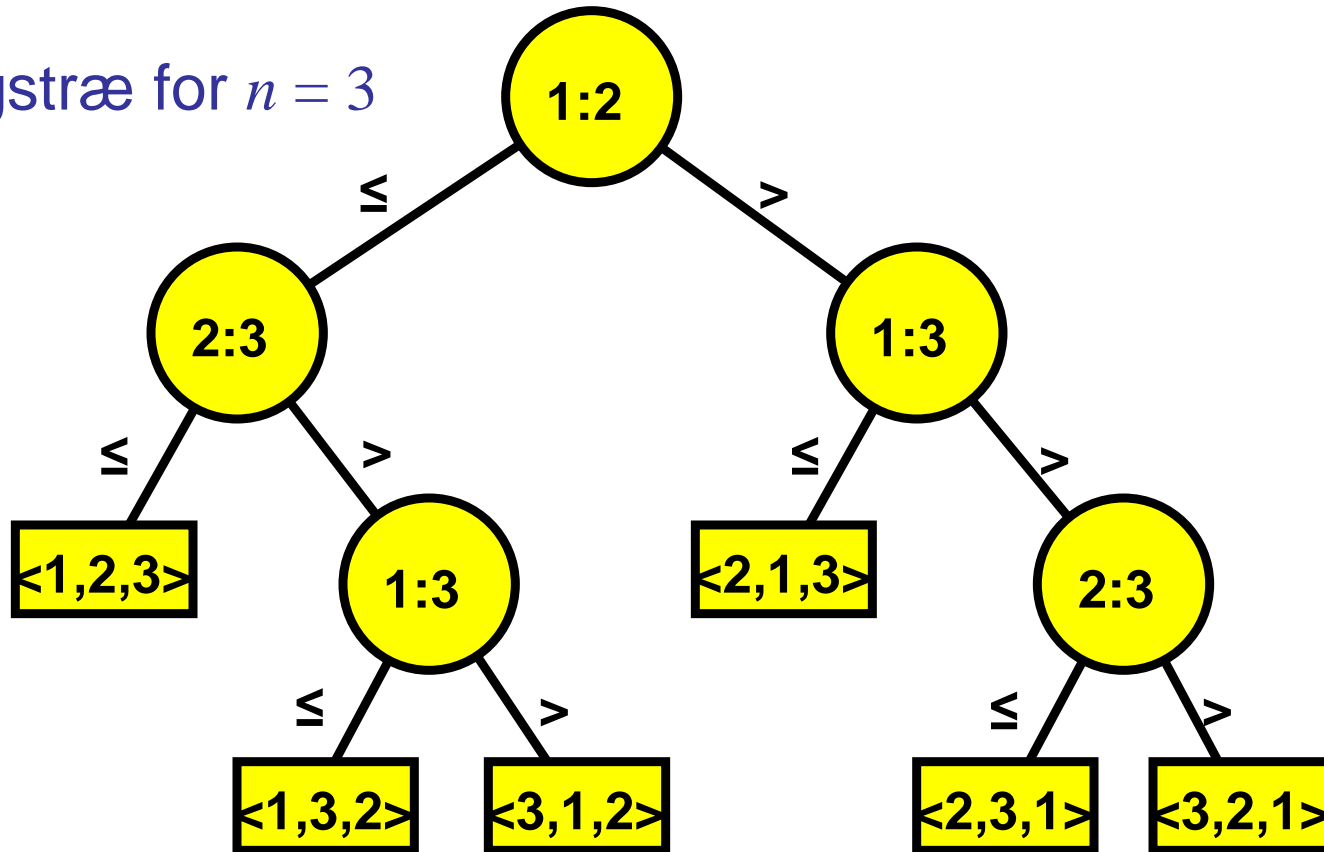
|       |       |       |
|-------|-------|-------|
| 1     | 2     | 3     |
| $x_1$ | $x_2$ | $x_3$ |

INSERTION-SORT( $A$ )

```
1  for  $j = 2$  to  $A.length$ 
2       $key = A[j]$ 
3      // Insert  $A[j]$  into the sorted sequence  $A[1 .. j - 1]$ .
4       $i = j - 1$ 
5      while  $i > 0$  and  $A[i] > key$ 
6           $A[i + 1] = A[i]$ 
7           $i = i - 1$ 
8       $A[i + 1] = key$ 
```

# Sortering ved sammenligninger: Nedre grænse

Beslutningstræ for  $n = 3$

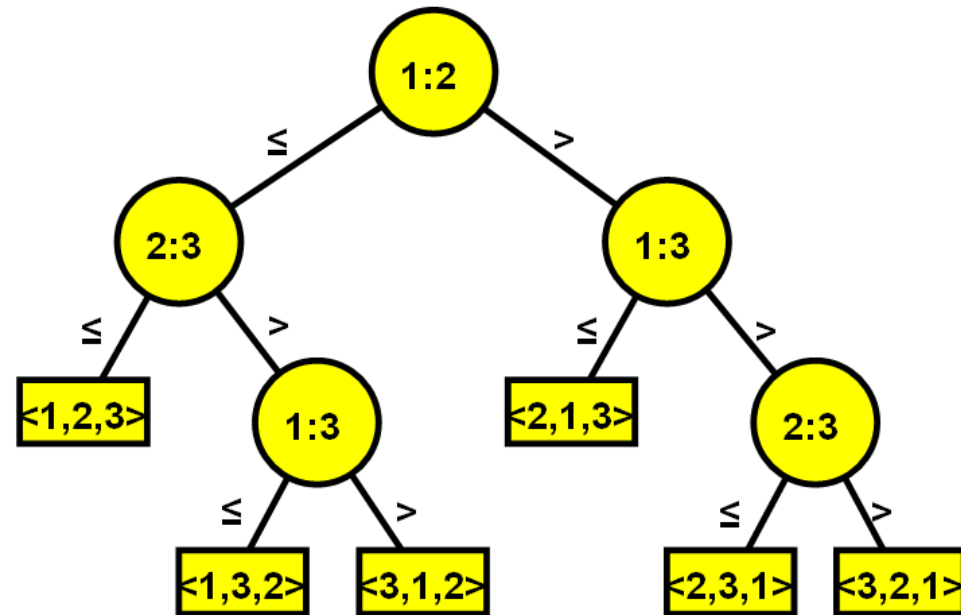


Interne knude  $i:j$  sammenligner  $x_i$  og  $x_j$   
Blade beskriver output **permutation**

# Sortering ved sammenligninger: Nedre grænse

$n!$  forskellige output  
 $\geq$  forskellige blade

træ af højde  $h$  har  
 $\leq 2^h$  blade



$$n! \leq 2^h$$

$$h \geq \log(n!) \geq \log((n/2)^{n/2}) = \Omega(n \cdot \log n)$$

**Worst-case  $\Omega(n \cdot \log n)$  sammenligninger**

# **Sortering af heltal**

**...udnyt at elementer er bitstreng**

# Counting-Sort:

**Input:**  $A$ , **output:**  $B$ , tal fra  $\{0, 1, \dots, k\}$

COUNTING-SORT( $A, B, k$ )

```
1  let  $C[0..k]$  be a new array
2  for  $i = 0$  to  $k$ 
3       $C[i] = 0$ 
4  for  $j = 1$  to  $A.length$ 
5       $C[A[j]] = C[A[j]] + 1$ 
6  //  $C[i]$  now contains the number of elements equal to  $i$ .
7  for  $i = 1$  to  $k$ 
8       $C[i] = C[i] + C[i - 1]$ 
9  //  $C[i]$  now contains the number of elements less than or equal to  $i$ .
10 for  $j = A.length$  downto 1
11      $B[C[A[j]]] = A[j]$ 
12      $C[A[j]] = C[A[j]] - 1$ 
```

**Worst-case tid**  $O(n+k)$

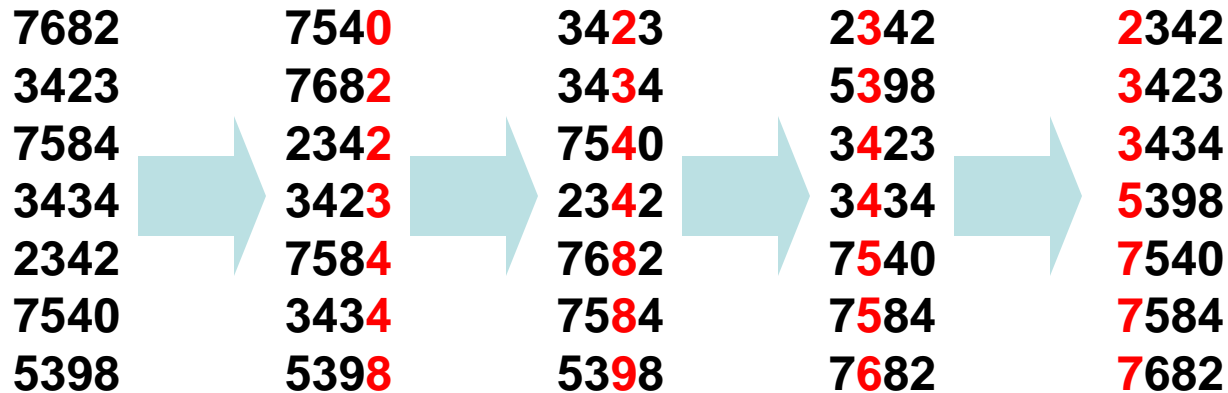


# Radix-Sort:

Input: array  $A$ , tal med  $d$  cifre fra  $\{0,1,\dots,k\}$

RADIX-SORT( $A, d$ )

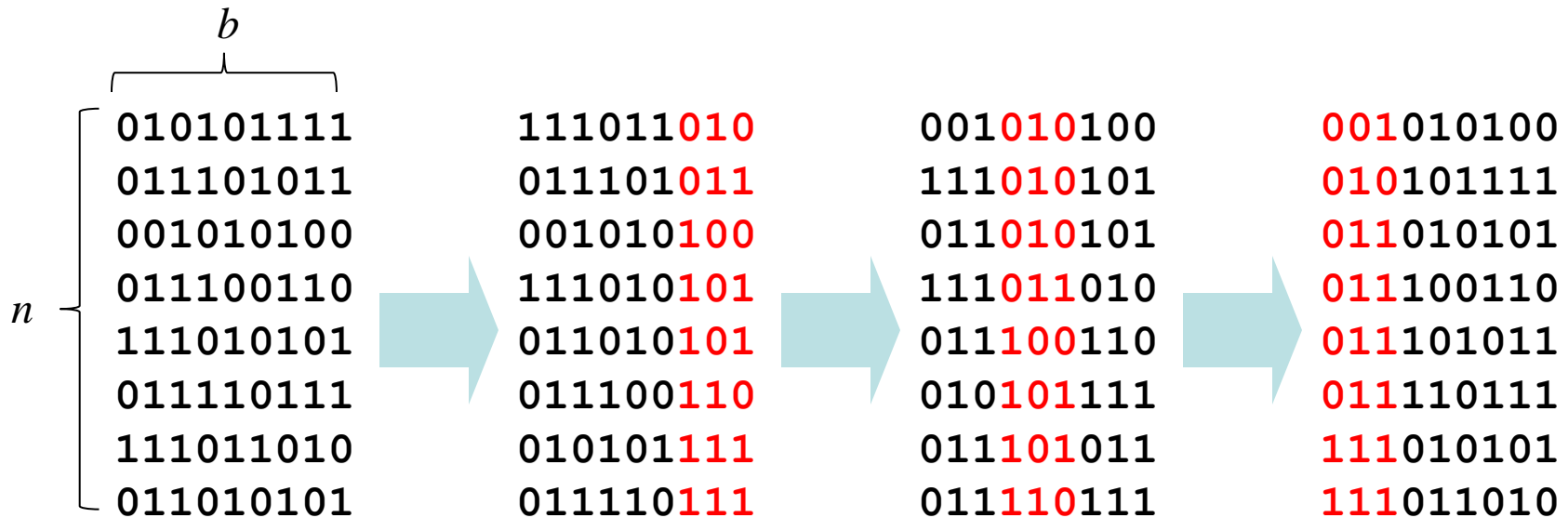
- 1 for  $i = 1$  to  $d$
- 2 use a **stable** sort to sort array  $A$  on digit  $i$



**Worst-case tid**  $O(d \cdot (n+k))$

# Radix-Sort:

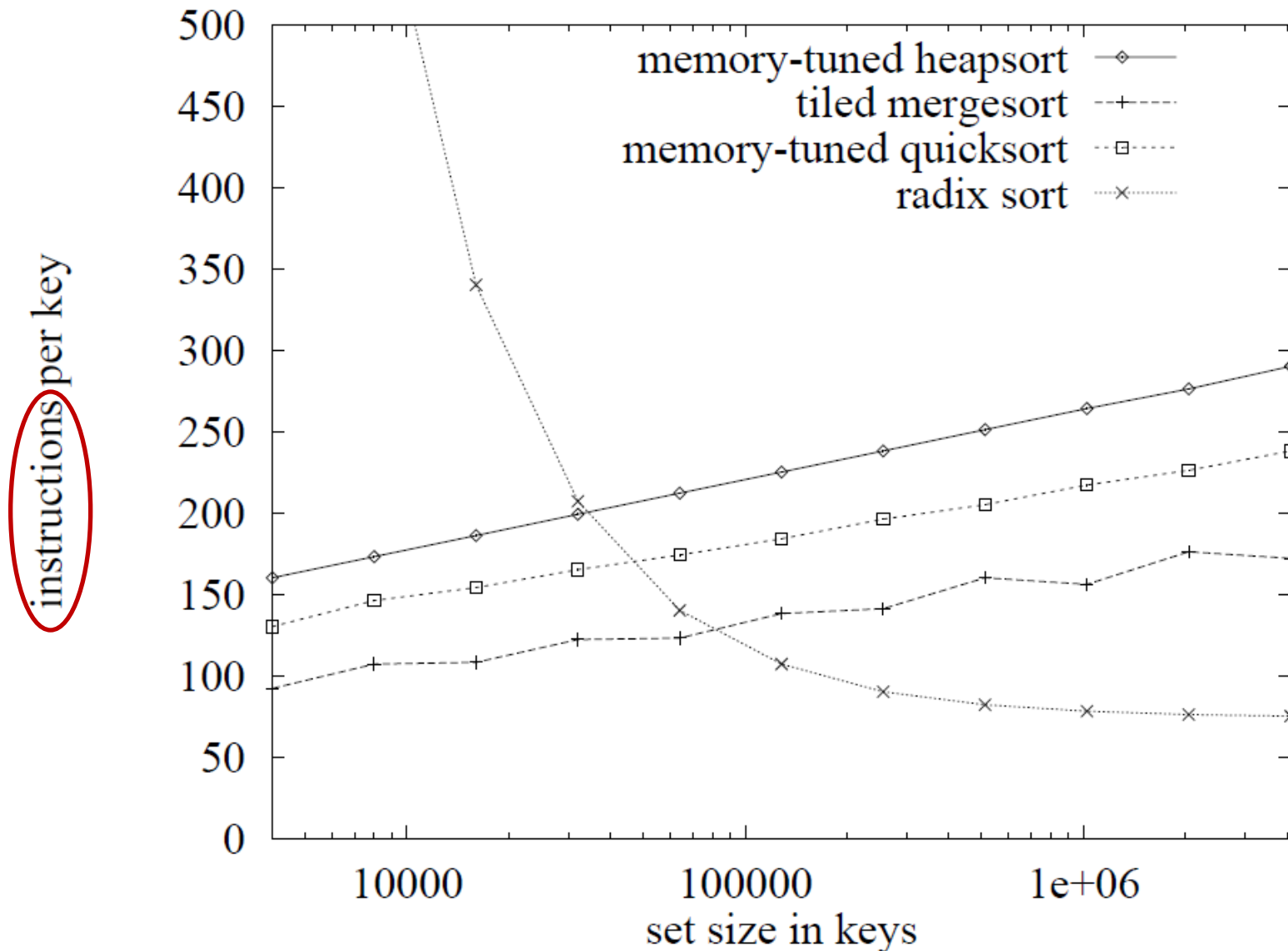
Input: array  $A$ ,  $n$  tal med  $b$  bits



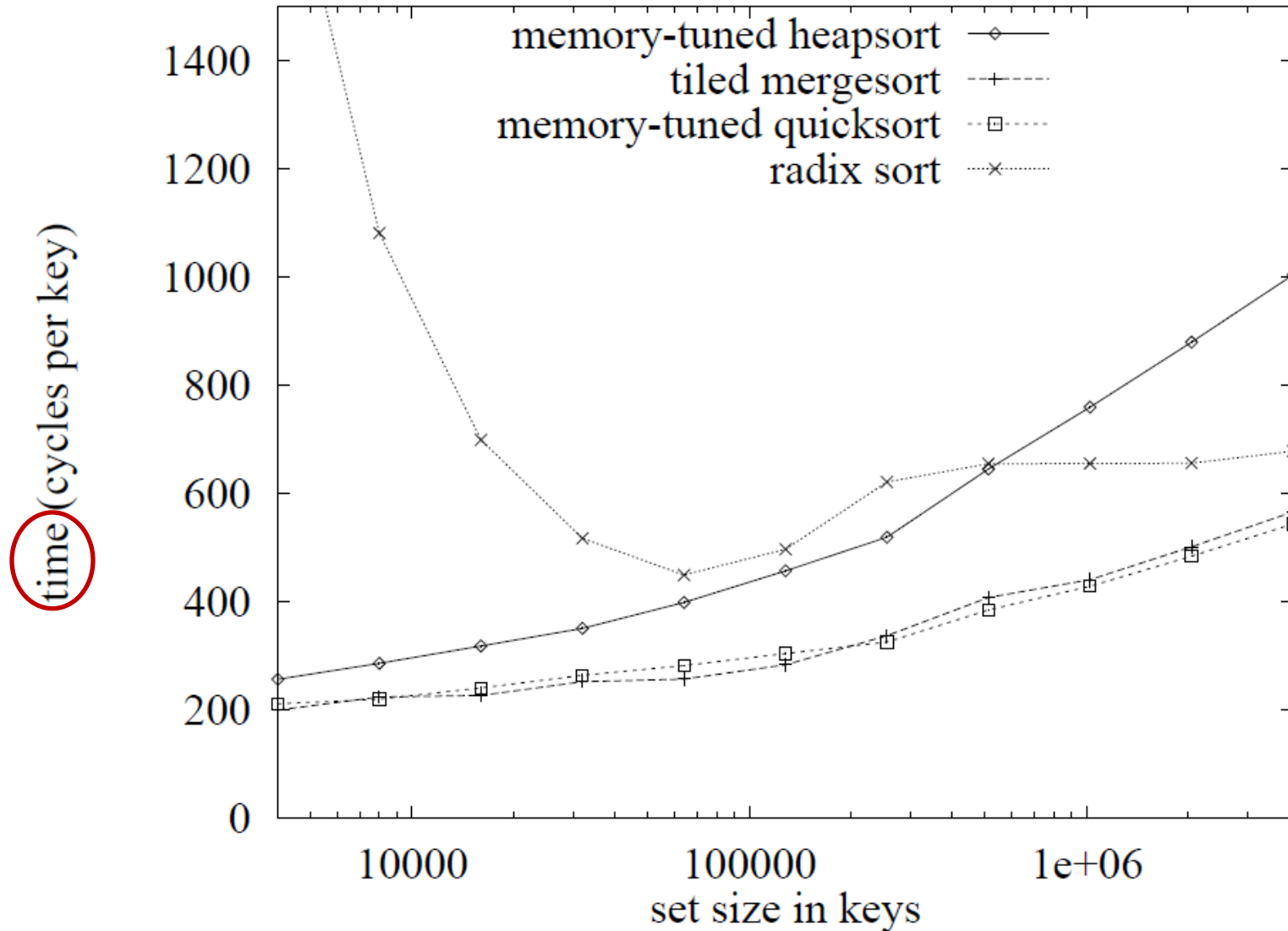
$n = 8$ ,  $k = 7$  (3 bits =  $\log n$  bits),  $d = 3$  (3 x 3 bits)

Input  $n$  tal med  $b$  bits: **Worst-case tid  $O(n \cdot b / \log n)$**

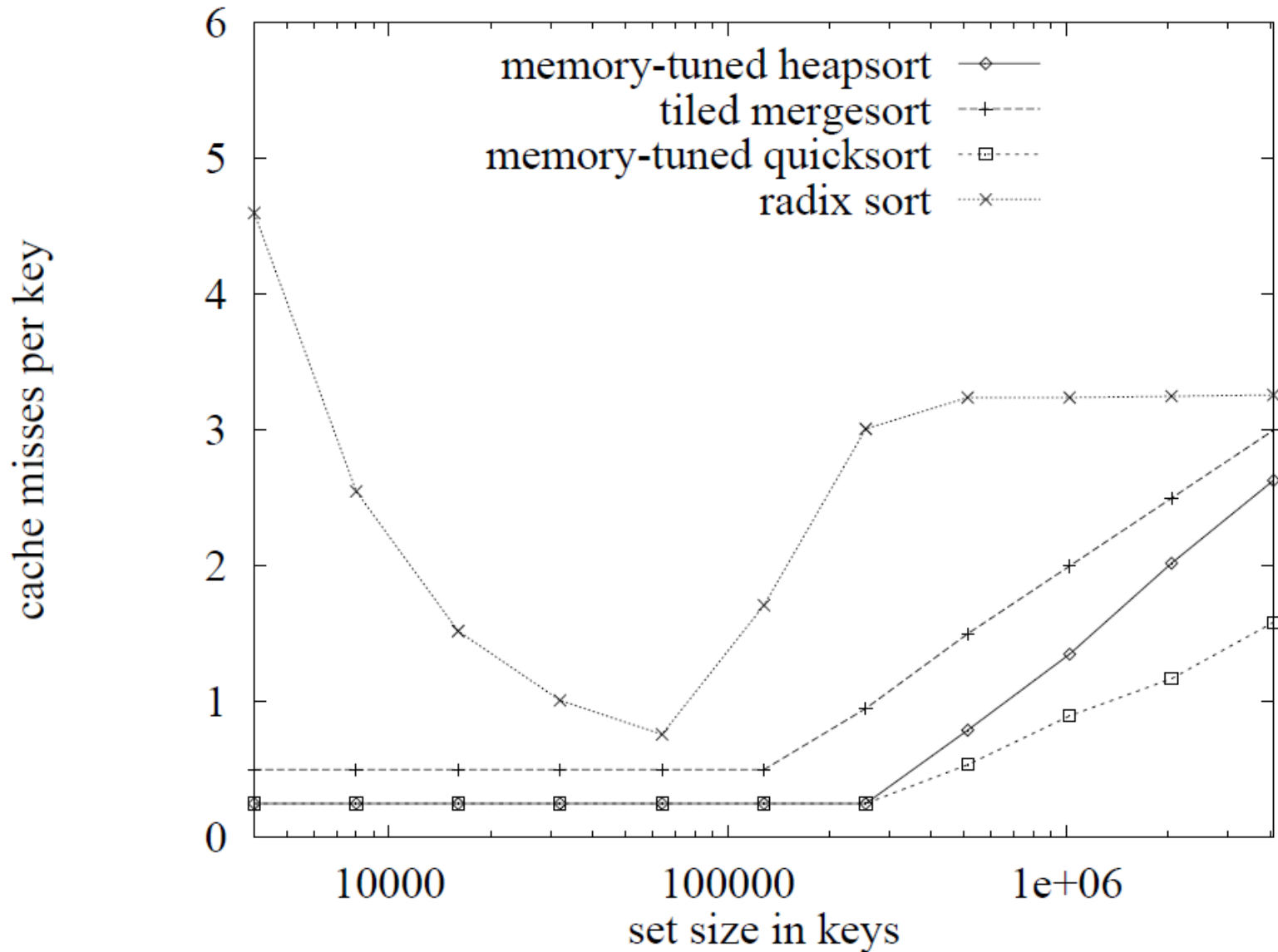
# Radix-Sort: Eksperimenter



# Radix-Sort: Eksperimenter



# Radix-Sort: Eksperimenter

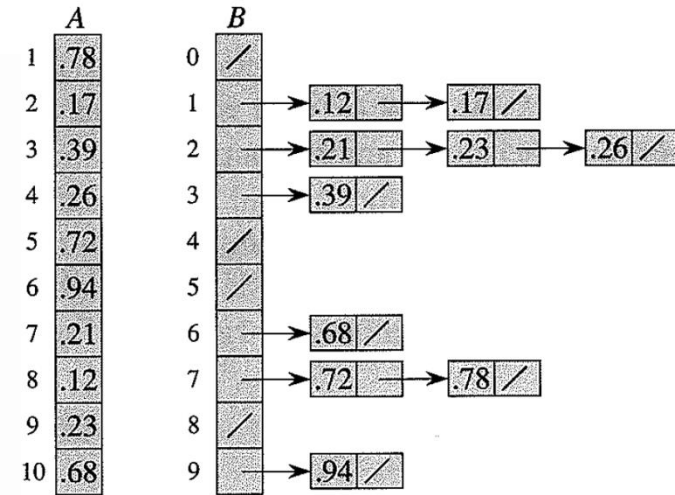


# Bucket-Sort:

Input:  $A$ , reelle tal fra  $[0..1]$

BUCKET-SORT( $A$ )

- 1 let  $B[0..n - 1]$  be a new array
- 2  $n = A.length$
- 3 for  $i = 0$  to  $n - 1$
- 4     make  $B[i]$  an empty list
- 5 for  $i = 1$  to  $n$
- 6     insert  $A[i]$  into list  $B[\lfloor n A[i] \rfloor]$
- 7 for  $i = 0$  to  $n - 1$
- 8     sort list  $B[i]$  with insertion sort
- 9 concatenate the lists  $B[0], B[1], \dots, B[n - 1]$  together in order



**Forventet tid  $O(n)$  – for tilfældigt input**

# Sortering – State-of-the-Art

Bedst kendte sorterings grænse for RAM modellen (med ubegrænset ordstørrelse) er en randomiseret algoritme der bruger  $O(n)$  plads og har en forventet køretid på:

$$O(n\sqrt{\log \log n})$$

Om dette kan opnås uden randomisering er ukendt.  
 $O(n \log \log n)$  er kendt. Kan man opnå forventet  $O(n)$  tid?  
Med randomisering og for ordstørrelse  $\Omega(\log^{2+\varepsilon} n)$  findes en randomiseret algoritme med  $O(n)$  tid.

Yijie Han, Mikkel Thorup: *Integer Sorting in  $O(n\sqrt{\log \log n})$  Expected Time and Linear Space.* Proc. 43rd Symposium on Foundations of Computer Science, 135-144, 2002.  
Arne Andersson, Torben Hagerup, Stefan Nilsson, Rajeev Raman: *Sorting in linear time?* Proc. 27th ACM Symposium on Theory of Computing, 427-436, 1995.