Syllabus

3. Sorting
   - Specification
   - Bubble Sort
   - Selection Sort
   - Insertion Sort
   - Merge Sort
   - Quicksort
   - Linear-Time Median
   - Sorting in Linear Time

- Specification
- Primitives: semantics and cost
- Design
- Analysis
- Optimality

of Comparison-Based Sorting

Martin Ziegler

3. Sorting

specification

Specification: Fix set X with total order $\leq$.

Input: $N \in \mathbb{N}$ and finite sequence $x_1,\ldots x_N \in X$ in array $x[1\ldots N]$

and array $\pi[1\ldots N]$:= identity permutation of $\{1,\ldots N\}$

Output: Permutation π of $\{1,\ldots N\}$ such that $x_{\pi(1)} \leq \ldots \leq x_{\pi(N)}$

Primitives:

ordered comparison “$x[n] \leq x[m]$?”	cost 1
Index integer arithmetic	cost 1
swapping $x[n] \leftrightarrow x[m]$	cost 1
swapping $\pi[n] \leftrightarrow \pi[m]$	cost 1

Preprocessing data in order to accelerate queries.

3. Sorting

Bubble Sort

Input: $N \in \mathbb{N}$ and array $x[1 \ldots N]$

Output: *Permuted* array $x \circ \pi$ s.t. $x_{\pi(1)} \leq \ldots \leq x_{\pi(N)}$

Primitives:

Comparison "$x[n] \leq x[m]$?"

Swapping $x[n] \leftrightarrow x[m]$

Index integer arithmetic

```
Procedure BubbleSort ( x[N] )
For m := N downto 2 do
    For k := 1 to m−1 do
        If x[k] > x[k+1] then
            Swap (x[k] , x[k+1])
        Endif
    Endfor
Endfor
```

6 5 3 1 8 7 2 4

runtime $O(N^2)$

Correctness: $x[1], \ldots x[m] \leq x[m+1] \leq \ldots \leq x[N]$

3. Sorting

Select Sort

Input: $N \in \mathbb{N}$ and array $x[1 \ldots N]$

Output: *Permuted* array $x \circ \pi$ s.t. $x_{\pi(1)} \leq \ldots \leq x_{\pi(N)}$

Primitives:

Comparison "$x[n] \leq x[m]$?"

Swapping $x[n] \leftrightarrow x[m]$

Index integer arithmetic

```
Procedure SelectSort ( x[N] )
For m := 1 to N−1 do
    min := m;
    For k := min+1 to N do
        If x[k] < x[min] then
            min := k ; Endif
    Swap( x[m], x[min] )
    Endfor
Endfor
```

8
5
2
6
9
3
1
4
0
7

runtime $O(N^2)$

Correctness: $x[1] \leq \ldots \leq x[m\text{-}1] \leq x[m], \ldots x[N]$

3. Sorting

Insert Sort

6 5 3 1 8 7 2 4

Input: $N \in \mathbb{N}$ and array $x[1 \ldots N]$

Output: *Permuted* array $x \circ \pi$ s.t. $x_{\pi(1)} \le \ldots \le x_{\pi(N)}$

Primitives:

Comparison "$x[n] \le x[m]$?"

Swapping $x[n] \leftrightarrow x[m]$

Index integer arithmetic

```
Procedure InsertSort ( x[N] )
For m := 2 to N do
    y := x[m];   k := m−1;
    While k>0 and x[k]>y
        x[k+1] := x[k]
        k := k −1
    Endwhile
    x[k+1] := y
Endfor
```

runtime $O(N^2)$

Correctness: $x[1] \le \ldots \le x[m\text{-}1] \le x[m], \ldots x[N]$

3. Sorting

Merge Sort

Input: $N \in \mathbb{N}$ and array $x[1 \ldots N]$

Output: *Permuted* array $x \circ \pi$ s.t. $x_{\pi(1)} \le \ldots \le x_{\pi(N)}$

```
Procedure MergeSort ( x[N] )
If N≤1 return.
l := ⌊N/2⌋ ; r := ⌈N/2⌉ ; array y[l] , z[r] ;
For m := 1 to l do y[m]:= x[m];
For m := 1 to r do z[m]:= x[l+m];
MergeSort(y);  MergeSort(z);
While l>0 and r>0 do ; If z[r] ≤ y[l]
        then  x[l+r] := y[l] ;  l:=l−1
        else  x[l+r] := z[r] ;  r:=r−1
While l>0 do ; x[l+r] := y[l] ;  l:=l−1 ; Endwhile
While r>0 do ; x[l+r] := z[r] ;  r:=r−1 ; Endwhile
```

6 5 3 1 8 7 2 4

runtime
$T(N) = 2 \cdot T(N/2) + O(N)$
$\le O(N \cdot \log N)$

memory
$O(N \cdot \log N)$

3. Sorting

Quick Sort

Procedure **QuickSort** ($x[]$; l,r)

If $l \geq r$ return. // $x[l]\ldots x[r]$ sorted

$y := x[\lfloor (l+r)/2 \rfloor] \in X$ // pivot

$a := l$; $b := r$; While $a < b$ do

While $a<b$ and $x[a] < y$ do $a := a+1$ Endwhile;

While $a<b$ and $x[b] \geq y$ do $b := b-1$ Endwhile;

Swap($x[a]$,$x[b]$);

Endwhile ; Return a;

QuickSort (x , l , $a-1$);

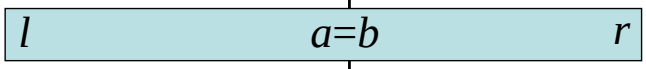
QuickSort (x , b , r);

Input: $N \in \mathbb{N}$ and array $x[1\ldots N]$

Output: *Permuted* array $x \circ \pi$ s.t. $x_{\pi(1)} \leq \ldots \leq x_{\pi(N)}$

$T(N) = T(N\cdot \varepsilon\ ) + T(N\cdot(1-\varepsilon)) + O(N)$

$\leq O(N\cdot\log N)$ for every *fixed* $\varepsilon \in (0,1)$



$\varepsilon\cdot(r-l+1) \leq a-l \leq (1-\varepsilon)\cdot(r-l+1)$

$\varepsilon\cdot(r-l+1) \leq r-a \leq (1-\varepsilon)\cdot(r-l+1)$

$c\cdot N\cdot\log(N) =: T(N) = c\cdot N\cdot\varepsilon\cdot\log(N\cdot\varepsilon) + c\cdot N\cdot(1-\varepsilon)\cdot\log(N\cdot(1-\varepsilon)) + N$

Ansatz $= c\cdot N\cdot\log(N) + N\cdot c\cdot(\varepsilon\cdot\log(\varepsilon) + (1-\varepsilon)\cdot\log(1-\varepsilon)) + N$

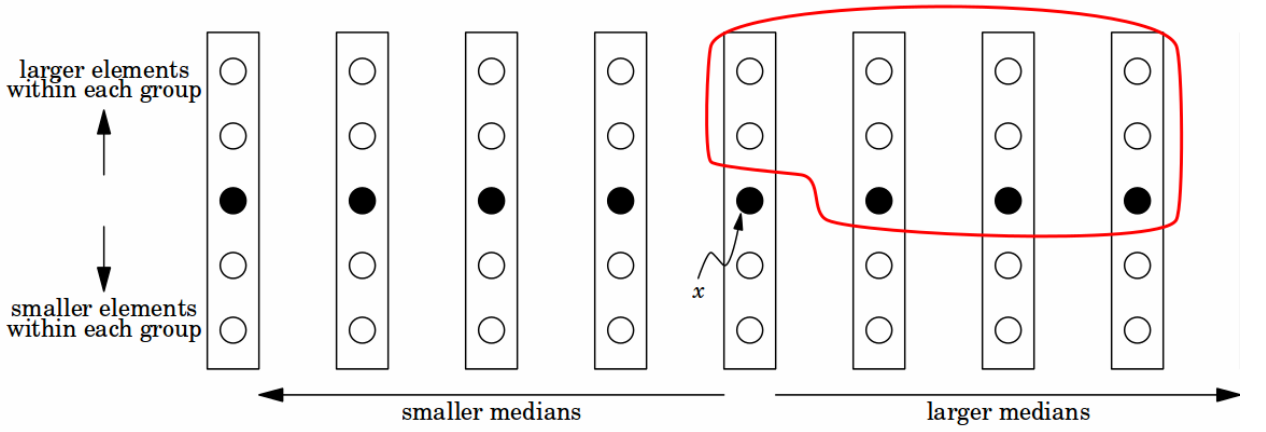
3. Sorting

Median Revisited

Function **Partition** ($x[]$; l , r ; y)
// partitions $x[l..r]$ into $x[l..m]$ entries $<y$
// and $x[m+1..r]$ those $\geq y$. Returns m.

Input: $N \in \mathbb{N}$ and array $x[1\ldots N]$

W.l.o.g. $x[n] \neq x[m]$ for all $n \neq m$



***K*-th order statist.:** m s.t. $\#\{n : l \leq n \leq r, x_n < x_m\} = K$

Lemma: *Median of 5-medians* is "$\geq$" at least ½·⅗ = 30% of entries, and "$<$" at most 70%.

Output: m such that $0.3\cdot N \leq \#\{n : x_n \leq x_m\} \leq 0.7\cdot N+1$

3. Sorting

Linear-time Median

Function **Partition** ($x[]$; l , r ; y)
// partitions $x[l..r]$ into $x[l..m]$ entries $<y$
// and $x[m+1..r]$ those $\geq y$. Returns m.

Input: $N \in \mathbb{N}$ and array $x[1\ldots N]$
W.l.o.g. $x[n] \neq x[m]$ for all $n \neq m$

Function **ApproxMed** ($x[]$; l , r)

Process $x[l...r]$ in groups of 5, sorting each one to find its median.

Then call **OrderStat** to determine the median of these 5medians.

Function **OrderStat** ($x[]$; l, r , K)
While $l<r$ do
  Call **ApproxMed** in order to determine an *approximate* median of $x[l\ldots r]$.
  Partition $x[l\ldots r]$ accordingly.
  Proceed to the left (=decrease r) /right (=increase l) accordingly.

$n := r - l + 1$

***K*-th order statist.:** m s.t. $\#\{n : l \leq n \leq r, x_n < x_m\} = K$

Lemma: *Median of* 5*-medians* is "$\geq$" at least ½·⅗ = 30% of entries, and "$<$" at most 70%.

$$T_A(n) = O(n) + T_O(0.2 \cdot n), \quad T_O(n) = T_A(n) + O(n) + T_O(0.7 \cdot n)$$

3. Sorting

Optimality

Specification: Fix set X with total order $\leq$.
Input: $N \in \mathbb{N}$ and array $x[1\ldots N]$ with values in X
Output: *Permutation* π s.t. $x_{\pi(1)} \leq \ldots \leq x_{\pi(N)}$

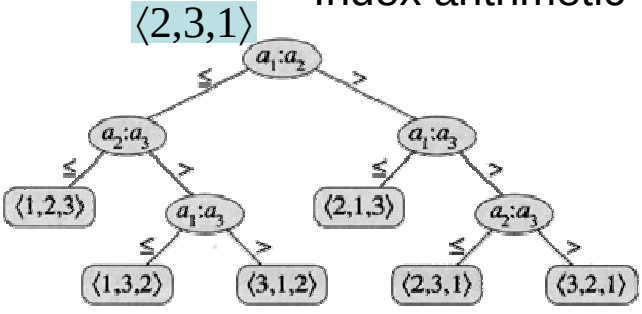
Primitives

"$x[\pi[n]] \leq x[\pi[m]]$?"

Swapping $\pi[n] \leftrightarrow \pi[m]$

Index arithmetic

Definition: A *Decision Tree for sorting N elements* is a binary tree whose nodes are labeled "$x[i] \leq x[j]$?", $1 \leq i < j \leq N$.
and whose leaves are labeled with permutations π such that every input $x = x[1\ldots N]$ ends up in a leaf whose label π satisfies $x[\pi[1]] \leq \ldots \leq x[\pi[N]]$.



3. Sorting

Optimality

Lemma: a) For fixed N, every sorting algorithm of worst-case runtime $T(N)$ can be "unrolled" into a decision tree for sorting N elements of depth $\le T(N)$.

b) Every permutation $x=\pi$ ends up in the unique leaf with label π^{-1}.

c) A binary tree with $N!$ leaves has depth $\ge \log(N!)=\Theta(N\cdot\log N)$

Definition: A *Decision Tree for sorting N elements* is a binary tree

whose internal nodes are labeled "$x[i]\le x[j]$?", $1\le i<j\le N$.

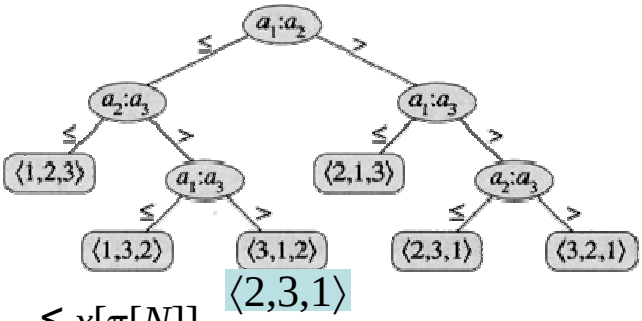
and whose leaves are labeled with permutations π such that

every input $x=x[1\ldots N]$ ends up in a leaf whose label π satisfies $x[\pi[1]] \le \ldots \le x[\pi[N]]$.

"$x[\pi[n]]\le x[\pi[m]]$?"

Swapping $\pi[n] \leftrightarrow \pi[m]$

Index arithmetic



3. Sorting

Counting Sort

Specification: Fix set $X=\{1,\ldots M\}$!

Input: $N\in\mathbb{N}$ and array $x[1\ldots N]$ with key values in X

Output: *Permuted* array $y=x\circ\pi$ s.t. $x.key[\pi[1]]\le\ldots\le x.key[\pi[N]]$

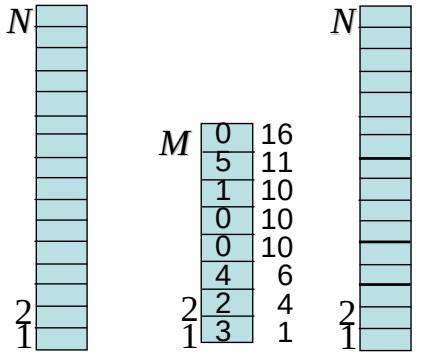
```
integer array count[1…M];
For m:=1 to M do  count[m] := 0;
For n:=1 to N do  count[x.key[n]]++;
sum := 1;  For  m:=1  to  M  do
  temp := count[m];  count[m]:= sum;
  sum += temp;  Endfor;
For n:=1 to N do
  y[count[x[n].key]++] := x[n];
```

runtime $O(N+M)$

"$x[\pi[n]]\le x[\pi[m]]$?"

~~Swapping $\pi[n] \leftrightarrow \pi[m]$~~

Index arithmetic



3. Sorting

Radix Sort

Specification: Fix set $X=\{\mathbf{0},\mathbf{1}\}^m$ with lexicographical order.

Input: $N\in\mathbb{N}$ and array $x[1\ldots N]$ with values in X

$0\ b_{m-1}\ldots b_2\ b_1$
$<\ 1\ c_{m-1}\ldots c_2\ c_1$

Output: *Permuted* array $y=x\circ\pi$ s.t. $x[\pi[1]]\le\ldots\le x[\pi[N]]$

Quick-/Mergesort in Bit-model: $O(N\cdot\log N\cdot m)$

~~"$x[\pi[n]]\le x[\pi[m]]$?"~~

Swapping $\pi[n]\leftrightarrow\pi[m]$

Index arithmetic

RadixSort($x[]$, l, r, m);
// Sort $x[l\ldots r]$ w.r.t. bits $\#m\ldots\#1$
If $l=r$ or $m<1$ then return;
// Put all entries with **0** as bit $\#m$ before those with **1**:
mid := *Partition* ($x[]$, l , r , m);
RadixSort($x[]$, l , mid , m-1);
RadixSort($x[]$, mid+1 , r , m-1);

$O(N\cdot m)$ operations

Bit-model: $O(N\cdot m^2)$