20 November, 2015 ARCO Meeting, Lyngby

Adjustable navigation pile

Jyrki Katajainen[1]

Tetsuo Asano[2] Omar Darwish[3]

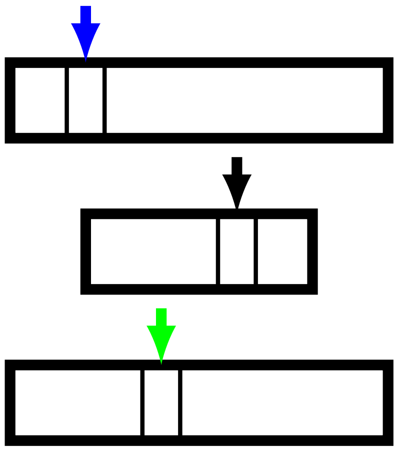
Amr Elmasry[4] Fabio Vitale[5]

[1] University of Copenhagen
[2] Japan Advanced Institute for Science and Technology
[3] Max-Planck Institute for Informatics
[4] Alexandria University
[5] University of Lille

These slides are available via my research information system
`http://www.diku.dk/~jyrki/Myris/slides-by-year.html`

Restricted SAM

SAM: **sequential-access machine**



input tape; **read only**

work tape; **read write**

output tape; **write only**

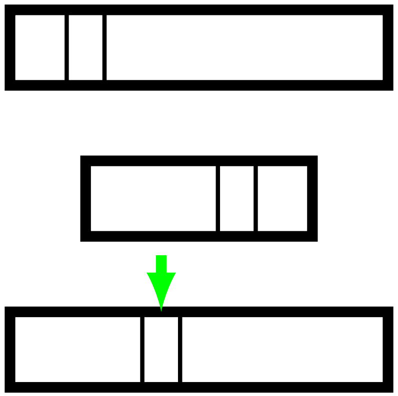
one-way read head

two-way read/write head

one-way write head

Restricted RAM

RAM: **random-access machine**



input; **read only**; **random access**

work space; **read write**; **random access**

output; **write only**; sequential access

one-way printing

Schönhage et al.'s order notation

N: **problem size**

a number	sets of functions
	$O(\lg N)$
$\mathcal{O} \lg N$	
	$\Omega(\lg N)$

$\mathcal{O}$ **read:** bounded

$\mathcal{O}$ **read:** an unspecified constant

silly question: What is $5N^2 + 20N$? A number or a function?

Space-time trade-offs

input size: N (measured in elements)

working space: $S(N)$ **bits**; $S(N) \geq \lg N$

running time: $T(N)$

expected: $S(N) \nearrow \quad \Longrightarrow T(N) \searrow$

typical question: What is the fastest algorithm for the problem $\mathcal{P}$ of size N when working space of $S(N)$ bits is available?

Sorting

input size: N (measured in elements)

working space: $S(N)$ **bits**; $S(N) \geq \lg N$

running time: $T(N)$

comparison-based lower bound: $T(N) \geq \emptyset N \lg N$

[Beame 1991] $S(N) \cdot T(N) \geq \emptyset N^2$

[Pagter & Rauhe 1998] $T(N) \leq \emptyset N^2 / S(N) + \emptyset N \lg(S(N))$, for any $S(N) \geq \emptyset \lg N$

Questions

(1) How would you sort N elements in $\mathcal{O}N^2/\lg N$ worst-case time when only working space of $\mathcal{O}\lg N$ bits is available?

(2) How would you sort N elements in $\mathcal{O}N\lg N$ worst-case time when working space of $\mathcal{O}N/\lg N$ bits is available?

optimally adjustable: How to achieve optimal worst-case running time $\mathcal{O}N^2/S(N)$ for every $S(N) \in [\mathcal{O}\lg N \mathinner{..} \mathcal{O}N/\lg N]$?

Our answer

procedure: $\mathit{pilesort}$
input: $A[0\,..\,N-1]$: read-only array of N elements
S: workspace size

$P \leftarrow \mathit{navigation\text{-}pile}(A, S)$

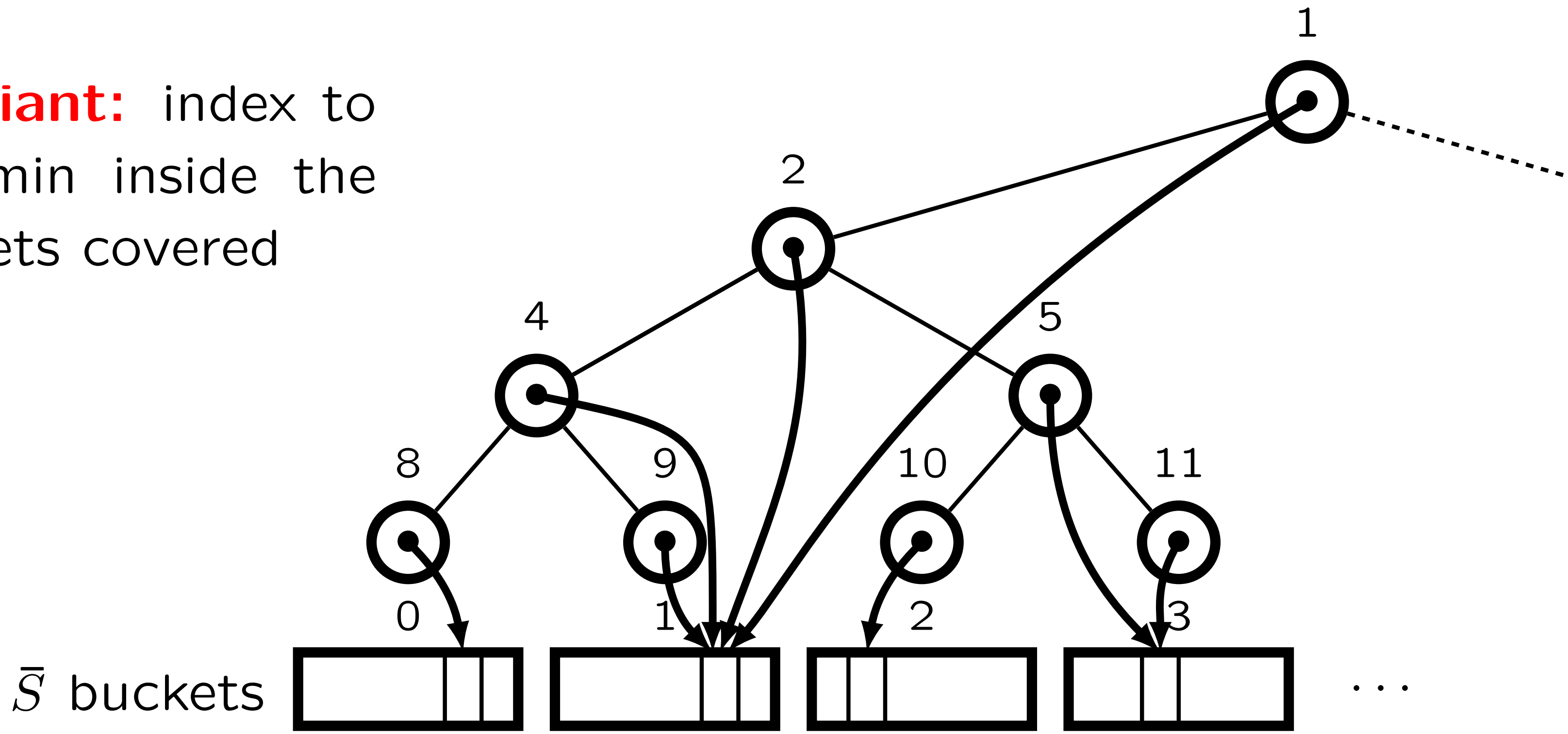
for $i \in \{0, 1, \ldots, N-1\}$:
  $P.\mathit{insert}(i)$

while $|P| > 0$:
  $j \leftarrow P.\mathit{minimum}()$
  $P.\mathit{extract}(j)$
  $\mathit{print}(A[j])$

Tournament tree

$\bar{S} := 2^{\lceil \lg S \rceil}$

invariant: index to the min inside the buckets covered



$\lceil N/\bar{S} \rceil$ elements per bucket

 # bits: about $2\bar{S} \cdot \lg N$; a log factor too much

procedure: *path-update*

input: *bucket-start*: index of the beginning of the bucket changed

bucket-min: index of the minimum **alive** element within this bucket

data: N: number of elements

$\bar{S}$: workspace size rounded to a power of 2

$A[0\,..\,N-1]$: read-only array of elements

$T[1\,..\,2\bar{S}-1]$: array of indices from $\{\mathbf{none}, 0, 1, \ldots, N-1\}$

$current \leftarrow \bar{S} + bucket\text{-}start / \lceil N/\bar{S} \rceil$

$T[current] \leftarrow bucket\text{-}min$

for $\ell \in \{\lg \bar{S}, \lg \bar{S} - 1, \ldots, 1\}$:

  $parent \leftarrow \lfloor current/2 \rfloor$

  $sibling \leftarrow$ **if** $current \bmod 2 = 0$: $this + 1$ **else** $this - 1$

  if $T[sibling] = \mathbf{none}$:

    $T[parent] \leftarrow T[current]$

  else if $T[current] = \mathbf{none}$:

    $T[parent] \leftarrow T[sibling]$

  else if $A[T[current]] < A[T[sibling]]$:

    $T[parent] \leftarrow T[current]$

  else:

    $T[parent] \leftarrow T[sibling]$

  $current \leftarrow parent$

Reestablishing the invariants after a bucket gets a new min

Operations

insert:

- add the new element to the last bucket
- update the min of that bucket, if necessary
- run *path-update* for this bucket

worst-case running time: $\mathcal{O}\lg \bar{S}$

insert **improved:**

- divide the work of *path-update* for $\lceil N/\bar{S}\rceil$ insertions

extract:

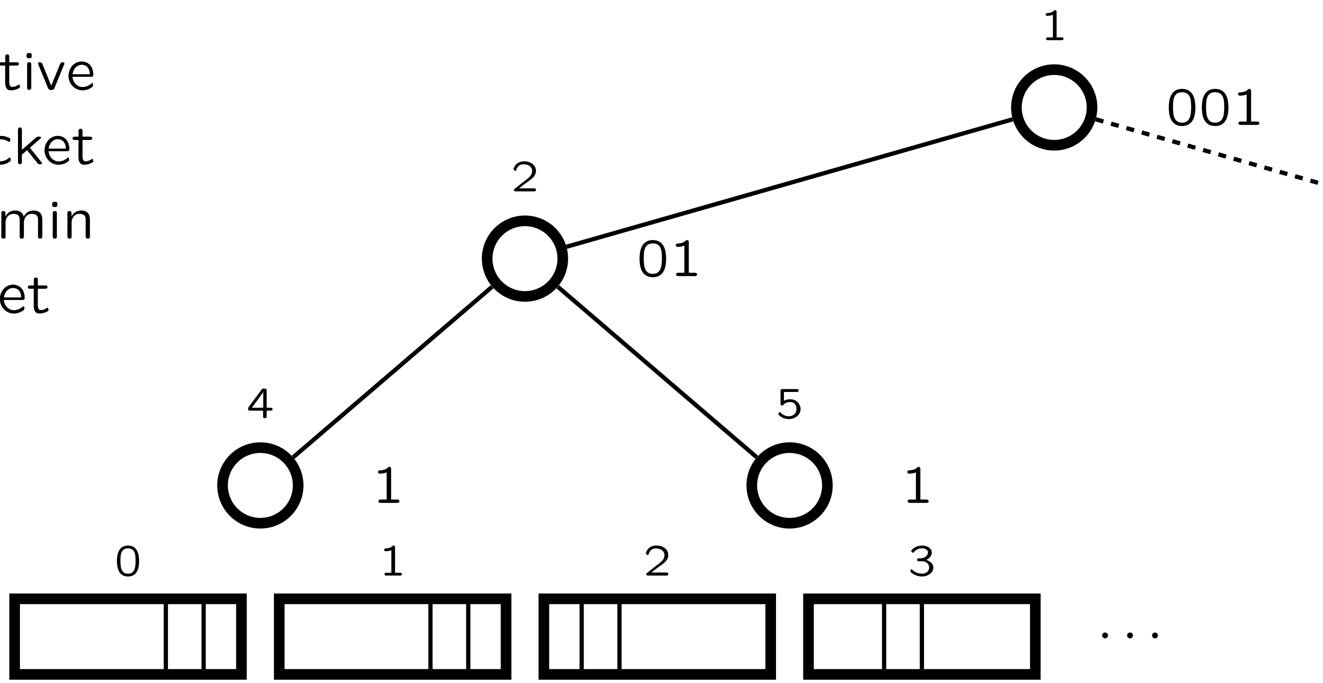
- find the new min of the bucket where a change was made
- run *path-update* for this bucket

worst-case running time: $\mathcal{O}\, N/\bar{S} + \mathcal{O}\lg \bar{S}$

Navigation pile

invariant: relative index of the bucket containing the min in the covered set

$\bar{S}$ buckets



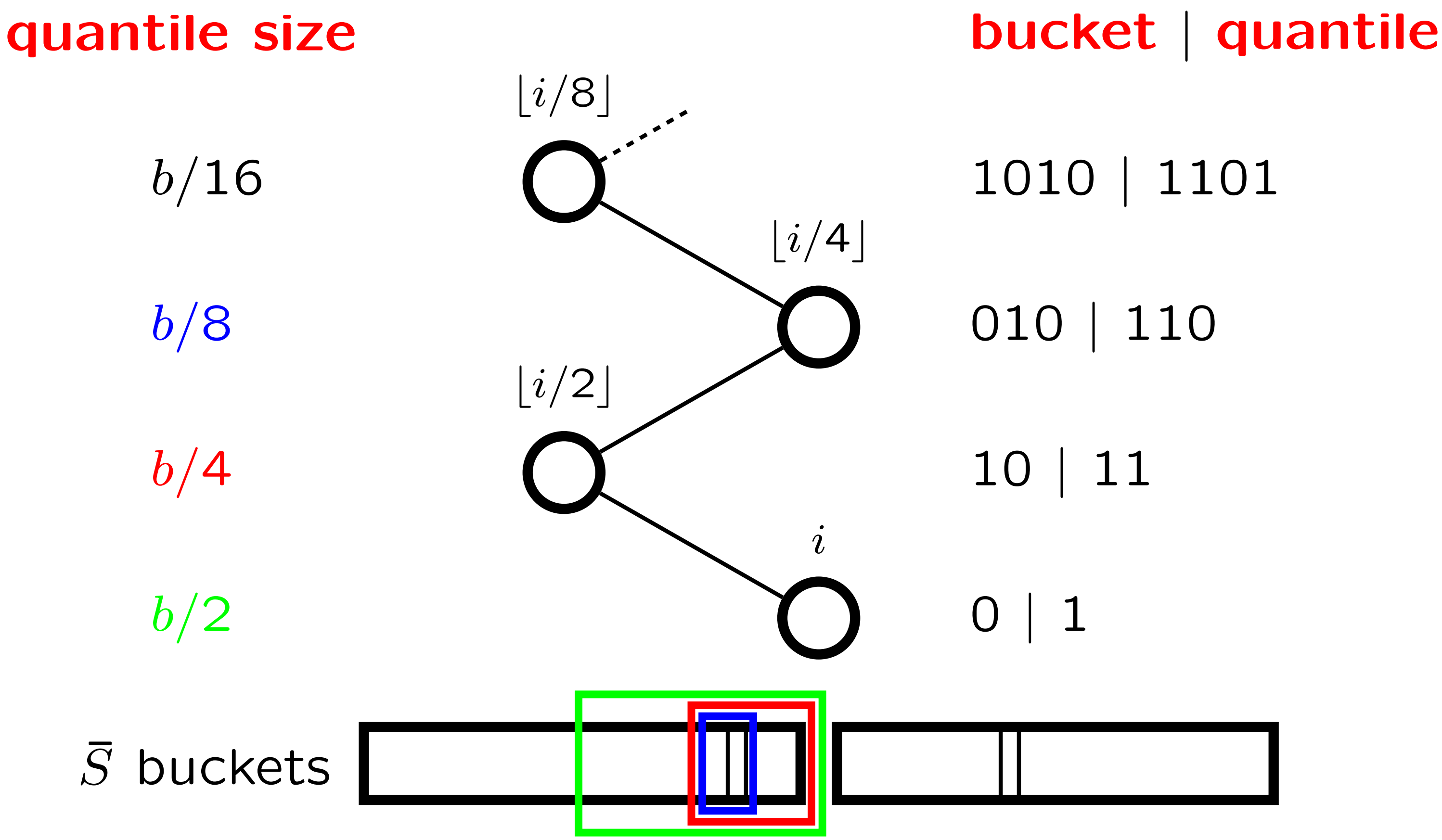
$\lceil N/\bar{S} \rceil$ elements per bucket

Bits stored in breadth-first order in a bit vector

 # bits: $\frac{\bar{S}}{2} \cdot 1 + \frac{\bar{S}}{4} \cdot 2 + \frac{\bar{S}}{8} \cdot 3 + \ldots \leq 2\bar{S}$

update: $\mathcal{O}(\lg \bar{S} \cdot N/\bar{S})$; a log factor too slow

Quantile thinning



$b := \lceil N/\bar{S} \rceil$ elements per bucket

 # bits: $\leq 4\bar{S}$

Running time per *update*

new bucket min: $\mathcal{O} N/\bar{S}$

path-update: $(\frac{1}{2}N/\bar{S}+\mathcal{O})+(\frac{1}{4}N/\bar{S}+\mathcal{O})+(\frac{1}{8}N/\bar{S}+\mathcal{O})+\ldots \leq \mathcal{O} N/\bar{S}+\mathcal{O} \lg \bar{S}$

$$\therefore$$

insert: $\mathcal{O} N/\bar{S} + \mathcal{O} \lg \bar{S}$; can be improved to $\mathcal{O}$, if wanted

extract: $\mathcal{O} N/\bar{S} + \mathcal{O} \lg \bar{S}$

minimum: can be supported in $\mathcal{O}$ worst-case time by maintaining a cursor to the overall min

Remarks

slow!

fun!

application: In a word RAM, an adjustable navigation pile that uses $\mathcal{O} N/\lg N$ bits of extra space supports *insert* in $\mathcal{O}$ worst-case time and *extract-min* in $\mathcal{O} \lg N$ worst-case time involving at most $\mathcal{O}$ **element moves**.

open: For almost any problem, the exact space-time trade-off is not known in the restricted RAM model.

Further reading

[Beame 1991] sorting lower bound

[Pagter & Rauhe 1998] sorting upper bound

[Schönhage et al. 1994] order notation

[Knuth 2011] Vol. 4A, bit manipulation, mentions navigation piles

[Katajainen & Vitale 2003] original N-bit version

[Asano et al. 2013] conference version

[Darwish et al. 2015] journal version `http://arxiv.org/abs/1510.07185`

[Elmasry et al. 2014] selection

[Elmasry et al. 2015] graph algorithms

[Elmasry & Kammer 2015] geometric algorithms