Algoritmer og Datastrukturer 1

Gerth Stølting Brodal

...mere Sortering
[CLRS, kapitel 8]

AARHUS UNIVERSITET

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

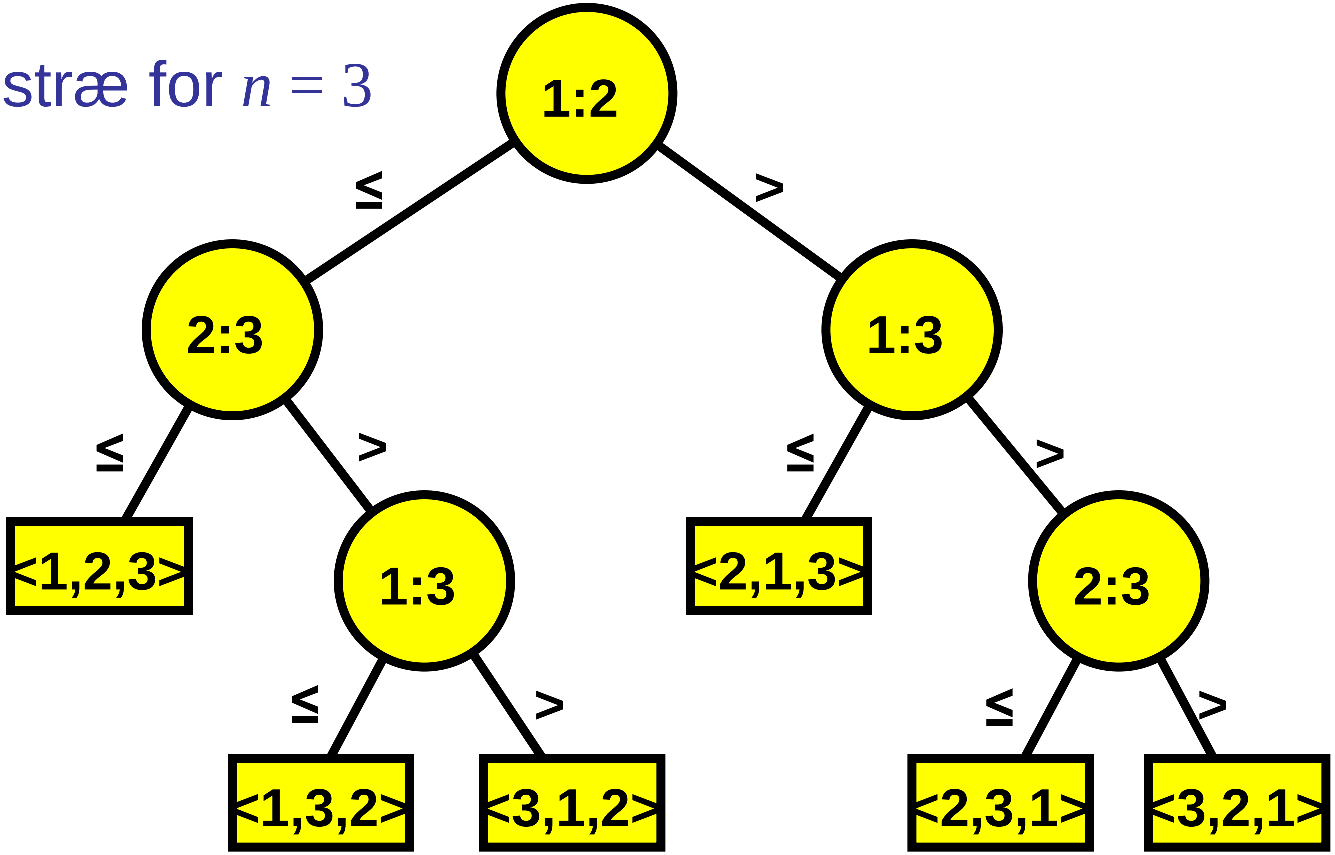
Sammenligninger for Insertion-Sort

1	2	3
x_1	x_2	x_3

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted sequence A[1 .. j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

Sortering ved sammmenligninger: Nedre grænse

Beslutningstræ for $n = 3$



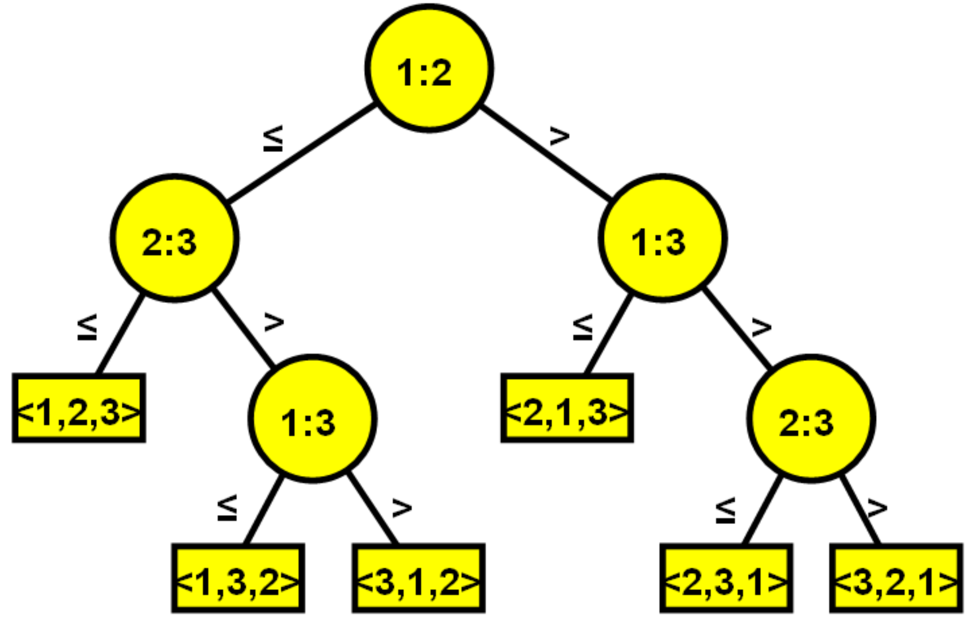
Interne knude $i{:}j$ sammenligner x_i og x_j

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

Sortering af heltal

...udnyt at elementer er bitstrenge

Counting-Sort:

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

```
COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i - 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] - 1
```

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

Radix-Sort:

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

RADIX-SORT(A, d)

for i = 1 **to** d

    use a $\boxed{\text{stable}}$ sort to sort array A on digit i

7682	7540	3423	2342	2342
3423	7682	3434	5398	3423
7584	2342	7540	3423	3434
3434	3423	2342	3434	5398
2342	7584	7682	7540	7540
7540	3434	7584	7584	7584
5398	5398	5398	7682	7682

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

RadixSort hos POST

Trin 1: Sorter efter husnummer

Trin 2: Sorter efter rute

Radix-Sort:

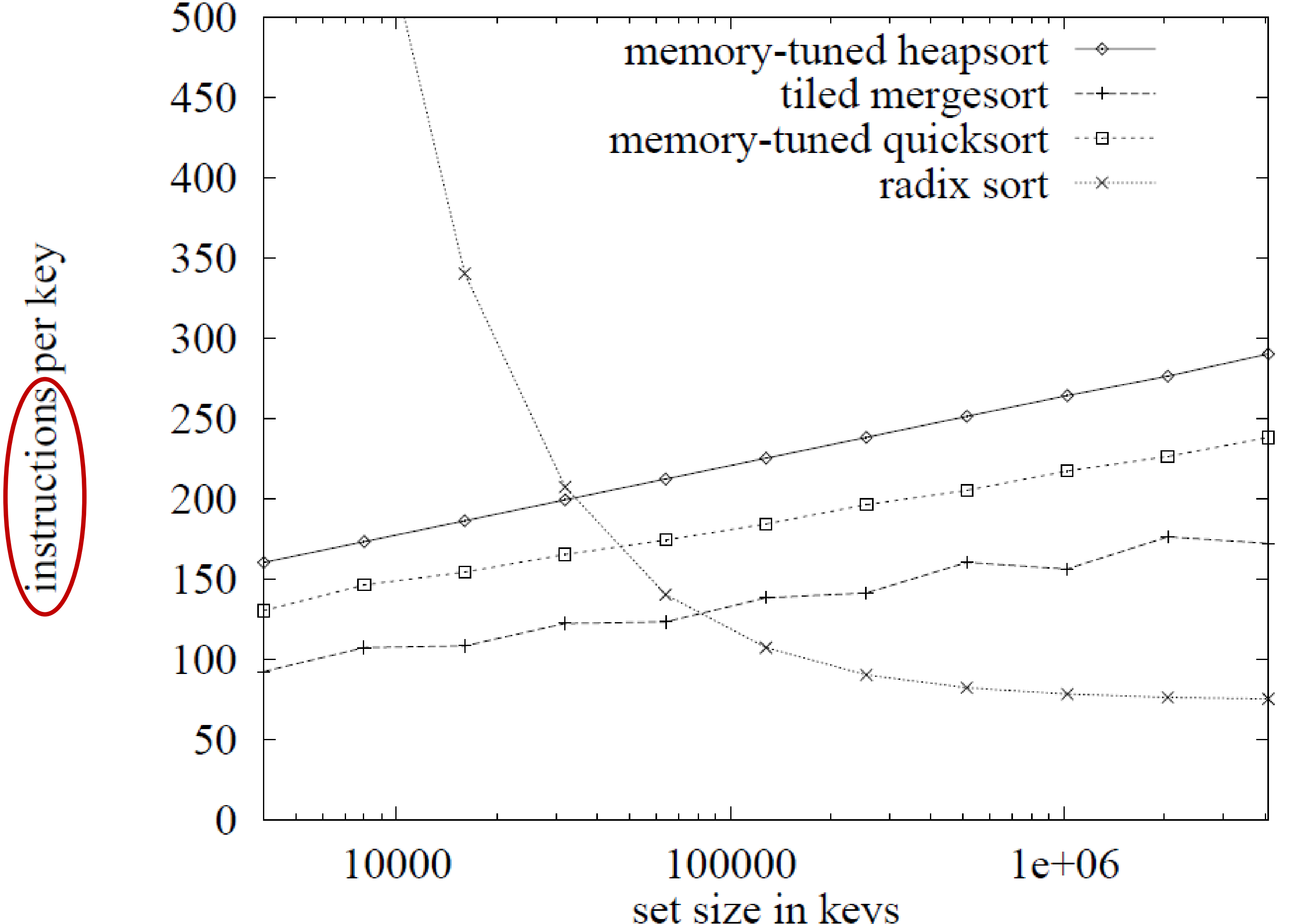
Input: array A, n tal med b bits

010101111	111011010	001010100	001010100
011101011	011101011	111010101	010101111
001010100	001010100	011010101	011010101
011100110	111010101	111011010	011100110
111010101	011010101	011100110	011101011
011110111	011100110	010101111	011110111
111011010	010101111	011101011	111010101
011010101	011110111	011110111	111011010

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

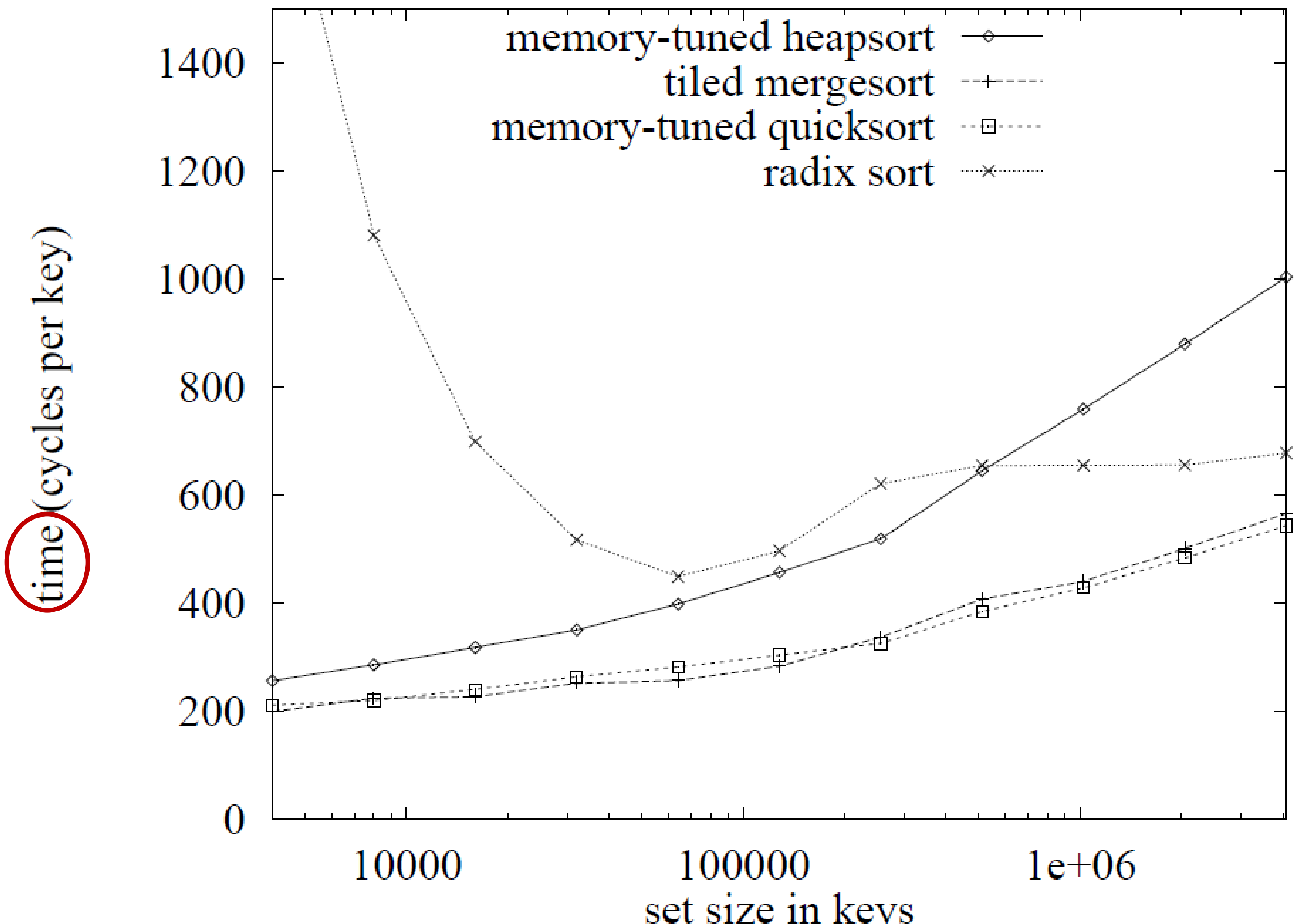
Input n tal med b bits: **Worst-case tid** $O(n \cdot b / \log n)$

Radix-Sort: Eksperimenter



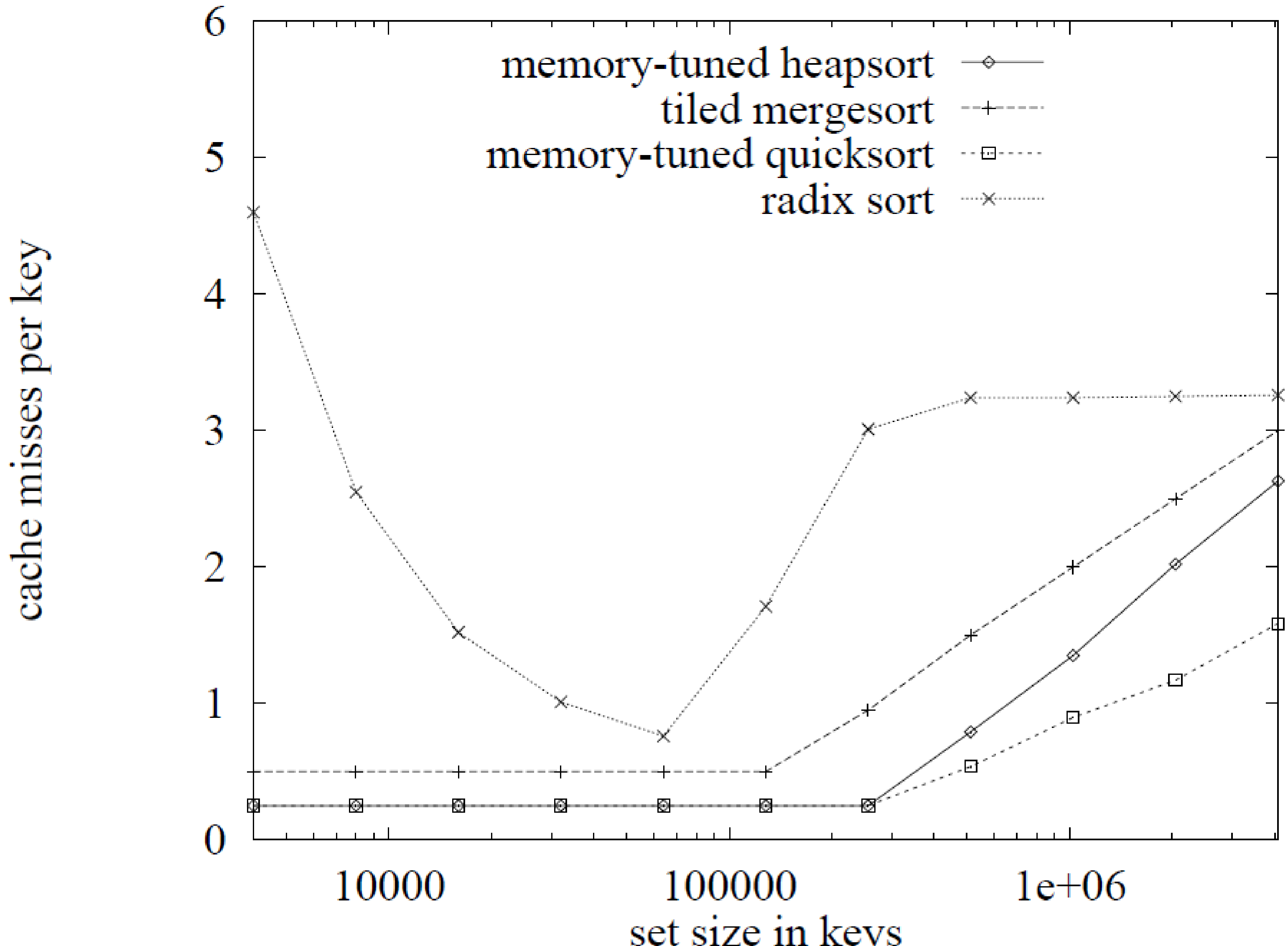
LaMarca, Ladner '97: The influence of Caches on the Performance of Sorting

Radix-Sort: Eksperimenter



LaMarca, Ladner '97

Radix-Sort: Eksperimenter

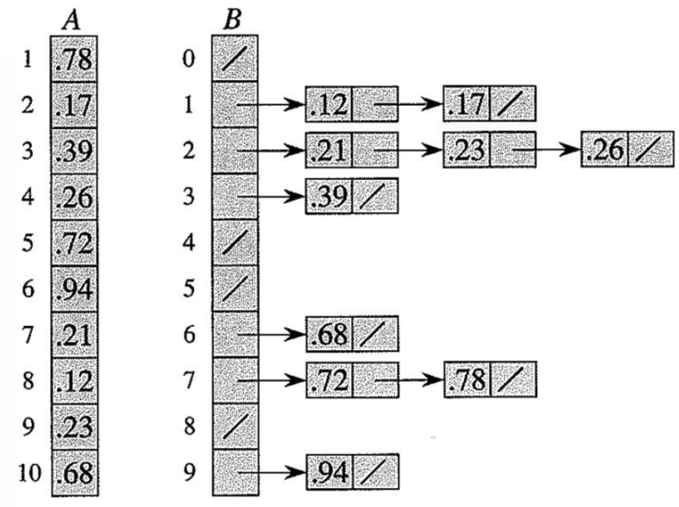


LaMarca, Ladner ’97

Bucket-Sort:

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

```
BUCKET-SORT(A)
let B[0 . . n − 1] be a new array
n = A.length
for i = 0 to n − 1
    make B[i] an empty list
for i = 1 to n
    insert A[i] into list B[⌊n A[i]⌋]
for i = 0 to n − 1
    sort list B[i] with insertion sort
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 $\mathrm{O}(n)$ plads og har en forventet køretid på:

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

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 $\Omega(\log^{2+\varepsilon} n)$ $\Omega(\log^2 n \cdot \log\log n)$ findes en randomiseret algoritme med O(*n*) tid.

Yijie Han, Mikkel Thorup: *Integer Sorting in* $\mathrm{O}(n\sqrt{\log\log n})$ *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.