

Presented at the *8th Scandinavian Workshop on Algorithm Theory* held on 3–5 July 2002 in Turku, Finland.

Title:

A randomized in-place algorithm for positioning the k th element in a multiset

Authors:

Jyrki Katajainen and Tomi Pasanen

Speaker:

Jyrki Katajainen

These slides are available at

<http://www.cphstl.dk>.

This bunch also contains slides that I did not have time to show.

Algorithm senility

Strictly in-place algorithms:

In addition to the input sequence, use only $O(1)$ extra words of memory.

Element moves:

Elements in the input sequence must be moved by swapping elements wordwise.

Practical relevance:

≈ 0

Theoretical motivation:

What can be done efficiently when only $O(1)$ memory cells are available?

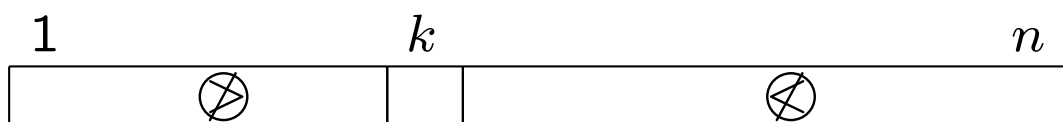
Positioning

Input:

A sequence A of n elements, an integer $k \in [1:\lceil n/2 \rceil]$, and an ordering function \ominus returning true or false.

Task:

Rearrange the elements of A such that $A[k] \ominus A[j]$ is false for all $j \in [1:k-1]$, and $A[\ell] \ominus A[k]$ is false for all $\ell \in [k+1:n]$.



Goodies:

1. Do positioning, not only selection.
2. Operate (strictly) in-place.
3. Handle multiset data.
4. Rely only on boolean ordering functions (binary element comparisons).

STL interface

```
template <
    typename random_access_iterator
>
void nth_element (
    random_access_iterator first,
    random_access_iterator nth,
    random_access_iterator one_past_the_end
);

template <
    typename random_access_iterator,
    typename ordering
>
void nth_element (
    random_access_iterator first,
    random_access_iterator nth,
    random_access_iterator one_past_the_end,
    ordering less
);

template <
    typename element
>
struct less: binary_function<element, element, bool> {
    bool operator() (
        const element& x,
        const element& y
    ) const {
        return x < y;
    }
};
```

Known in-place results

Reference	Runtime	#Comps	#Swaps	Comments
[Hoare 1961, Kirschenhoffer et al. 1997]	exp. $O(n)$	$2.75n + o(n)$	$0.46n + o(n)$	median of 3 bounds for $k = \lceil n/2 \rceil$
[Floyd & Rivest 1975]	exp. $O(n)$	$n + k + o(n)$	$k + o(n)$	for sets
[Lai & Wood 1988]	$O(n)$	$6.9n + o(n)$	$> 9n$	3-way comps
[Cunto & Munro 1989]	$\Omega(n)$	$n + k - O(1)$	k	all Las-Vegas algorithms
[Carlsson & Sundström 1995]	$O(n)$	$(2.95 + \varepsilon)n$	$O(n)$	median finding 3-way comps moves in registers gratis
[Carlsson & Sundström 1995]	$O(n)$	$3.75n + o(n)$	$> 4.5n + o(n)$	selection 3-way comps moves in registers gratis
[Geffert 2000]	$O(n)$	$O(n \log_2(1/\varepsilon))$	εn	selection 3-way comps

Our results

A Las-Vegas algorithm:

Runtime	#Comps	#Swaps	Comments
$O(n)$	$n+k+o(n)$	$k+o(n)$	if both \otimes and \ominus are given
$O(n)$	$2n+o(n)$	$k+o(n)$	if only \otimes is given

The probability that these resource bounds are exceeded is at most $e^{-n^{\Omega(1)}}$.

A deterministic algorithm:

Runtime	#Comps	#Swaps	Comments
$O(n)$	$3.64n + 0.72k + o(n)$	$O(n)$	based on the algorithm of Schönhage et al. [1976]

Not in the proceedings.

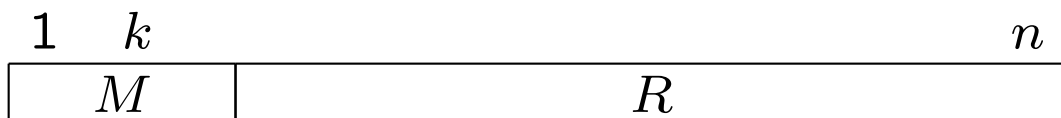
Randomized algorithm using $o(n)$ extra space

Position(A, k, \ominus)

- 1 $n \leftarrow |A|$; $s \leftarrow n^\beta \quad \triangleright 0 < \beta < 1$
- 2 **if** $n < \text{some constant}$ **or** space available $< s$:
- 3 Sort(A, \ominus); **return**
- 4 Pick a random sample S of size s from A ;
tag each element with its index
- 5 Sort($S, \text{lex-}\ominus$)
- 6 **if** $k < n^\gamma$: $\triangleright 1 - \beta < \gamma < 1$
- 7 $\mu \leftarrow n^\gamma s / n$; $y \leftarrow S[2\mu]$
- 8 $M, R \leftarrow 2\text{-Partition}(A, y, \text{lex-}\ominus)$
- 9 **if** $|M| < n^\gamma$: Sort(A, \ominus)
- 10 **else**: Sort(M, \ominus) \triangleright normal mode
- 11 **else**:
- 12 $\mu \leftarrow ks / n$; $\Delta \leftarrow n^\alpha \mu^{1/2} \quad \triangleright 0 < \alpha < \beta$
- 13 $\lambda \leftarrow \lfloor \mu - \Delta \rfloor$; $\nu \leftarrow \lceil \mu + \Delta \rceil$
- 14 $x \leftarrow S[\lambda]$; $y \leftarrow S[\nu]$
- 15 $L, M, R \leftarrow 3\text{-Partition}(A, x, y, \text{lex-}\ominus)$
- 16 **if** $|L| \geq k$ **or** $|R| \geq n - k$: Sort(A, \ominus)
- 17 **else**: Sort(M, \ominus) \triangleright normal mode

Analysis: normal mode

6 **if** $k < n^\gamma$:



11 **else**:

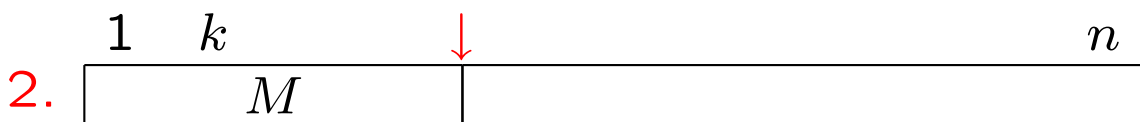
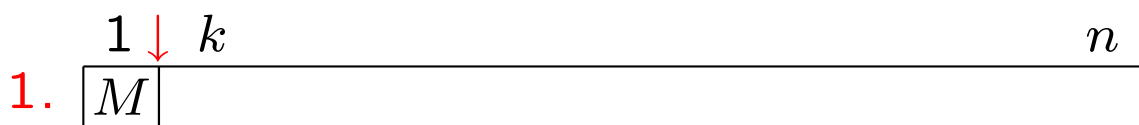


- In this mode, the k th element falls in M and M is small.
- Since $s = n^\beta$, $0 < \beta < 1$, the manipulation of the sample takes $o(n)$ time.
- If $|M| < o(n/\log_2 n)$, the sorting of M takes $o(n)$ time.
- By carefully implementing 2-Partition and 3-Partition, the claimed bounds follow.

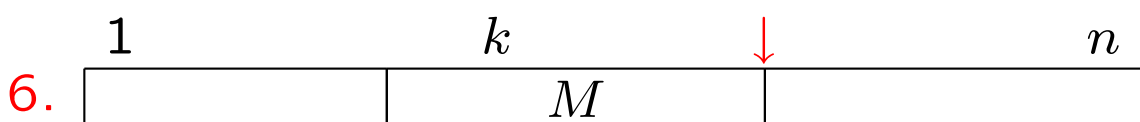
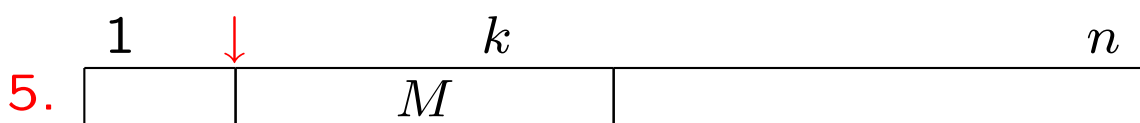
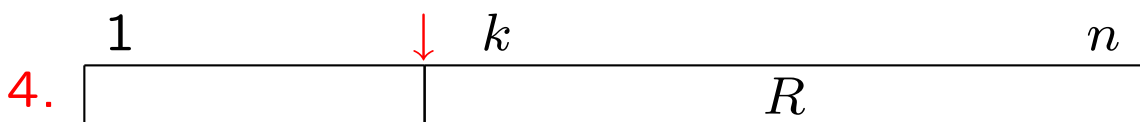
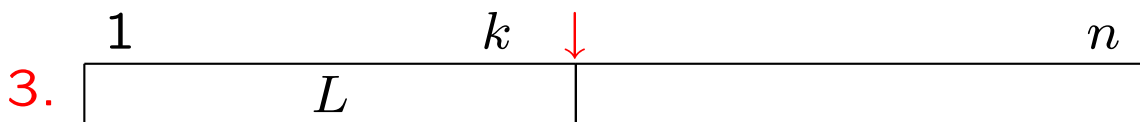
Failure modes

The algorithm may fail in six ways:

$$k < n^\gamma$$

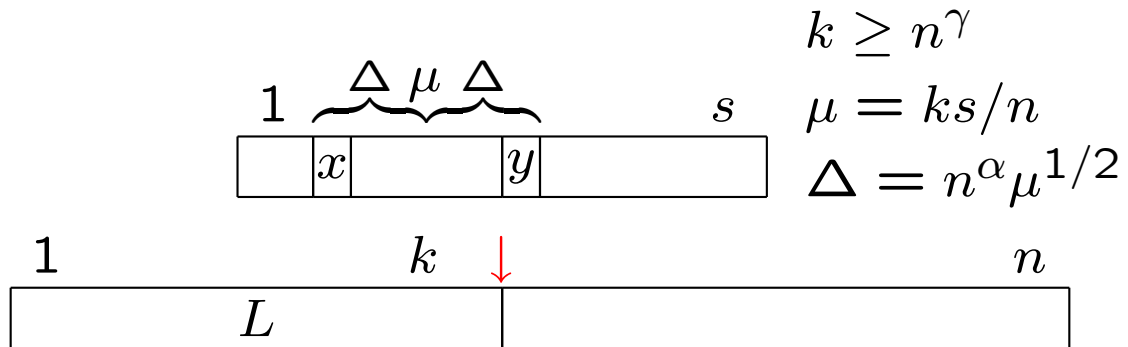


$$k \geq n^\gamma$$



The probabilities of these failures can be bounded above by Chernoff bounds.

Analysis: failure mode 3



Define $X_i = 1$, if the i th sample element is lexicographically smaller than the k th element, and $X_i = 0$ otherwise.

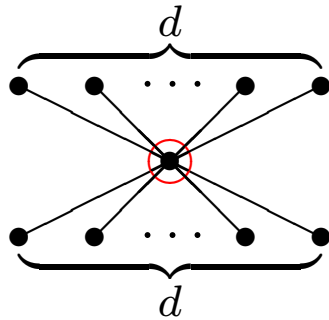
For $X = \sum_{i=1}^s X_i$, $\mathbf{E}[X] = \mu = ks/n$. In the case of failure, $X < \mu - \Delta$.

We bound the lower tail probability of X using the simplified Chernoff bound [Motwani & Raghavan 1995, Theorem 4.3]:

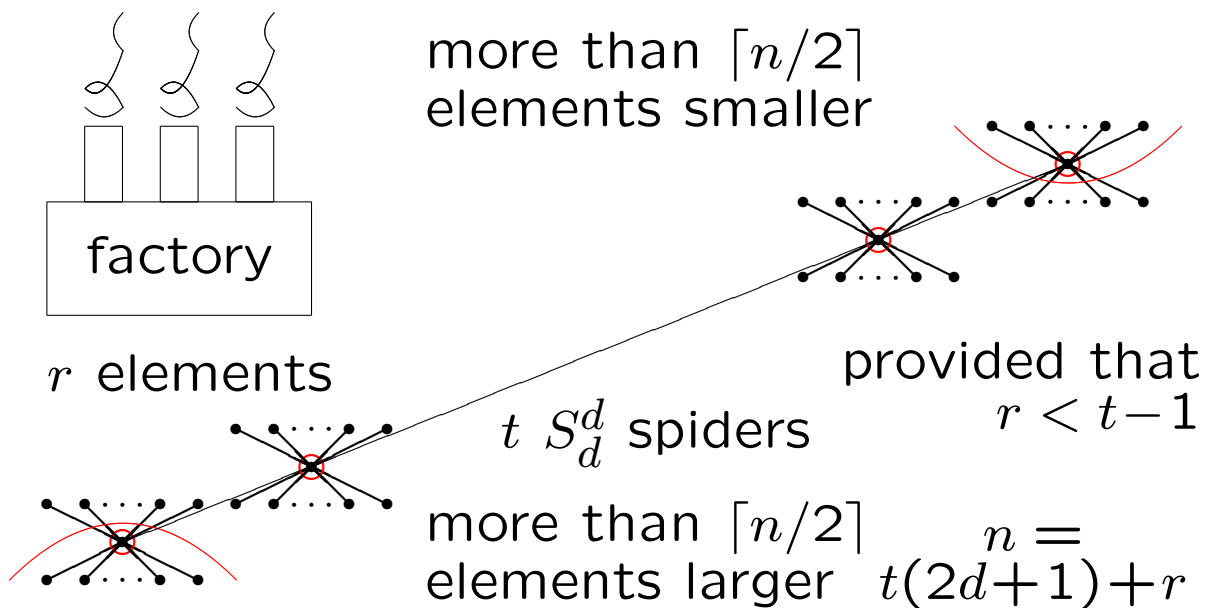
$$\begin{aligned}
 \Pr[X < \mu - \Delta] &\stackrel{\delta = \Delta/\mu}{=} \Pr[X < (1 - \delta)\mu] \\
 &\stackrel{\text{Theorem 4.3}}{\leq} e^{-\mu\delta^2/2} \\
 &= e^{-n^{2\alpha}/2}.
 \end{aligned}$$

For parameters $\alpha = 1/6$, $\beta = 2/3$, and $\gamma = 5/6$, we have that $\delta \leq 2e - 1$, so we can use the simplified Chernoff bound.

Spiders and their use



Hasse diagram of a S_d^d spider; the **centre** of the spider is circled.

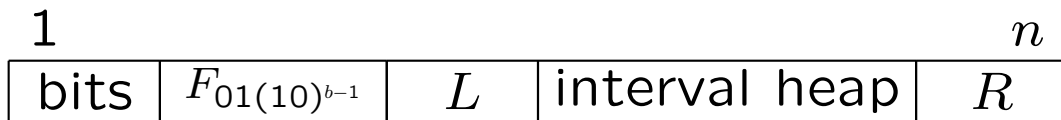


Keep the spiders in a **priority deque**, and repeatedly remove from this deque the spider with the smallest centre and the spider with the largest centre.

Deterministic algorithm using $o(n)$ extra space

1. Let e be a power of 2 between $\lfloor n^{3/10} \rfloor$ and $2 \lfloor n^{3/10} \rfloor$, and let $b = \log_2 e$, and $d = e - 1$.
2. Use a factory tree of type $F_{01(10)^{b-1}}$ to generate S_d^d spiders, and keep the spiders in a priority deque.
3. Repeatedly remove from the deque the spider S_{\min} with the smallest centre and the spider S_{\max} with the largest centre. Move the bottom (top) elements of S_{\min} (S_{\max}) to the **pool** L (R) of **left-eliminated** (**right-eliminated**) **elements**.
4. Use the elements of S_{\min} and S_{\max} that are not eliminated for new spiders.
5. Repeat the elimination process until $|L| > k - c \cdot e^3$ for some constant c .
6. Construct a heap storing the rest elements, and use that heap to eliminate the remaining elements no larger than the k th element.

Making it in-place



- The structure of $F_{01(10)^{b-1}}$ is regular; at each node only a bit is needed to indicate which of the two subtrees T_0 or T_1 is stored first. Bit encoding is used to get the bits needed.
- Due to the regularity also the pruning of $F_{01(10)^{b-1}}$ can be accomplished in-place.
- A multiway interval heap of height at most 3 is used to realize the priority deque, and it is be maintained between the pools L and R .
- The centre of S_{\min} (S_{\max}) is removed only every second round to keep the size of the heap section as a multiple of $2d+1$.

Conclusions

- Both our in-place algorithms are quite complicated, but if $o(n)$ extra space is available, the bit encoding can be avoided.
- In CPH STL (see www.cphstl.dk) the implementation of the `nth_element` function is based on the randomized algorithm using $o(n)$ extra space.
- Is it possible to reach the optimal resource bounds in the randomized case if only \ominus is given as part of the input?
- Can the deterministic algorithm be improved?