

Suppose we are given an array of n distinct elements. The i th **order statistic** is the i th smallest element.

- $i = 1$: minimum element
- $i = n$: maximum element
- $i = \lfloor (n + 1)/2 \rfloor$: median

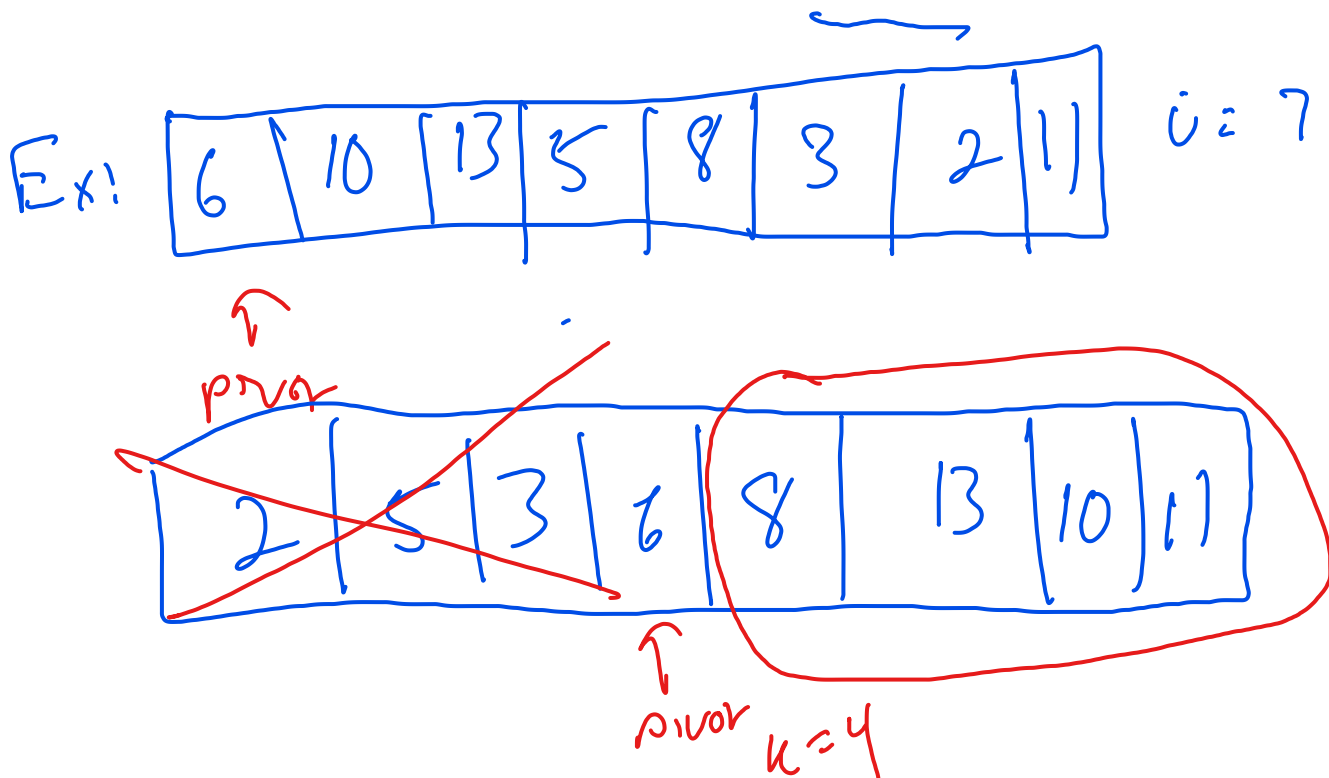
$O(n \log n)$

Trivial algorithm: Sort the input using merge sort. Return the element at index i . Worst case running time is $\Theta(n \log n)$.

Can we develop an algorithm which has a better worst-case running time than the trivial algorithm?

Consider a divide and conquer algorithm for this problem:

1. Use a randomized partitioning scheme similar to the one used in quicksort (choose a pivot and partition the elements into two sets “around” the pivot).
2. Let k be the index of the pivot after partitioning.
3. If $i = k$, then the pivot is the element we are looking for, and we return the pivot.
4. If $i < k$, then we know that the element we are looking for is in the first subarray, and we recursively find the i th smallest element in this subarray.
5. If $i > k$, then we now that the element we are looking for is in the second subarray, and we recursively find the element in position $i - k$ in this subarray.



If we choose a good pivot whose largest subarray is of size at most $9n/10$:

$$\left[\begin{array}{c} n \\ 1 \\ \frac{an}{b} \\ 1 \\ \frac{\ln n}{\ln b} \\ \vdots \end{array} \right]$$

- $T(n) = T(n - 1) + dn$

\wedge

$n-1$

$n-2$

$n-3$

\vdots

n^2

$\approx \frac{9n}{10} = cN$

for $c = \frac{9}{10}$

3

The running time will depend on the sizes of the subproblems we generate. The two subarrays computed by partition will be of size $(\underline{k}, \underline{n - k - 1})$ for some $k \in \{0, 1, \dots, n - 1\}$.

To obtain an upper bound on the running time, we will assume that the i th element always falls in the larger subarray (i.e. the subproblem size will be $\underline{\max(k, n - k - 1)}$).

Thus we can express the running time of the algorithm in the following way:

$$T(n) = \begin{cases} T(\max(0, n-1)) + \ln & \text{if } 0 \text{ is } n-1 \text{ split} \\ T(\max(1, n-2)) + \ln & \text{if } 1 \text{ is } n-2 \text{ split} \\ \vdots \\ T(\max(n-1, 0)) + \ln & \text{if } n-1 \text{ is } 0 \text{ split} \end{cases}$$

We can express the running time as a sum, if we introduce the following indicator RV X_k :

$$X_k = \begin{cases} 1 & \text{if } k : n-k-1 \text{ split} \\ 0 & \text{otherwise} \end{cases}$$

Using these indicator RVs, we have that the running time of the algorithm is:

$$T(n) = \sum_{k=0}^{n-1} X_k \cdot \left(T(\max(k, n-k-1)) + 2n \right)$$

$$\leq 2 \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} X_k \cdot (T(k) + 2n)$$

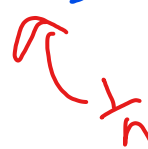
Calculating $E[T(n)]$:

$$E[T(n)] = E\left[2 \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} X_k \cdot (T(k) + \Delta n)\right]$$

$$= 2 \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E[X_k \cdot (T(k) + \Delta n)]$$

$$= 2 \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E[X_k] \cdot E[T(k) + \Delta n]$$

$$= \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E[T(k) + \Delta n]$$



$$= \left(\frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E[T(k)] \right) + \Delta n$$

Note that the subproblem $T(k)$ is a RV (which depends on pivot choices for the future subproblems). We can use induction to show that $E[T(n)] = O(n)$.

$$\sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} k \leq \frac{3}{8} n^2$$

Substitute $c \cdot k$ for $E[T(k)]$

$$\frac{2}{n} \cdot \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} E[T(k)] + dn$$

$$\leq \frac{2}{n} \sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} c \cdot k + dn$$

$$\leq \frac{2c}{n} \left(\frac{3}{8} n^2 \right) + dn$$

$$= \frac{3}{4} cn + dn = (cn - \frac{1}{4} cn) + dn$$

$$\leq cn \text{ when } dn - \frac{1}{4} cn \geq 0$$

$$\boxed{c \geq 4d}$$

The randomized algorithm described is excellent in practice (linear expected running time); however, the worst case running time ($\Theta(n^2)$) is slower than the trivial algorithm.

Is it possible to obtain an algorithm whose worst-case running time is better than the $\Theta(n \log n)$ running time of merge sort?

Answer is yes [Blum, Floyd, Pratt, Rivest, and Tarjan - 1973].

The idea is to recursively generate a good pivot.

So the running time is $T(n) = T(n/5) + T(7n/10 + 3) + dn$. We will use induction to show that $T(n) = O(n)$.

The worst case running time is $O(n)$, but the hidden constants are quite large. The algorithm is mainly of theoretical interest.

In practice, it is better to use the randomized algorithm than this algorithm.