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

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

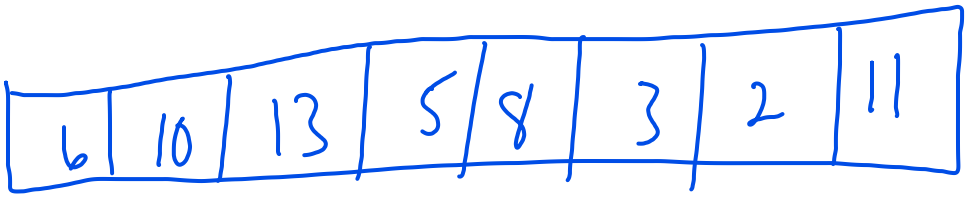
Trivial algorithm: Sort the input using merge sort. Return the element at index i. Worst case running time is $\Theta(n \log n)$.

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

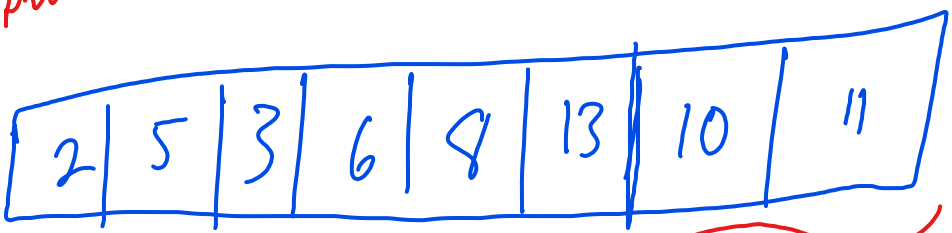
Consider a divide and conquer algorithm for this problem:

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 ith 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.

$i = 7$



pivot



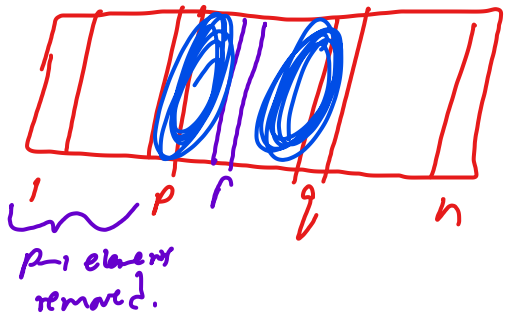
pivot

Next Subproblem

update: $i = i - k \Rightarrow i = 7 - 4 = 3$

```
Rand-Select(A, p, q, i)
{
    if p==q, return A[p]

    r = Rand-Partition(A, p, q)
    k = r-(p-1)
    if i==k, return A[r]
    else if i<k, return Rand-Select(A, p, r-1, i)
    else, return Rand-Select(A, r+1, q, i-k)
}
```



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

- $T(n) \leq T(9n/10) + dn$

$a = 1,\ b = \frac{10}{9},\ f(n) = n^1,\ n^{\log_b a} = n^0$

$n^1 \in \Omega(n^{0+\varepsilon})$ for $\varepsilon = 1$

$af(\frac{n}{b}) = \frac{9n}{10} \Rightarrow c = \frac{9}{10}$ ✓

Case 3 holds. $T(n) \in \Theta(n)$

If we get unlucky and pick the worst possible pivot:

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

n
1
$n-1$
1
$n-2$
$\vdots$
1

$\sum_{i=1}^{n} i = \Theta(n^2)$

What is the expected running time of the algorithm?

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

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

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

$$T(n) \leq \begin{cases} T(\max(0, n-1)) + dn & \text{if } 0:n-1 \text{ split} \\ T(\max(1, n-2)) + dn & \text{if } 1:n-2 \text{ split} \\ \vdots & \\ T(\max(n-1, 0)) + dn & \text{if } n-1:0 \text{ split} \end{cases}$$

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:

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

$$\leq 2 \cdot \sum_{k=\lceil \frac{n}{2} \rceil}^{n-1} X_k \cdot \left(T(k) + dn \right)$$

Calculating $E[T(n)]$:

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

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

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

$E[X_k] = 1/n$

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

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

$\frac{2}{n}\left(\frac{n}{2}\right) dn = dn$

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

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 $E[T(n)] = O(n)$.

Inductively Assume: $E[T(k)] \le c \cdot k$ for some $c > 0$ for all $k < n$.

We will use the following fact: $\sum_{k=\lfloor \frac{n}{2} \rfloor}^{n-1} k \le \frac{3}{8} n^2$.

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

$$\le \frac{2c}{n} \cdot \frac{3n^2}{8} + dn$$

$$= \frac{3}{4} cn + dn$$

$$= \underbrace{cn}_{\text{desired}} - \underbrace{\frac{1}{4}cn + dn}_{\text{residual}}$$

$$\le cn$$

$\leftarrow$ true when $-\frac{1}{4}cn + dn \le 0$, i.e. $c \ge 4d$

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.