Grundlæggende Algoritmer og Datastrukturer

...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 (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 udførsel afhænger af både input og de tilfældige valg

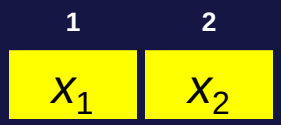
Output

- en permutation af input

Sammenligningsbaseret algoritme

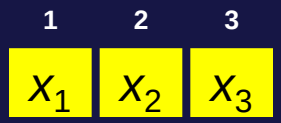
- output afhænger kun af sammenligninger af input elementer

Antal sammenligninger for korrekt at sortere 2 elementer ?



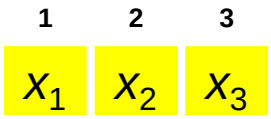
a) ingen sammenligninger
b) mindst 1 sammenligning
c) mindst 2 sammenligninger
d) ved ikke

Antal sammenligninger for korrekt at sortere 3 elementer ?



a) ingen sammenligninger
b) mindst 1 sammenligning
c) mindst 2 sammenligninger
d) mindst 3 sammenligninger
e) mindst 4 sammenligninger
f) mindst 5 sammenligninger
g) ved ikke

Sammenligninger for Insertion-Sort

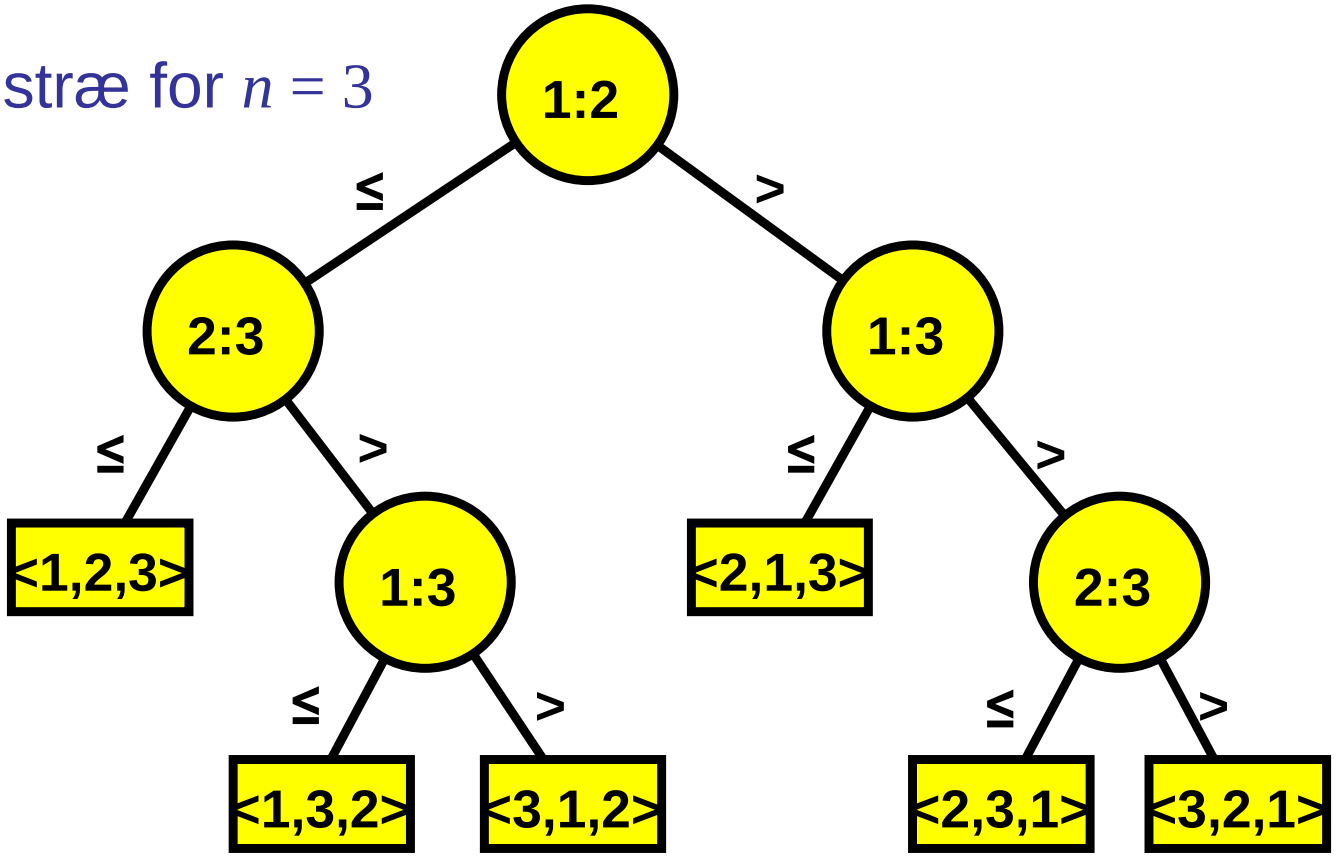


INSERTION-SORT(A)

```
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 sammmenligninger: Nedre grænse

Beslutningstræ for $n = 3$



Interne knude $i{:}j$ sammenligner x_i og x_j

Blade beskriver output **permutation**

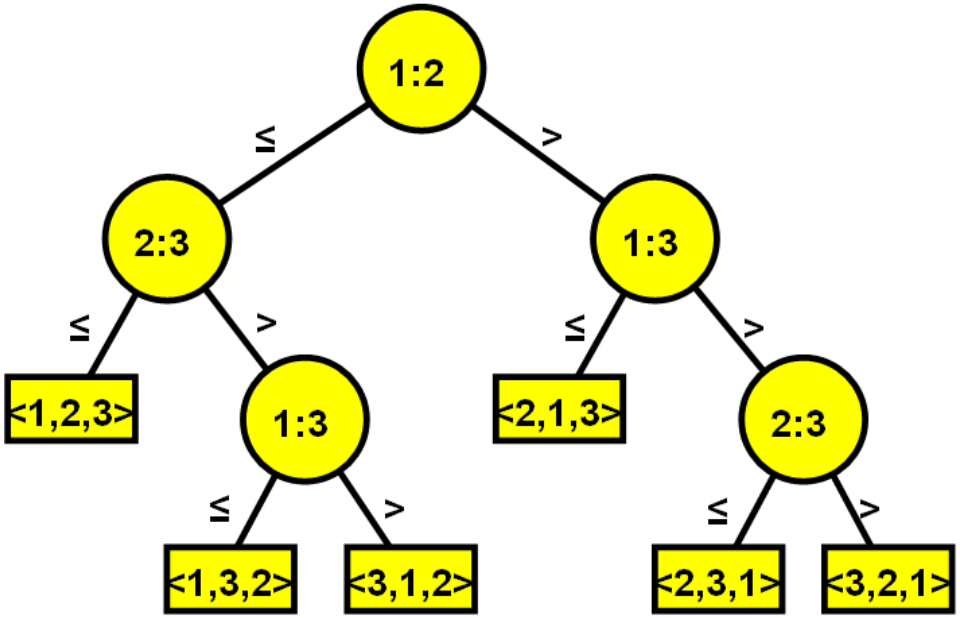
Sortering ved sammmenligninger: Nedre grænse

$n!$ 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

Antal sammenligninger for at sortere 10 elementer ?

a) mindst 9 sammenligning
b) mindst 10 sammenligninger
c) mindst 21 sammenligninger
d) mindst 22 sammenligninger
e) mindst 23 sammenligninger
f) ved ikke

n	1	2	3	4	5	6	7	8	9	10	11	12
$n!$	1	2	6	24	120	720	5.040	40.320	362.880	3.628.800	39.916.800	479.001.600

d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2^d	1	2	4	8	16	32	64	128	256	512	1.024	2.048	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576

d	21	22	23	24	25	26	27	28	29
2^d	2.097.152	4.194.304	8.388.608	16.777.216	33.554.432	67.108.864	134.217.728	268.435.456	536.870.912

Antal sammenligninger for at sortere 1-12 elementer ?

n	1	2	3	4	5	6	7	8	9	10	11	12
≥ sammenligninger	0	1	3	5	7	10	13	16	19	22	26	29

n	1	2	3	4	5	6	7	8	9	10	11	12
$n!$	1	2	6	24	120	720	5.040	40.320	362.880	3.628.800	39.916.800	479.001.600

d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2^d	1	2	4	8	16	32	64	128	256	512	1.024	2.048	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576

d	21	22	23	24	25	26	27	28	29
2^d	2.097.152	4.194.304	8.388.608	16.777.216	33.554.432	67.108.864	134.217.728	268.435.456	536.870.912

Sortering af heltal

...udnyt at elementer er bitstrenge

Counting-Sort:

Input: A, ouput: B, tal fra $\{0,1, \ldots,k\}$

```
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 $O(n+k)$

Hvilke algoritmer er ikke stabile ?

a) InsertionSort
b) HeapSort
c) QuickSort
d) MergeSort
e) HeapSort og Quicksort
f) InsertionSort og HeapSort
g) MergeSort og QuickSort
h) InsertionSort og QuickSort
i) ved ikke

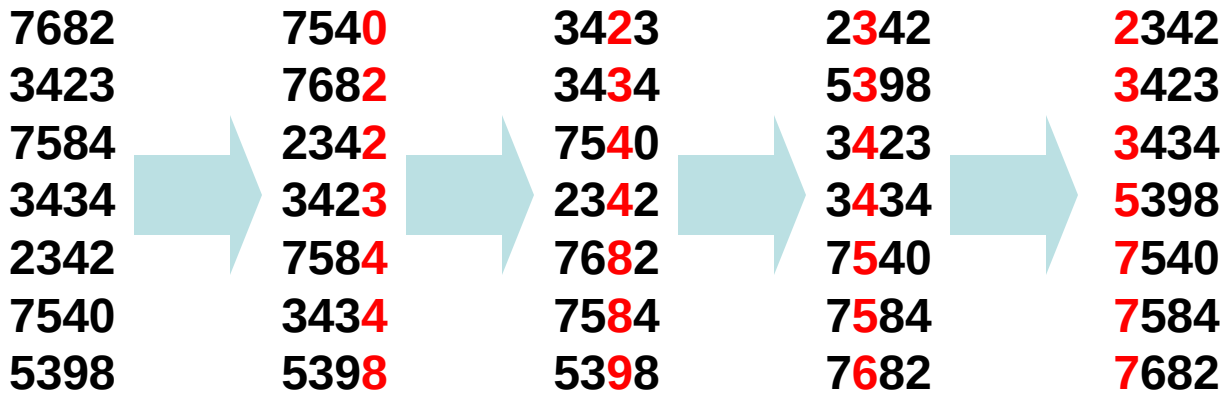
Radix-Sort:

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

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 $O(d \cdot (n+k))$

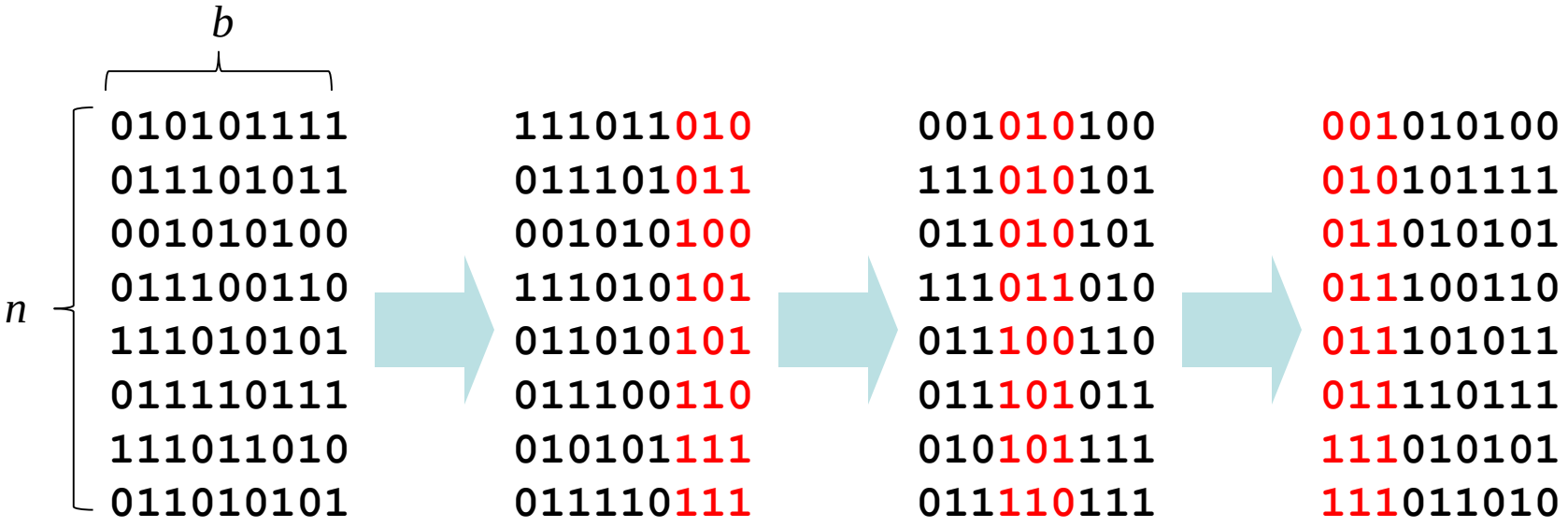
RadixSort hos POST

Trin 1: Sorter efter husnummer

Trin 2: Sorter efter rute

Radix-Sort:

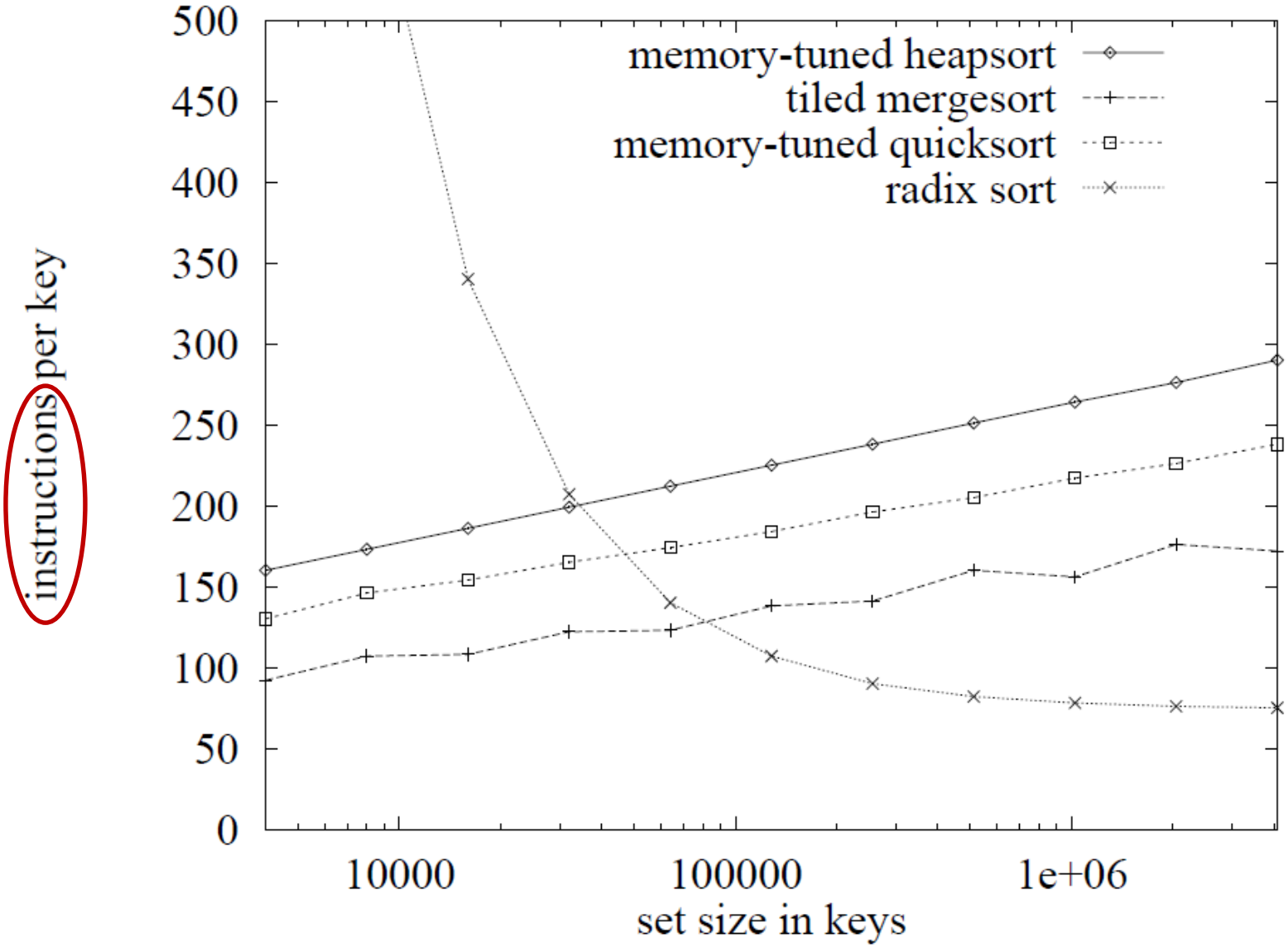
Input: array A, n tal med b bits



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

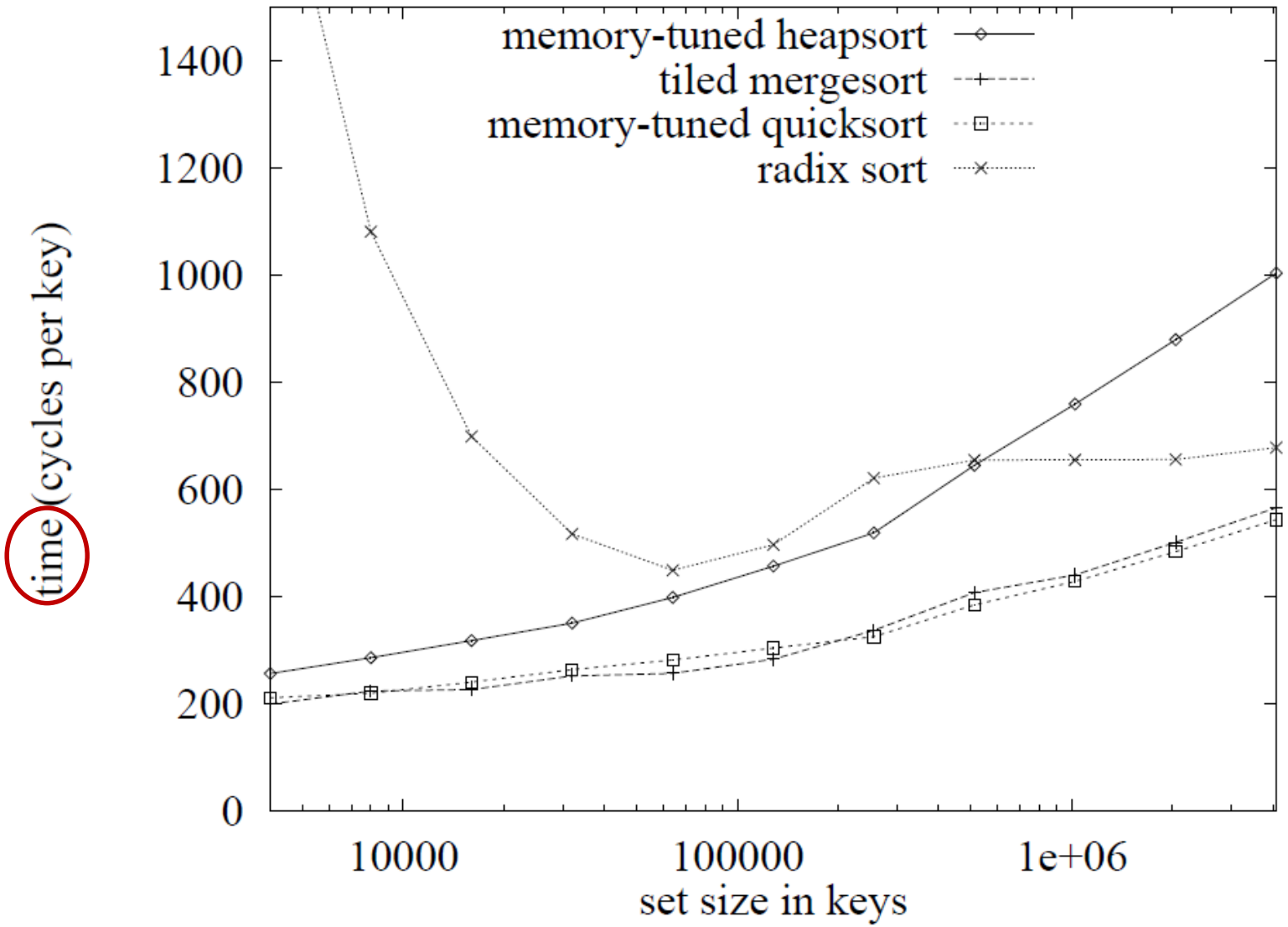
Input n tal med b bits: **Worst-case tid** $O(n \cdot b/\log n)$

Radix-Sort: Eksperimenter



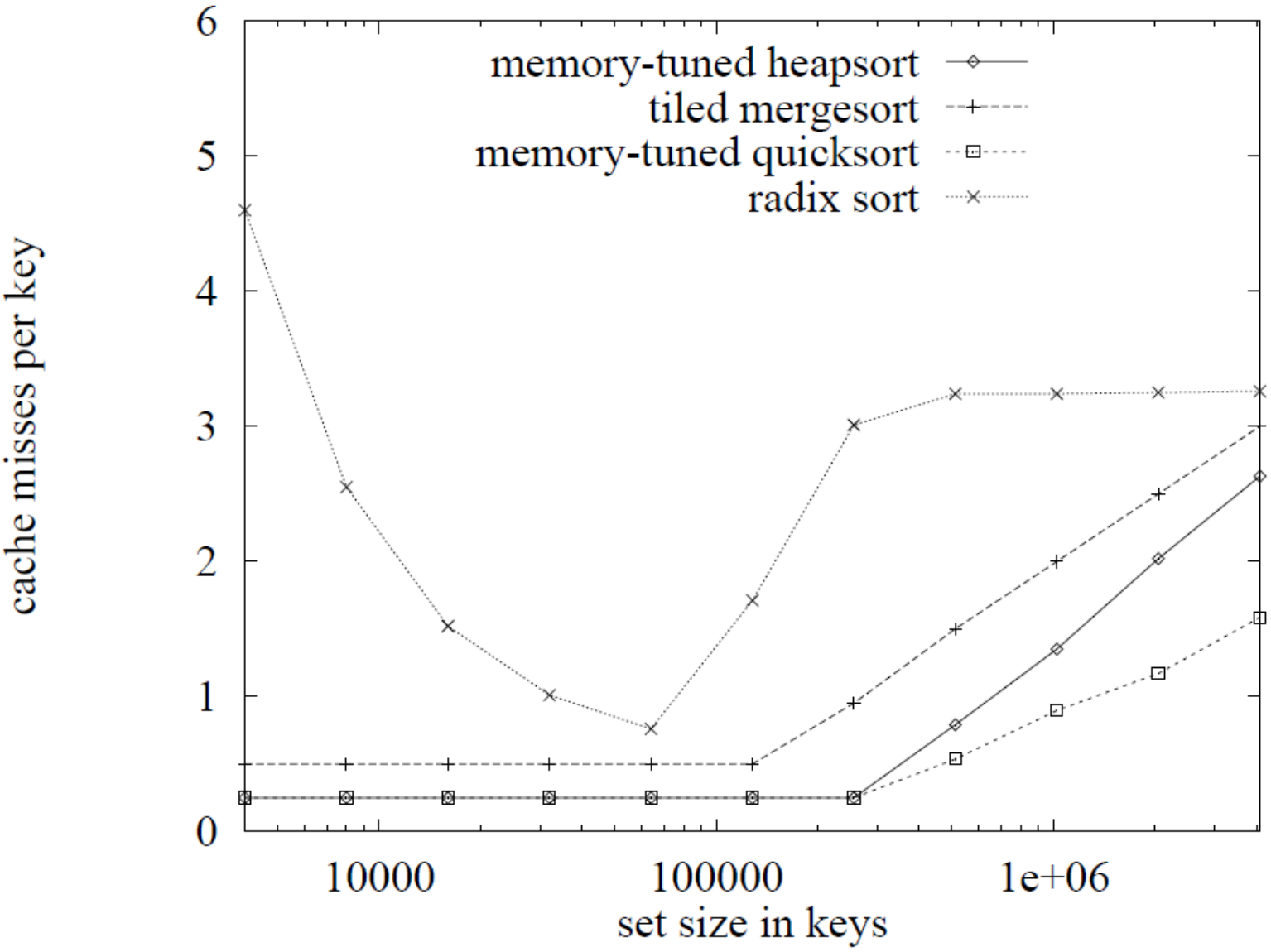
LaMarca, Ladner ’97: The influence of Caches on the Performance of Sorting

Radix-Sort: Eksperimenter



LaMarca, Ladner ’97

Radix-Sort: Eksperimenter

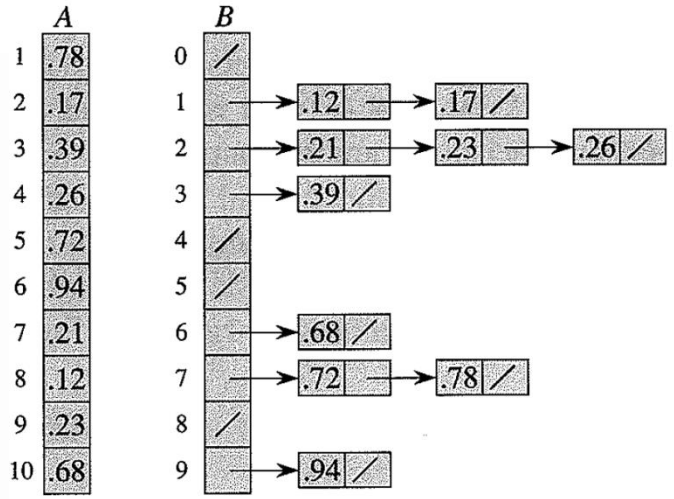


LaMarca, Ladner ’97

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[⌊n A[i]⌋]
7  for i = 0 to n − 1
8       sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], ..., 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* loglog *n*) er kendt. Kan man opnå forventet O(*n*) tid?
Med randomisering og for ordstørrelse $\Omega(\log^2 n \cdot \log\log n)$ findes en randomiseret algoritme med forventet 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.

Djamal Belazzougui, Gerth Stølting Brodal, and Jesper Sindahl Nielsen: *Expected Linear Time Sorting for Word Size* $\Omega(\log^2 n \cdot \log\log n)$. SWAT 2014.