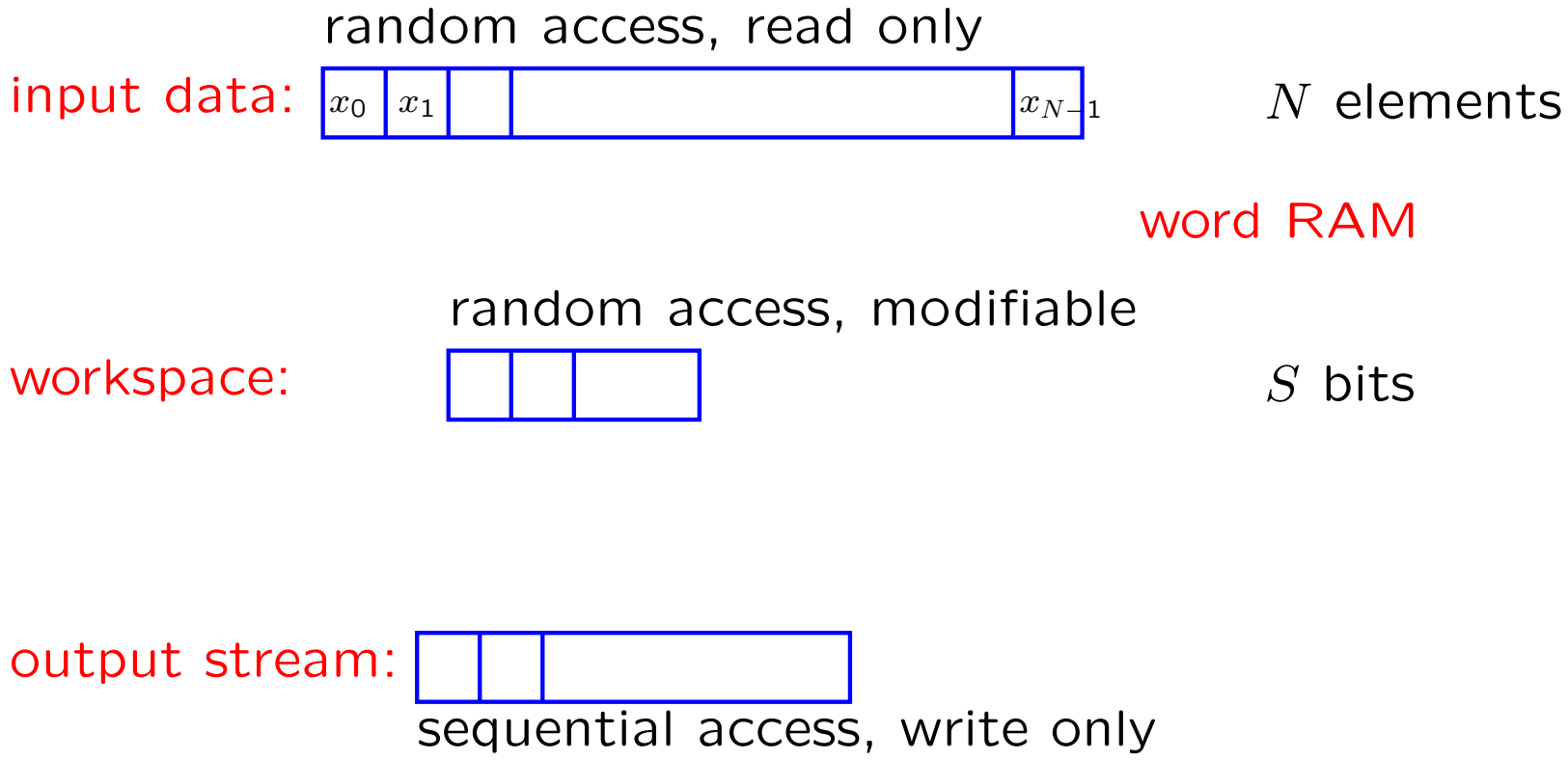
Priority Queues and Sorting for Read-Only Data

Tetsuo Asano[1], **Amr Elmasry**[2], **and**

Jyrki Katajainen[3,4]

[1] Japan Advanced Institute for Science and Technology
[2] Alexandria University
[3] University of Copenhagen
[4] Jyrki Katajainen and Company

Model of computation



Related model: space-bounded Turing machine

Motivation: special devices where working space is limited (mobile devices) and where writing is expensive (flash memories)

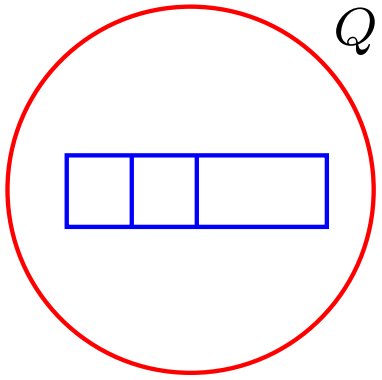
Online exercise: Selection sort

1. How would you modify selection sort so that it is suitable for the space-bounded random-access machine?
2. What is the space-time trade-off of this algorithm?
3. Can you improve the space-time trade-off?

Problem: A priority queue for read-only data

input data: 

N elements

workspace: 

$O(S + w)$ bits

(w word size, $\lg N \leq S \leq N/\lg N$)

Operations for a priority queue Q

$Q.minimum()$: Return the position of the minimum element in Q.

$Q.insert(p)$: Insert the element at position p of the read-only array into Q.

$Q.extract(p)$: Extract the element at position p of the read-only array from Q.

Market analysis

Data structure	Space	*minimum*	*insert*	*extract*
queue of pennants	$\Theta(N \lg N)$	$O(1)$	$O(1)$	$O(\lg N)$
navigation pile	$\Theta(N)$	$O(1)$	$O(\lg N)$	$O(\lg N)$
adjustable binary heap	$\Theta(S)$	$O(1)$	$O(N \lg N/S + \lg S)$	$O(N \lg N/S + \lg S)$
common precursor	$\Theta(S)$	$O(N/S^2 + \lg S)$	$O(N/S + \lg S)$ amort.	$O(N/S + \lg^2 S)$
adjustable navigation pile	$\Theta(S)$	$O(1)$	$O(1)$	$O(N/S + \lg S)$

Beame 1991: The space-time product of any sorting algorithm is $\Omega(N^2)$.

Pagter & Rauhe 1998: An optimal sorting algorithm is obtained by combining an adjustable binary heap and their adjustable priority queue.

this paper: Simplification of the latter result; heapsort with an adjustable navigation pile gives the optimal sorting bound.

Application: Priority-queue sort

procedure: *priority-queue-sort*
input: A: read-only array of N elements; S: space target
output: stream of elements produced by the *print* statements

```
1 P ← navigation-pile(A, S)
2 for x ∈ {0, 1, ..., N − 1}
3 |   P.insert(x)
4 repeat N times
5 |   y ← P.minimum()
6 |   P.extract(y)
7     print(A[y])
```

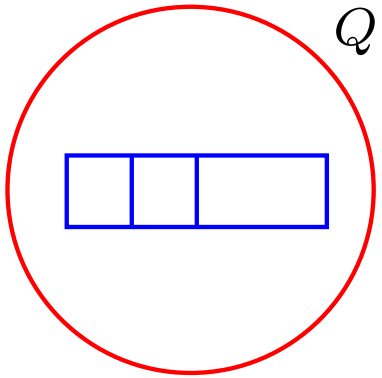
Workspace: $O(S + w)$ bits (w word size in bits)
Worst-case running time: $O(N^2/S + N \lg S)$ $(\lg N \leq S \leq N/\lg N)$

Assumptions

1. N is known beforehand.
2. The elements are extracted from the data structure in monotonic fashion.
3. The elements are inserted into the data structure sequentially in streaming-like fashion starting from the first element stored in the read-only input.

input data: 

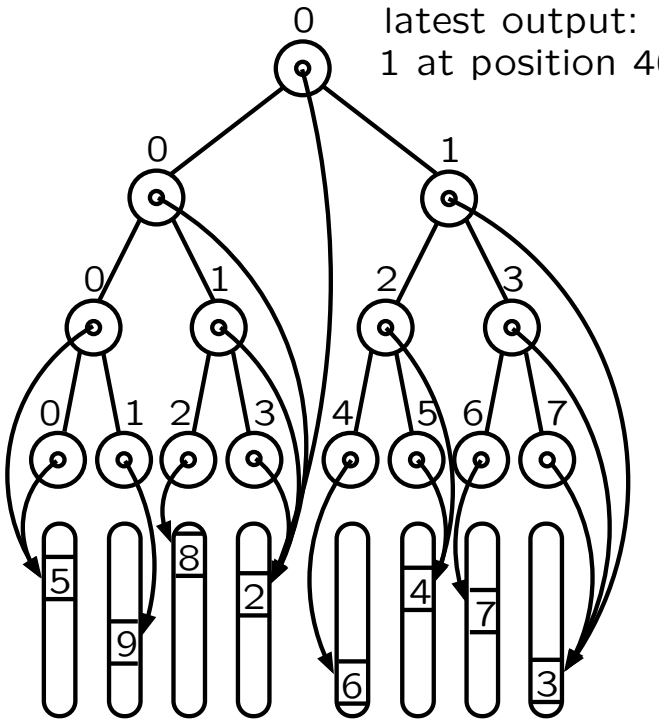
N elements

workspace: 

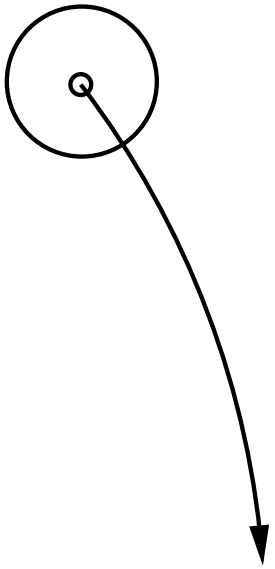
$O(S + w)$ bits

(w word size, $\lg N \leq S \leq N/\lg N$)

Memory-adjustable tournament tree



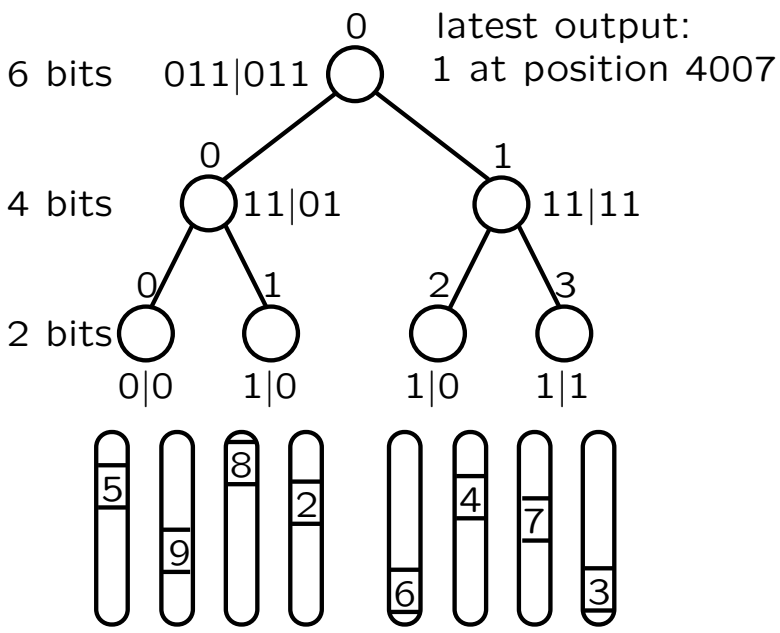
$S = 8$; $\lceil N/S \rceil$ elements per bucket



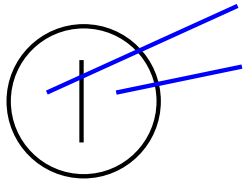
– pointer to the minimum in the covered range

– nodes stored in breadth-first order

Memory-adjustable navigation pile



$S = 8$; $\lceil N/S \rceil$ elements per bucket



– height h
– which bucket h bits
– which quantile h bits

– nodes stored in breadth-first order

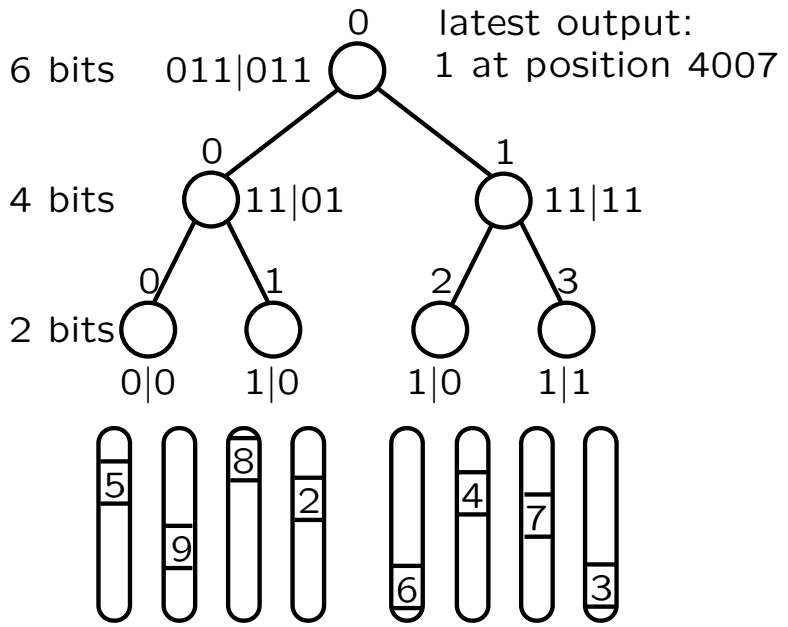
– $\displaystyle\sum_{h=1}^{\lg \bar{S}} \frac{\bar{S} \cdot \min\{2h, \lceil \lg N \rceil\}}{2^h} < 4\bar{S}$ bits

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

minimum

- The last bucket in use is an **insertion buffer**.
- The second last bucket is a **submersion buffer** (if any).
- Maintain pointers to the minima of the buffers and the pile, and information where the overall minimum is.

Worst-case running time: $O(1)$

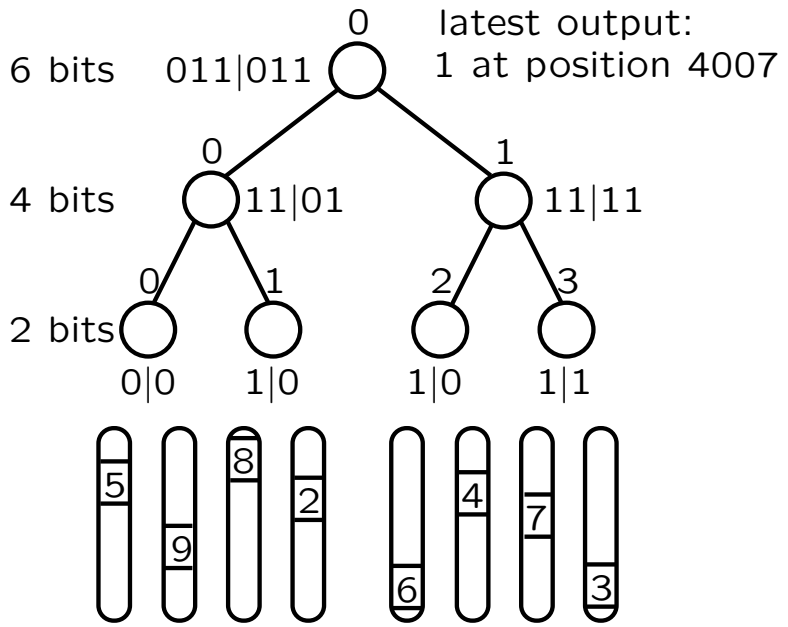


$S = 8$; $\lceil N/S \rceil$ elements per bucket

insert

- Insert into the insertion buffer; perform part of the incremental submersion; update the minimum pointers if necessary.
- If the insertion buffer is full, make this buffer a submersion buffer.

Worst-case running time: $O(1)$



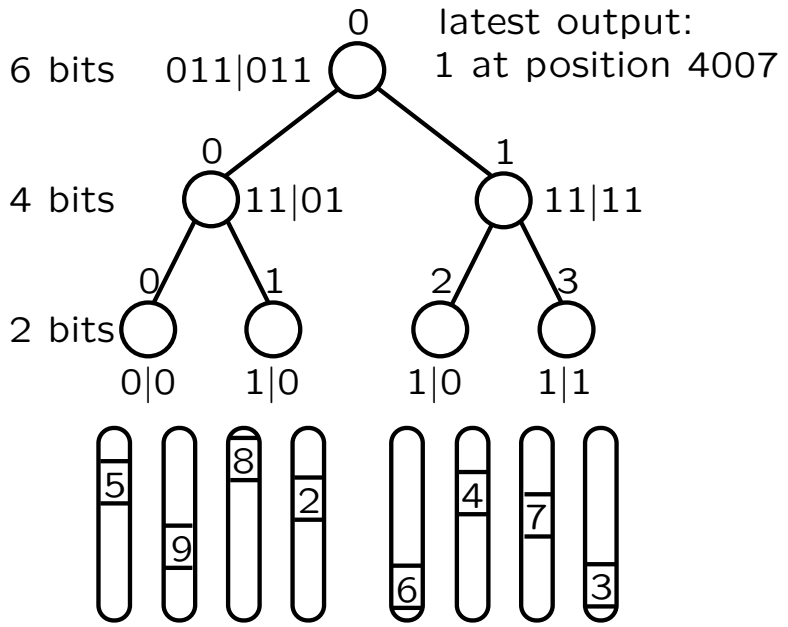
$S = 8$; $\lceil N/S \rceil$ elements per bucket

extract

- Locate the correct bucket.
- Insertion buffer: Update the minimum pointers.
- Submersion buffer: Redo the whole submersion and update the minimum pointers.
- Pile: Find the minimum of the bucket; update the minimum pointers.
- Update the navigation information on the path above.
- Scan the elements in the quantiles of the siblings of the nodes along the path.

- The work done when scanning the quantiles of the siblings is proportional to $\sum_{h=1}^{\lg \bar{S}} \left\lceil N/(\bar{S} \cdot 2^h) \right\rceil \approx N/\bar{S}$.

Worst-case running time: $O(N/S + \lg S)$



$S = 8$; $\lceil N/S \rceil$ elements per bucket

Open: Find the kth smallest of N elements

Inventors	Workspace in bits	Running time
Munro and Raman 1996	$\Theta(\lg N)$	$O(N^{1+\varepsilon})$
Raman and Ramnath 1999	$\Theta(\lg^2 N)$	$O(N \lg^2 N)$
Frederickson 1987	$\Theta(\lg^3 N)$	$O(N \lg N / \lg \lg N)$
Blum et al. 1973	$\Theta(N \lg N)$	$\Theta(N)$
COCOON 2013	$O(N)$	$\Theta(N)$

- What is the correct worst-case space-time trade-off?
- In the randomized case, the bound $\Theta(N \lg \log_S N)$ is tight for all $S \gg \lg N$.

Open: Compute the convex hull of N points

- Is the space-time trade-off the same for this problem as for sorting?

Summary: The key techniques used

1. Node numbering with implicit links between nodes (as in a binary heap),
2. buffering,
3. incremental construction,
4. bit packing and unpacking, and
5. quantile thinning.

Further direction

Siu Wing Cheng: Does the model and the algorithm extend naturally to the external-memory case? I.e. calculate the number of I/Os and keep the input on a read-only media.

My answer: The model extends naturally and may even be more relevant than the model considered by us. However, the data structure does not extend optimally since the quantiles are scattered over the read-only input.