October 22, 2020

Quicksort and Quickselect

Randomized algorithms

We are given an array $A[1:n]$ of different elements for the sake of simplicity (algorithms work in the case when duplicates are present as well)

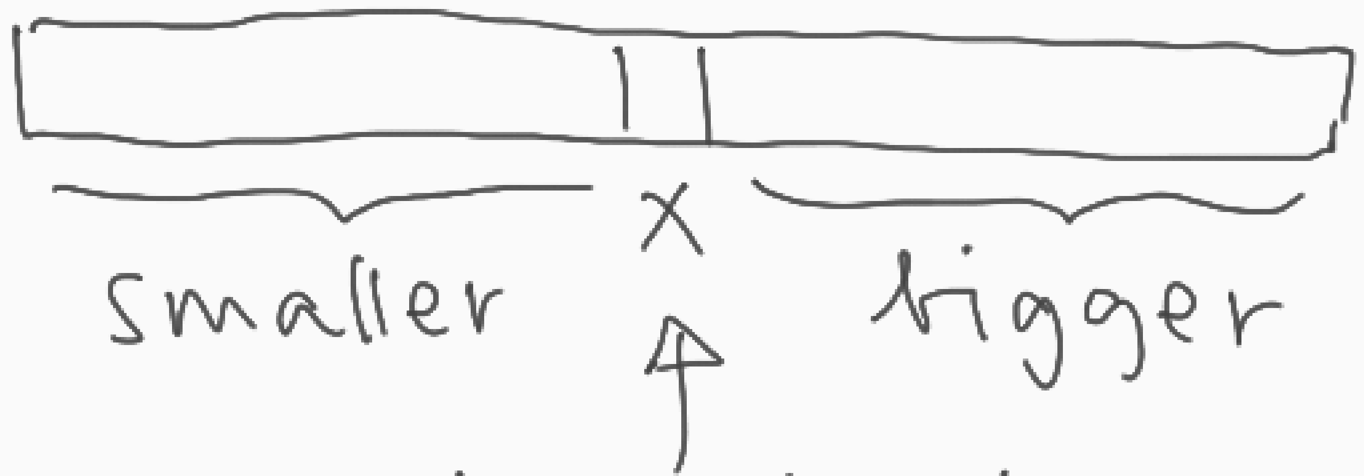
Quicksort

Divide and conquer algorithm (like merge sort)

Idea

Take a random element, say x, of the array

Rearrange the array in such a way that the elements smaller than x are before x and the elements greater than x are after x



it is in its right place

Then sort recursively the first part (before x) and the second part (after x)

Complexity

Besides the size of the array the cost depends on the random choices

Worst case :

The random element is always the biggest element of the subarray

→ (complicated) max selection sort

$O(n^2)$ time

Best case

(less obvious)

The random element is always the middle one in the subarray

→ $O(n \log n)$

[if the rearrangement can be done in linear time]

For randomized algorithm we usually consider the expected cost (average)

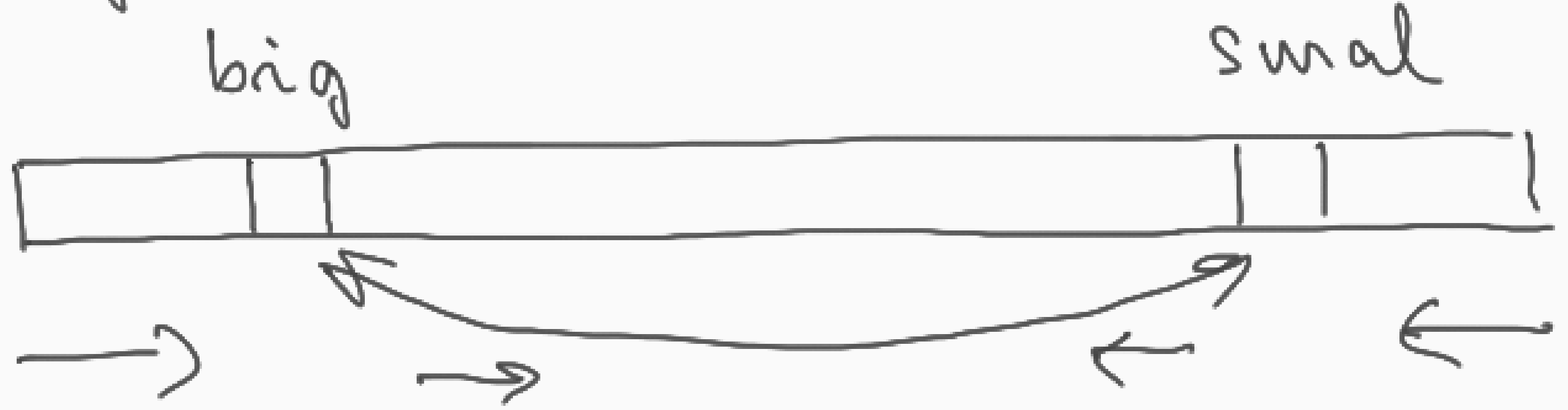
$\rightarrow O(n \log n)$

[in fact $\leq 1.39\, n \log n$]

<u>Details</u>

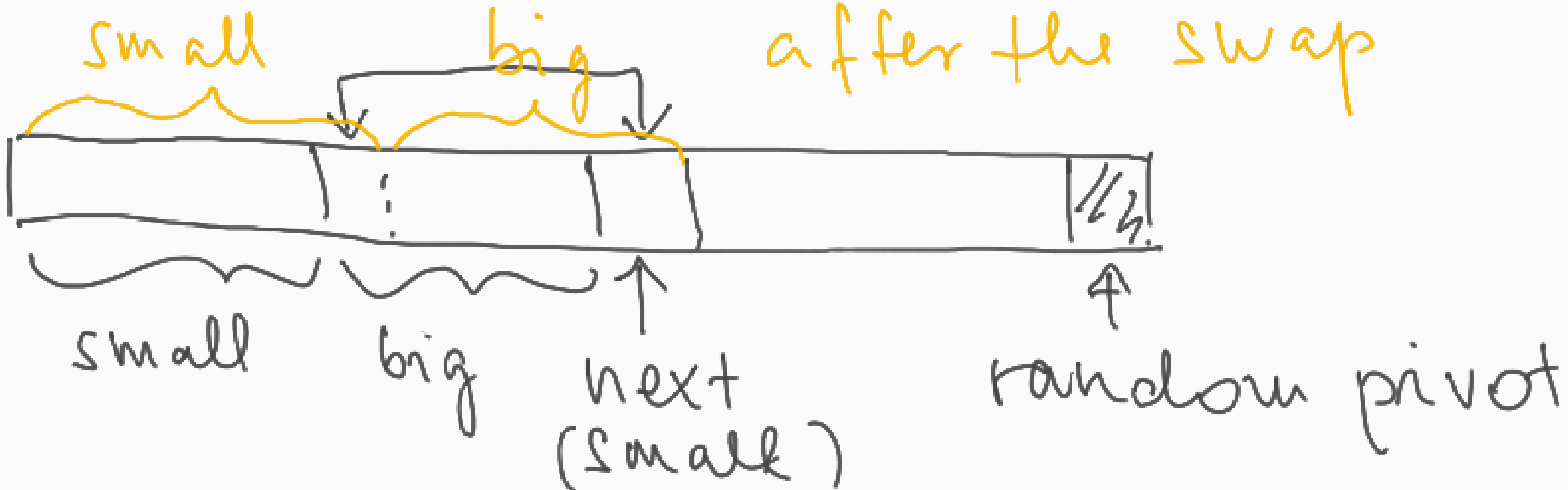
Rearrangement (linear time in-place)

Original idea

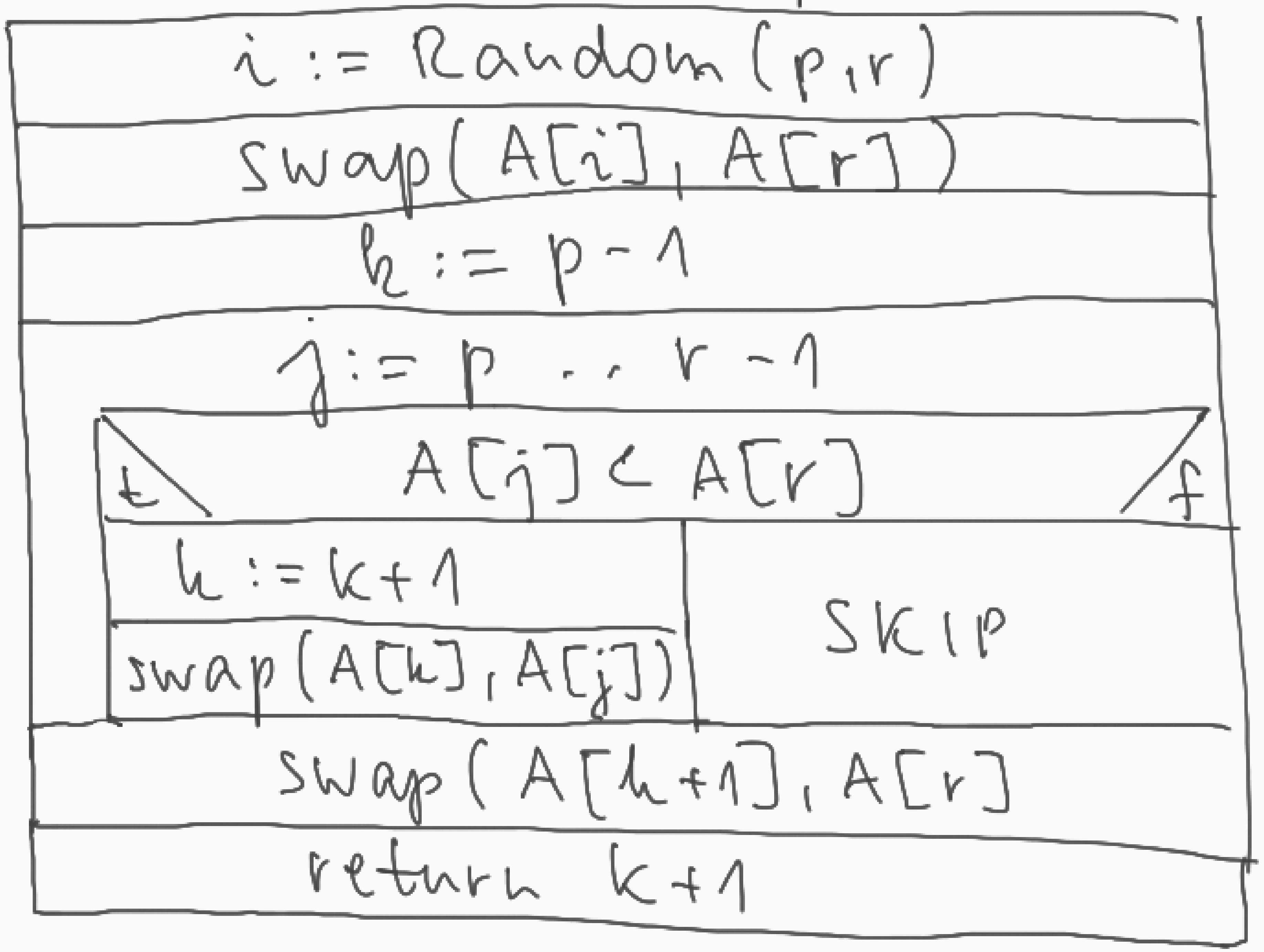


simple, but not easy to implement

Improved idea (Lomuto)



Partition (A, p, r)



A[p : r]

random $p \le i \le r$

last element in the "small part"

Quicksort(A, p, r) ← (A, 1, n)

<table>
<tr><td>t</td><td>p < r</td><td>f</td></tr>
<tr><td colspan="2">q := Partition(A, p, r)</td><td rowspan="3">SKIP</td></tr>
<tr><td colspan="2">Quicksort(A, p, q-1)</td></tr>
<tr><td colspan="2">Quicksort(A, q+1, p</td></tr>
</table>

Example (partition)

6	3	7	5	2	8	4	1

Random $i = 4$

1	2	3	4	5	6	7	8
6	3	7	1	2	8	4	5

$k = 0\ ,\ j = 1$

6	3	7	1	2	8	4	5

$k = 0\ ,\ j = 2$

3	6	7	1	2	8	4	5

$k = 1\ ,\ j = 3$

3	6	7	1	2	8	4	5

$k = 1\ ,\ j = 4$

3	1	7	6	2	8	4	$\cancel{5}$

$k = 2 \quad j = 5$

3	1	2	6	7	8	4	$\cancel{5}$

$k = 3 \quad j = 6$

3	2	1	6	7	8	4	$\cancel{5}$

$k = 3 \quad j = 7$

3	2	1	4	7	8	6	5

$k = 4$

we leave the loop

3	2	1	4	5	8	6	7

Quickselect

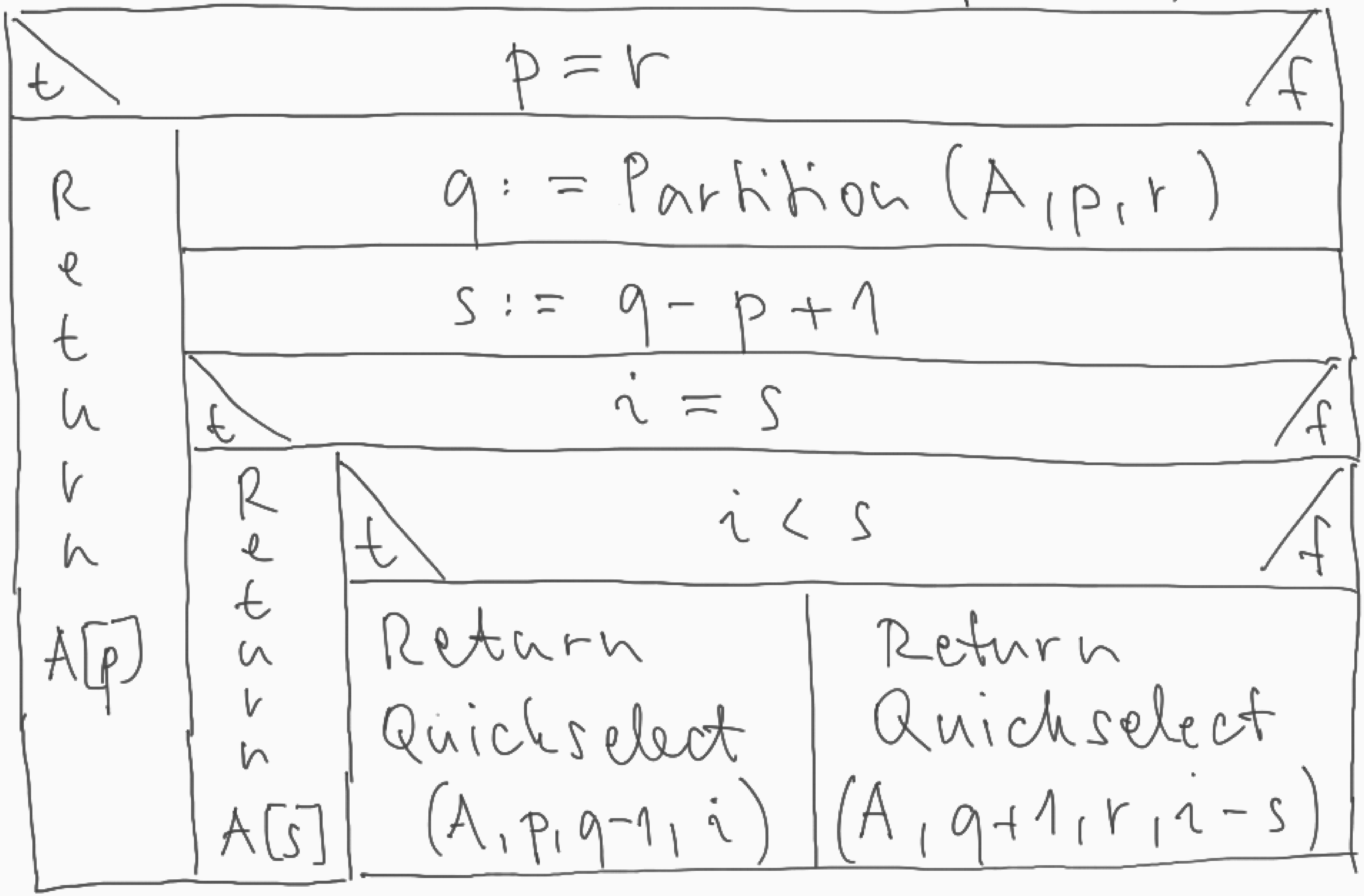
We have a paramerer $1 \leq k \leq n$ here as well and we want to identify the k^{th} element in the array according to the increasing order of the elements

Call first Partition for $A[1:n]$ and let the procedure return j

We have 3 cases

1. if $j = k$ then we are done, report $A[j]$ as the k^{th} element
2. if $j > k$ then select the k^{th} element recursively in $A[1 : j-1]$
3. if $j < k$ then select the $(k-j)^{th}$ element recursively in $A[j+1 : n]$

QuickSelect (A, p, r, i) $\leftarrow (A, 1, n, k)$



Expected cost : $O(n)$