Algoritmer og Datastrukturer

...mere Sortering
[CLRS, kapitel 8]

Sorterings-algoritmer
(sammenligningsbaserede)

Algoritme	Worst-Case Tid
Heap-Sort	$O(n \cdot \log n)$
Merge-Sort	$O(n \cdot \log n)$
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

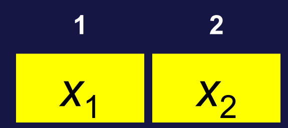
Output

- 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 ?

1	2	3
x_1	x_2	x_3

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

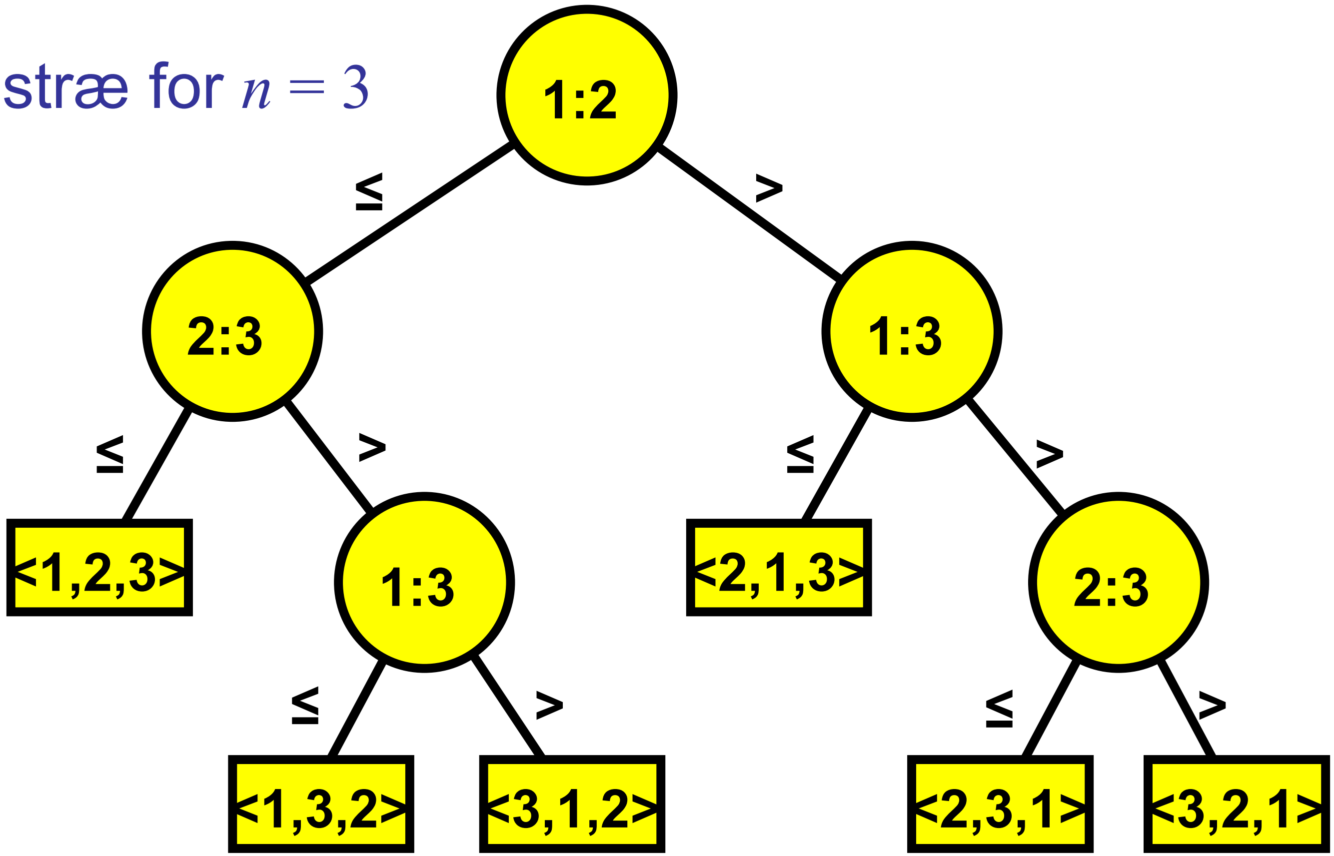
Sammenligninger for Insertion-Sort

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

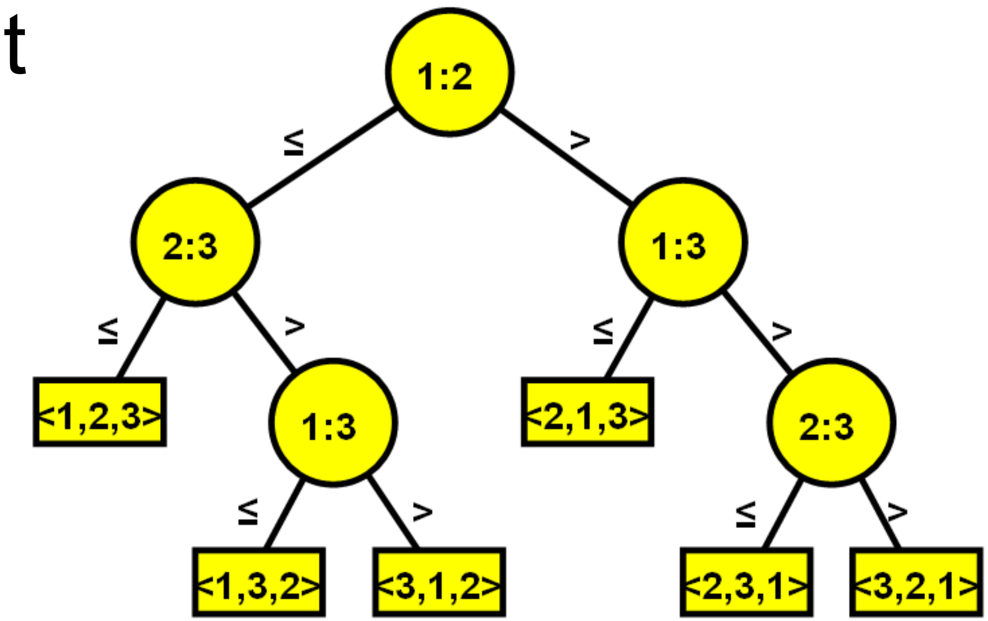
$n!$ forskellige output

$\leq$ forskellige blade

træ af højde h har

$\leq 2^h$ blade

$$n! \leq 2^h$$



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

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)
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 $O(n+k)$

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 $\{0,1,\ldots,k\}$

```
RADIX-SORT(A, d)
for i = 1 to d
    use a 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:

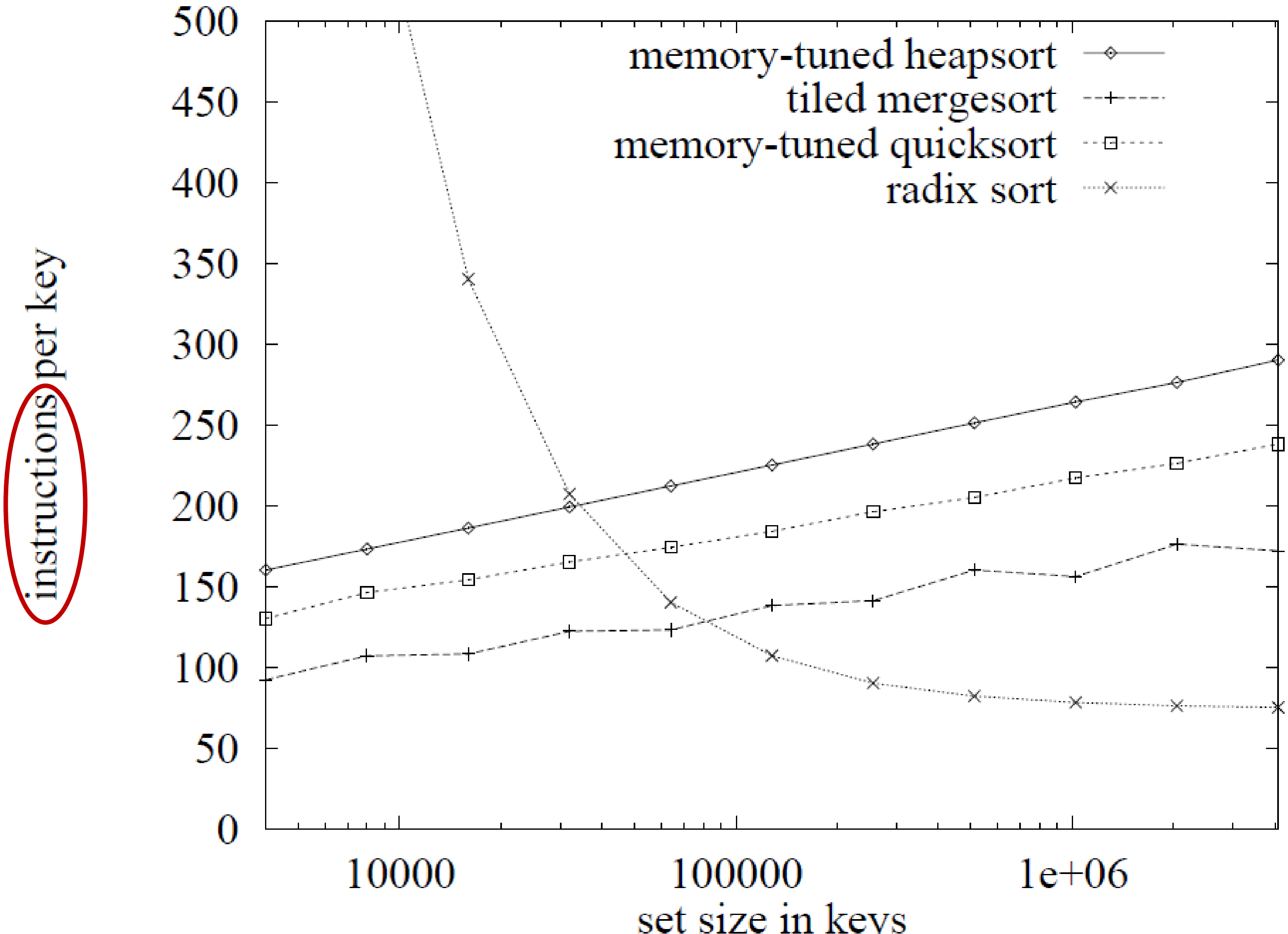
Input: array A, n tal med b bits

Input (n tal, b bits)	→		→		→	
010101111		111011010		001010100		001010100
011101011		011101011		111010101		010101111
001010100		001010100		011010101		011010101
011100110		111010101		111011010		011100110
111010101		011010101		011100110		011101011
011110111		011100110		011101011		011110111
111011010		010101111		010101111		111010101
011010101		011110111		011110111		111011010

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

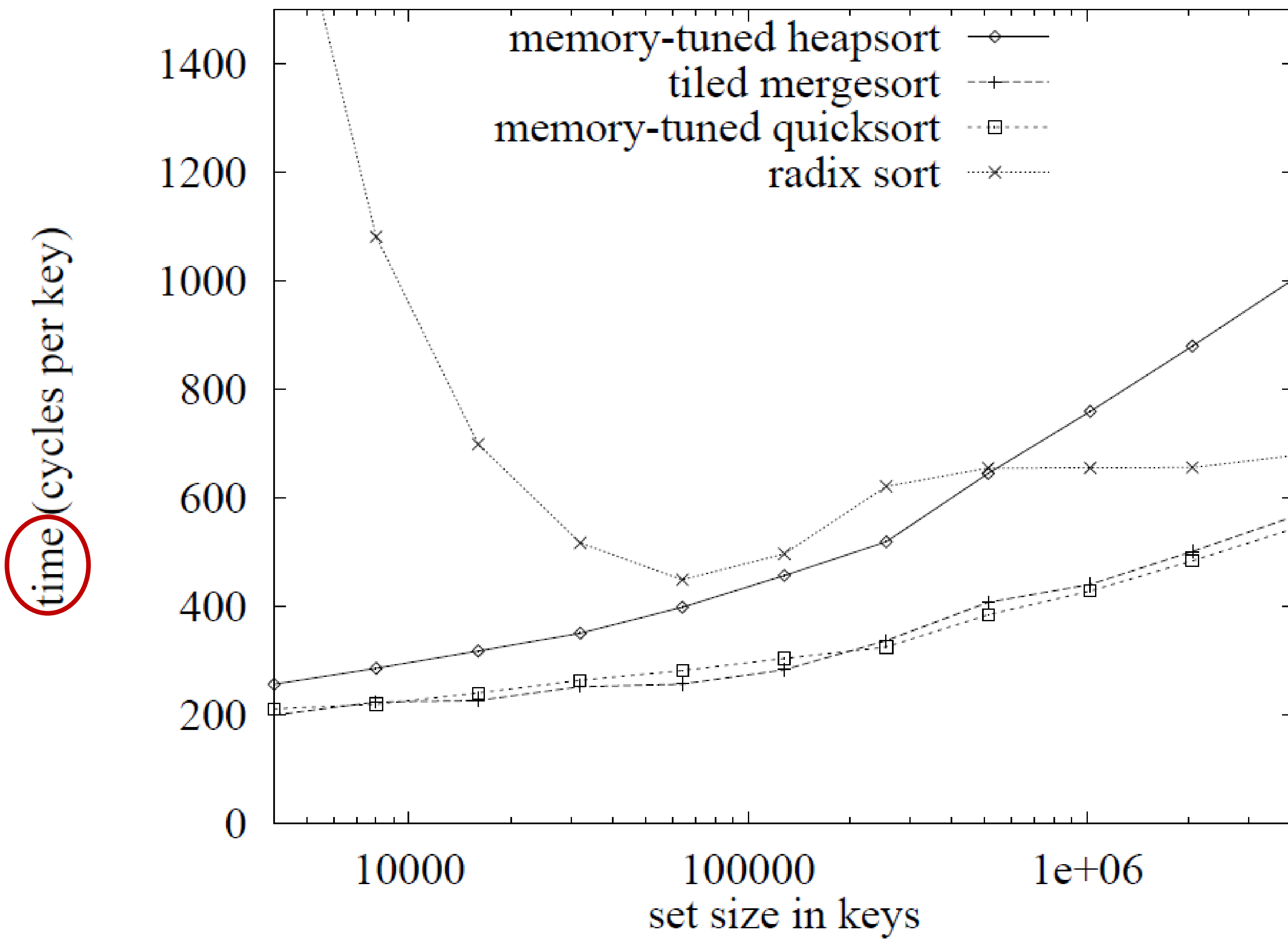
Input n tal med b bits: **Worst-case tid** $O(n \cdot b / \log n)$

Radix-Sort: Eksperimenter



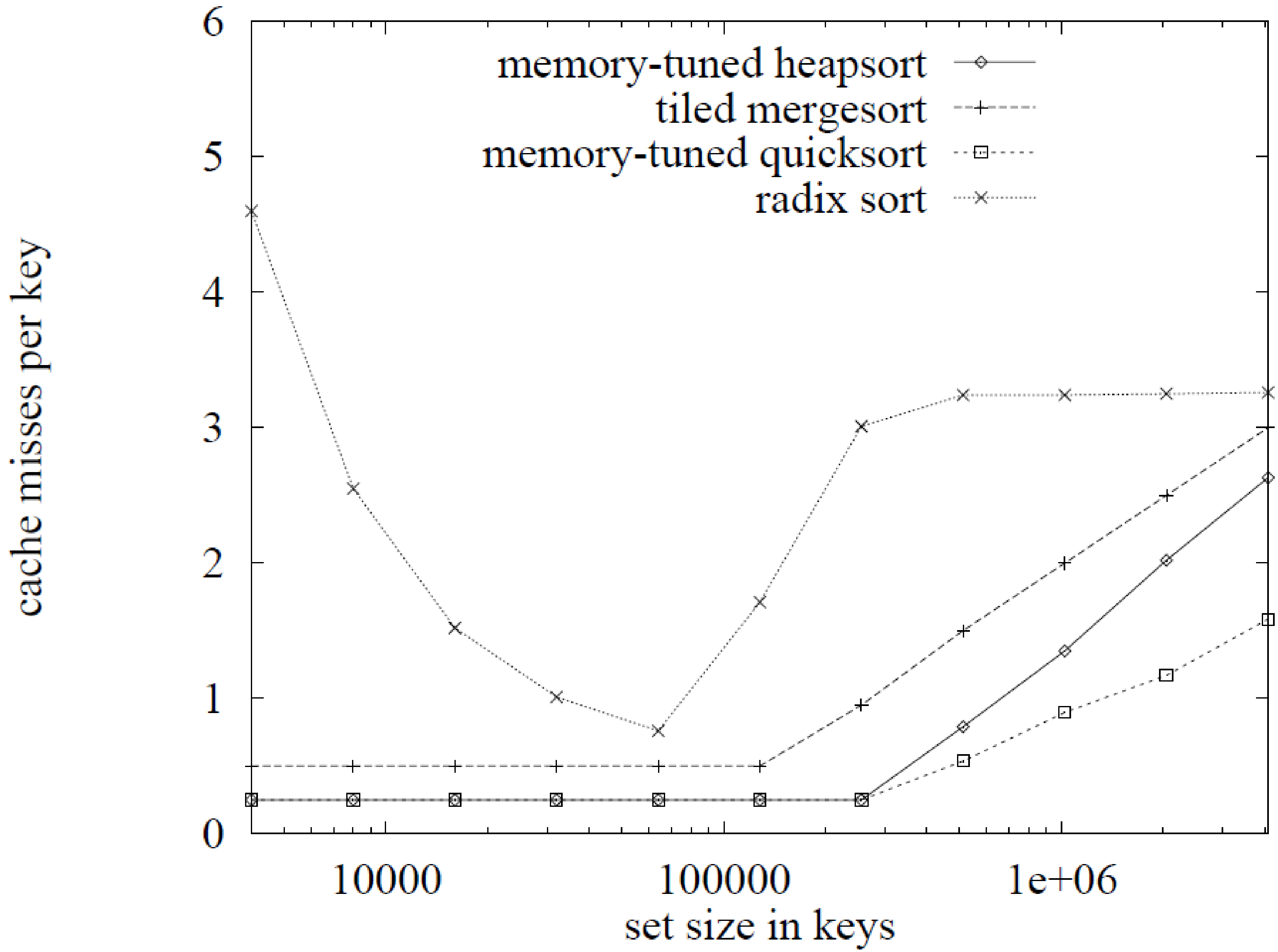
LaMarca, Ladner '97: The influence of Caches on the Performance of Sorting

Radix-Sort: Eksperimenter



LaMarca, Ladner '97

Radix-Sort: Eksperimenter

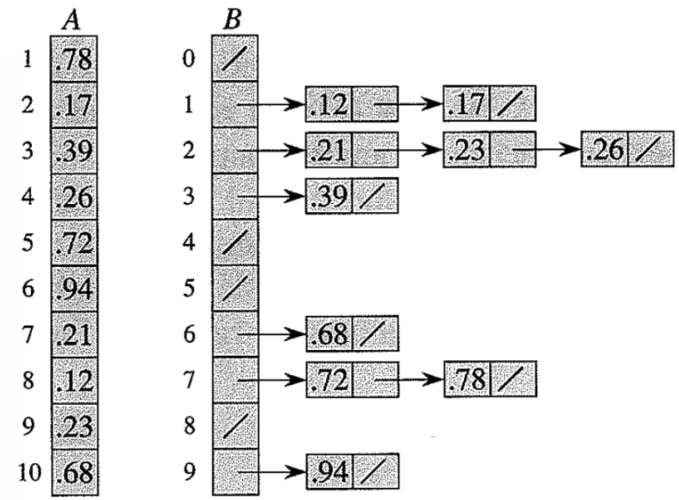


LaMarca, Ladner ’97

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

Ikke pensum

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 forventet $O(n)$ tid.

$\Omega(\log^2 n \cdot \log\log n)$

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.

Ikke pensum

Sorteringsalgoritmer i Java, Python, C++ og C

	Java (JDK SE 12.0.1)	Python (CPython 3.7)	C++ (GCC 9.1)	C (GNU C)
Metode	Java.util.Arrays.sort	sorted	std::sort	qsort
Dokumentation	https://docs.oracle.com/javase/10/docs/api/java/util/Arrays.html#sort(int[])	https://docs.python.org/3/library/functions.html#sorted	http://www.cplusplus.com/reference/algorithm/sort/	https://www.gnu.org/software/libc/manual/html_node/Array-Sort-Function.html
Algoritme	*Indbyggede typer (int, float, double...)* **Dual Pivot Quicksort** Insertion-Sort for ≤ 47 elementer Merge-Sort for input ≤ 67 sorterede "runs" Counting-Sort for stor BYTE og SHORT input *Object* **Tim Sort**	**Tim Sort** Merge-Sort variant der identificerer sorterede "runs" til at reducere antal fletninger	**Introsort** Quicksort (max dybde 2·log n) ellers skift til Heapsort Insertion-Sort ≤ 16 elements	**Quicksort** Ikke rekursiv
Kildekode	Installer JDK fra www.oracle.com/technetwork/java/javase/downloads/ Fil java.base\java\util\DualPivotQuicksort.java fra C:\ProgramFiles\Java\jdk-12.0.1\lib\src.zip	https://github.com/python/cpython/blob/master/Objects/listobject.c https://github.com/python/cpython/blob/master/Objects/listsort.txt	https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_algo.h	https://code.woboq.org/userspace/glibc/stdlib/qsort.c.html

Ikke pensum

Binary Heaps i Java, Python, C++

	Java (JDK SE 12.0.1)	Python (CPython 3.7)	C++ (GCC 9.1)
Metode	java.util.PriorityQueue	heapq	std::priority_queue
Dokumentation	https://docs.oracle.com/javase/10/docs/api/java/util/PriorityQueue.html	https://docs.python.org/3.7/library/heapq.html	http://www.cplusplus.com/reference/queue/priority_queue/
Kildekode	Installer JDK fra www.oracle.com/technetwork/java/javase/downloads/ Fil java.base\java\util\PriorityQueue.java fra C:\ProgramFiles\Java\jdk-12.0.1\lib\src.zip	https://github.com/python/cpython/blob/master/Lib/heapq.py	https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_heap.h

Alle tre implementation bruger 0-indekserede arrays hvor

left(i) = $2 \cdot i + 1$ right(i) = $2 \cdot i + 2$ parent(i) = $\lfloor (i - 1) / 2 \rfloor$