# Order Statistics KT 13.5 DPV 2.4

Frank

October 1, 2021

## 1 Introduction

Given a list of numbers

$$A = a_1, a_2, \ldots, a_n$$

We can easily compute the minimum element, max, average in $O(n)$ time

- Another statistical measure known as median of $A$ is the middle element if the list was sorted. However, this list can be given to us in any order. We can define the median as

$$a_{\pi(\lceil \frac{n}{2} \rceil)}$$

Where $\pi$ is a permutation of $1 \ldots n$ s.t.

$$a_{\pi(1)} \leq a_{\pi(2)} \leq \ldots \leq a_{\pi(n)}$$

Note that we can just sort the list and pick the middle element in $O(n \log n)$, but we can actually do this is $O(n)$

To solve this problem, we solve a more general problem, the selection problem.

## 2 Selection Problem

Given list $A$ & $k$ s.t. $1 \leq k \leq |A|$, find k$^{\text{th}}$ smallest element in $A$

Note that if $k = 1$, then return the minimum, if $k = |A|$ then we return the maximum, but if $k = \lceil \frac{n}{2} \rceil$, then we return the median.

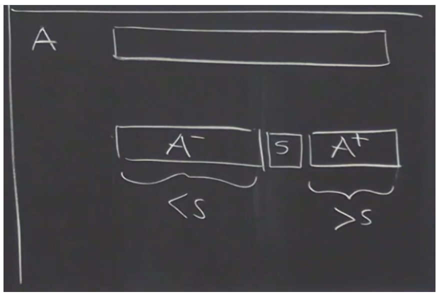
This is a more general case of our original problem, but like induction, general case DC algorithms are generally easier to prove because we get to leverage a strong induction hypothesis

### 2.1 Assumption

Assume elements in $A$ are distinct, but we can do this and not assume this without loss of generality by converting the elements into pairs $(a_i, i)$ with the first entry being the element value and the second entry being the element's position, and since every position is different, we then have the means of comparing two elements with the same value. But to make our life easier, we make the above assumption.

## 2.2 Algorithm

Visual example



```
procedure SELECT($A, k$)                                ▷ $1 \le k \le |A|$
    if ($|A| = 1$) then
        return $A[1]$
    else
        $s \leftarrow$ some element of $A$                   ▷ splitter
        $A^- \leftarrow$ elements in $A < s$                 ▷ not sorted
        $A^+ \leftarrow$ elements in $A > s$                 ▷ not sorted
        if ($k \le |A^-|$) then                          ▷ look in $A^-$
            return SELECT($A^-, k$)
        else if ($k = |A^-| + 1$) then                   ▷ $s$ is the kth smallest
            return $s$
        else                                             ▷ look in $A^+$
            return SELECT($A^+, k - (|A^-| + 1)$)
```

Note that if we had an algorithm that solves for median, we wouldn't be able to do the above, because we cannot recursively call on the median, but we could recursively call on the kth smallest element

## 2.3 Running time

- Suppose $s$ is the ith smallest element in $A$, then evaluate worse case as recurrence

$$T(n) = T(\ max(i-1, n-i-1)\ ) + cn$$

We use *max* to because we want the worst case. Then our recursive step input depends on $i$. Note we include a constant amount of work in line 6-7.

- Then consider, what is the best choice for our splitter $s$, to minimize our recurrence. The obvious choice is that we want $s$ to be the median, because we are taking the max. But remember that finding the median every time is not an option.

If $s$ is the median, then we have the best case:

$$T(n) = T\left(\frac{n}{2}\right) + cn \implies O(n)$$

If we are very unlucky, then we choose $s$ to be either the minimum or maximum, then the recurrence becomes:

$$T(n) = T(n-1) + cn \implies O(n^2)$$

## 2.4 Observations

1. Algorithm is linear for <u>any</u> $b > 1$. (master thm)

   ex. Shrink the input by some factor greater than 1, then we would still have $a < b^d$

   Thus, we do not have to pick to median, but something close to the median

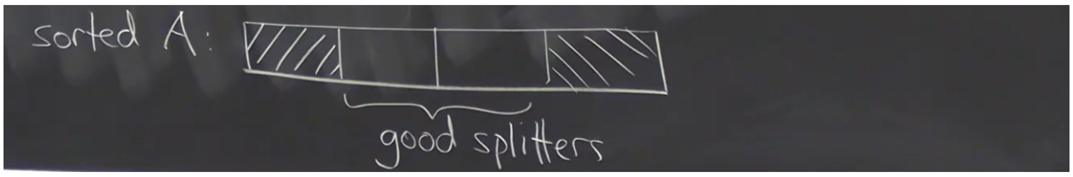
2. We do not need to have a *good* splitter every recursion

   ex. Even if we get unlucky and pick the min or max every other time, we'd still only have a linear time

   Define $s$ to be good if

   - At least $\frac{1}{4}$ elements of $A$ are $< s$
   - At least $\frac{1}{4}$ elements of $A$ are $> s$

   If $A$ is sorted, we'd have:



   Note $A$ is not necessarily sorted.

3. If my splitter is *good*, then the worst case subarray we are giving to the recursive step is $\frac{3}{4}$ (extreme ends of the good splitters range).

$$\text{good splitter} \implies \text{input sizes shrinks from } n \text{ to something} \leq \frac{3}{4}n$$

4. Probability of picking a good splitter is $\frac{1}{2}$ chance.

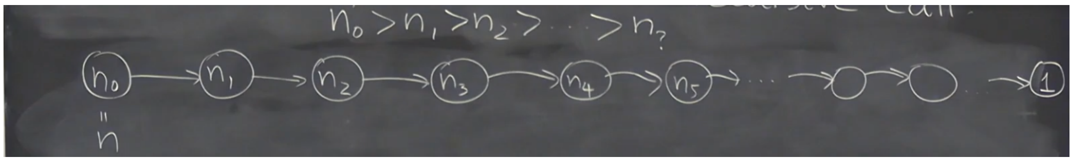
   Then on average, the expected <u>number of trails before we pick a good splitter</u> (call this a RV $Z$) is 2.

$$\begin{aligned} E[Z] &= \sum_v v \cdot P[Z = v] \\ &= 1 \cdot \frac{1}{2} + 2 \cdot \frac{1}{4} + 3 \cdot \frac{1}{8} + \ldots \\ &= \sum_{i=0}^{\infty} \frac{i}{2^i} \\ &= \frac{\frac{1}{2}}{(\frac{1}{2})^2} \\ &= 2 \end{aligned}$$

## 2.5 Randomized SELECT

```
procedure R-SELECT($A, k$)                                          ▷ $1 \leq k \leq |A|$
    if ($|A| = 1$) then
        return $A[1]$
    else
        $s \leftarrow$ uniformly random element from $A$
        $A^- \leftarrow$ elements in $A < s$                                ▷ not sorted
        $A^+ \leftarrow$ elements in $A > s$                                ▷ not sorted
        if ($k \leq |A^-|$) then                                        ▷ look in $A^-$
            return R-SELECT($A^-, k$)
        else if ($k = |A^-| + 1$) then                          ▷ $s$ is the kth smallest
            return $s$
        else                                                            ▷ look in $A^+$
            return R-SELECT($A^+, k - (|A^-| + 1)$)
```

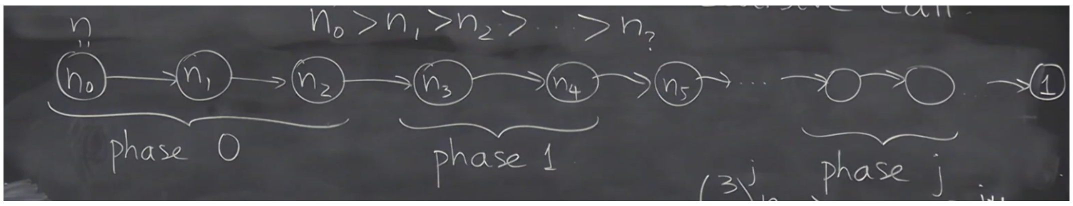
Let $n_i$ be the size of input in the ith recursive call



Imagine the execution as a recursive tree, divide up the execution into phases such that every phase lasts until the input size becomes $\frac{3}{4}$ of what it was before.

Let phase $j$ be recursive calls where input size is

$$\left(\frac{3}{4}\right)^{j+1} \cdot n < n_i \leq \left(\frac{3}{4}\right)^{j} \cdot n$$



Note that we may be unlucky that we take a while to reach the next phase, or we may get lucky that we skip some phases, for example, choosing min vs. choosing the median.

- **Running time:**

  Define RV $X$ = number of steps taken by the algorithm on input of size $n$

  Define RV $X_j$ = number of steps taken by phase $j$ recursive calls

  Then

$$X = X_1 + X_2 + X_3 + \dots \implies E[X] = E\left[\sum_j X_j\right] = \sum_j E[X_j]$$

  Note that

$$\begin{aligned} X_j \leq &[\text{maximum number of steps for } \underline{\text{one}} \text{ recursive call in phase } j] \\ &\times [\text{number of phase } j \text{ recursive calls made}] \end{aligned}$$

  Observe that the [number of phase $j$ recursive calls made] is actually our RV $Z$, because we know if we choose a good splitter, we would move on to the next phase for sure.

  Also observe that [max number of steps for one recursive call in phase $j$] is

$$\underline{c \cdot \left(\frac{3}{4}\right)^j \cdot n}$$

  So we have

$$\begin{aligned} E[X_j] &= c \cdot \left(\frac{3}{4}\right)^j \cdot n \cdot E[Z] \\ &= 2cn \cdot \left(\frac{3}{4}\right)^j \end{aligned}$$

  Finally,

$$\begin{aligned} E[X] &= \sum_j E[X_j] \\ &= \sum_j 2cn \cdot \left(\frac{3}{4}\right)^j \\ &= 2cn \cdot \left(\frac{1}{1 - \frac{3}{4}}\right) \\ &= 8cn \end{aligned}$$

  $\therefore$ the <u>expected</u> running time of R-SELECT is $O(n)$

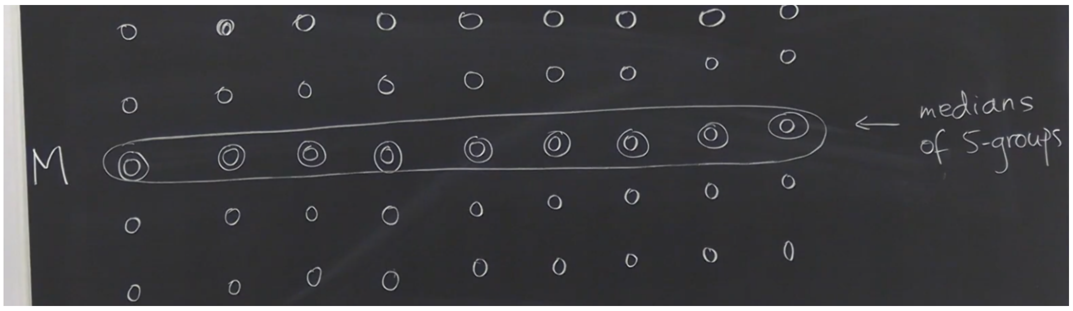
  Note that worst case is $O(n^2)$

## 2.6 D-SELECT

Deterministic, not random

```
procedure D-SELECT($A, k$)                          ▷ $1 \le k \le |A|$
    if ($|A| \le 5$) then                           ▷ brute force solvable
        return brute force solution
    else
        $s \leftarrow (\star)$
        $A^- \leftarrow$ elements in $A < s$        ▷ not sorted
        $A^+ \leftarrow$ elements in $A > s$        ▷ not sorted
        if ($k \le |A^-|$) then                     ▷ look in $A^-$
            return D-SELECT($A^-, k$)
        else if ($k = |A^-| + 1$) then              ▷ $s$ is the kth smallest
            return $s$
        else                                        ▷ look in $A^+$
            return D-SELECT($A^+, k - (|A^-| + 1)$)
```

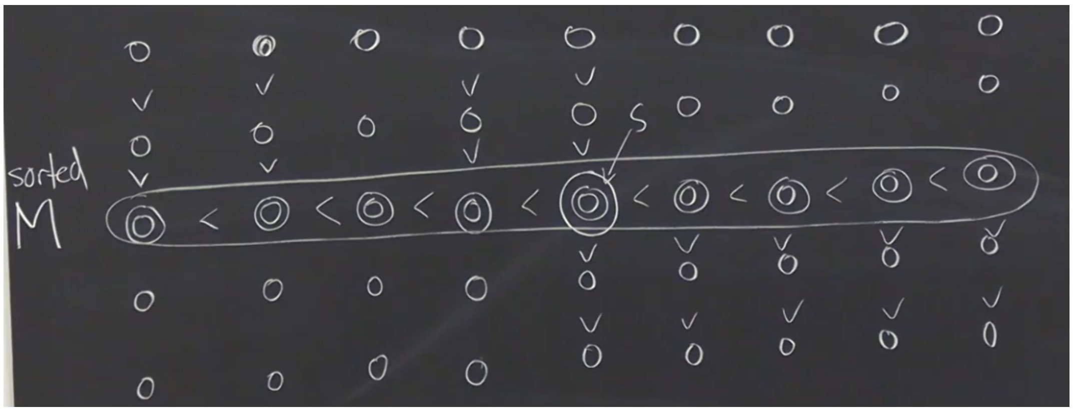
- **Choosing splitter** $s$ $(\star)$ :
  - partition $A$ into $\frac{n}{5}$ groups of size 5 each
  - Find the median of each 5-group by sorting them
  - $M =$ list of $\lceil \frac{n}{5} \rceil$ *medians*



  - $s = \text{D-SELECT}\left(M, \left\lceil \frac{|M|}{2} \right\rceil\right)$

    Where $\left\lceil \frac{|M|}{2} \right\rceil$ is the median of medians

Imagine that $M$ is sorted, then we'd pick a *good* splitter $s$, 100% of the time, because it is greater than $\frac{1}{4}$ of the elements in $A$, and less than $\frac{1}{4}$ of the elements in $A$. The below example shows this, imagine that the entire matrix is sorted (look at the inequality signs)



- **Running Time:**

$$T(n) = c \text{ for } n \leq 5$$

$$T(n) \leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(\left\lfloor \frac{3}{4}n \right\rfloor\right) + cn \text{ for } n > 5$$

Since we have a recursive step for determining a good splitter, and because we have a good splitter, we are guaranteed to throw away at least $\frac{1}{4}$ of the array $A$ in the main recursive call, plus the amount of work to generate $A^-$ and $A^+$, which is linear

The above recurrence is not applicable to Master Thm, so lets simplify

$$T(n) \leq T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(\left\lfloor \frac{3}{4}n \right\rfloor\right) + cn \text{ for } n > 5$$

This is kinda cheating, but we could rewrite the above algorithm with a bit more overhead so that above recurrence is the result.

- Let $T(n) \leq dn$ where $d$ is some constant, so assume that for values smaller than $n$, our inequality holds. (IH)

By def:

$$\begin{aligned} T(n) &\leq T\left(\left\lfloor \frac{n}{5} \right\rfloor\right) + T\left(\left\lfloor \frac{3}{4}n \right\rfloor\right) + cn \\ &\leq d\left\lfloor \frac{n}{5} \right\rfloor + d\left\lfloor \frac{3}{4}n \right\rfloor + cn \text{ (IH)} \\ &\leq d\frac{n}{5} + d\frac{3}{4}n + cn \\ &\leq \frac{4dn + 15dn}{20} + cn \\ &= \frac{19}{20}dn + cn \end{aligned}$$

Note we want $\frac{19}{20}dn + cn \leq dn \iff \frac{1}{20}dn \geq cn \iff d \geq 20c$

Thus, if we pick $d \geq 20c$, then

$$T(n) \leq dn$$

So pick $d = max(20c, \frac{T(1)}{1}, \frac{T(2)}{2}, \dots, \frac{T(5)}{5})$ to cover base cases.

Then we have a deterministic algorithm whose worst case time is $O(n)$

- Lets come back to why we chose 5 in the first place.

  We want to satisfy

  $$\frac{n}{x} + \frac{3}{4}n < 1$$

  And it turns out that 5 is the minimum number that satisfies this inequality.

  This algorithm should remind us of QUICKSORT, which picks a pivot.

## 2.7 Is choosing a careful splitter worth it?

Probably not, but for some real applications, we cannot afford to be *unlucky*