SELECT (Ch. 9, Lect #6)

Problem Statement:

Input: An unsorted array \( A[1, 2, \ldots, n] \) of numbers and pos. int. \( k \); \( 1 \leq k \leq n \).

Desired output: An index \( j \) such that \( A[j] \) is the \( k^{th} \) smallest.


"Brute Force" Algorithm

- Sort the array \( \Theta(n \log n) \)
- Select \( k^{th} \) element.

\[ \begin{align*}
& \text{if } k=1, \quad \Theta(n) \\
& k=2, \quad \Theta(n) \\
& k: \text{fixed, ind. of } n: \Theta(n) \\
& \text{But what about } k(n) ?
\end{align*} \]

Goal: Want an algorithm with \( \Theta(n) \) complexity.

This algorithm uses a deterministic subroutine from the (randomized) QUICKSORT algorithm called PARTITION. (Prerequisite material)
PARTITION \[ \Theta (n) \] (p171 of your book) \[ (2) \]

**Input:** An unsorted array \( A[1, 2, \ldots, n] \) and a pivot value \( z = A[p] \).

**Output:** A rearranged (but not totally sorted) array \( A' \) such that it looks like:

\[ A'[\text{low}], A[p], A'[\text{high}] \]

Where each element in \( A'[\text{low}] \) is \( \leq A[p] \) and each element in \( A'[\text{high}] \) is \( \geq A[p] \).

**Note:** Neither \( A'[\text{low}] \) nor \( A'[\text{high}] \) needs to be sorted.

**Algorithm SELECT** (\( A[1], \ldots, n \))

This is an example of Divide-and-Conquer algorithm which uses another Divide-and-Conquer algorithm as a Subroutine.

We describe the algorithm next.
Small: will be decided at the end so that we get a $O(n)$ algorithm.

Imagine, that somehow, we have selected a "good" pivot element $A[p]$, and we call $\text{PARTITION}$ on $A[1, \ldots n]$ using $A[p]$.

\[ A[\text{Low}] \quad A[p] \quad A[\text{High}] \]

\[ \text{Low} \quad \text{High} \]

is the result.

\[
\text{if} \quad |A[\text{Low}]| = k-1, \text{ return } A[p] \text{ as } k^{th} \text{ smallest}
\]

\[
\text{else if} \quad |A[\text{Low}]| > k-1, \text{ do } \text{SELECT} [A[\text{Low}], k]
\]

\[
\text{else if} \quad |A[\text{Low}]| < k-1, \text{ do } \text{SELECT} [A[\text{High}], k-|A[\text{Low}]| - 1]
\]

This has the "feel" of $\text{BINARY SEARCH}$ except we do not know $A[p]$ is the middle element of the sorted version of $A[1, \ldots n]$.

The corresponding recurrence would be:

\[
t(n) = t\left(\frac{|A[\text{Low}]|}{2}\right) + \frac{Cn}{2}
\]

For partition.

In the worst case, we need to keep both $|A[\text{Low}]|$ and $|A[\text{High}]|$ as small as possible.
But keeping \( A \text{ [low]} \) \( \cong \) \( A \text{ [high]} \) is itself a problem of this kind! \([\text{Find median!}]\)

So we do an "approximation" of the above. We keep each side \( \leq 70\% \) of original size. And for this we use divide-and-conquer!

**Selection of \( A[p] \)**

1. Group the elements into sets of 5 each
   (Why "5" will be discussed later)
   \[ A \text{ [1...5]} \]
   \[ A \text{ [6...10]} \]
   \[ \ldots \]
   \[ A \text{ [15...20]} \]

\[ \left\lfloor \frac{n}{5} \right\rfloor \]

\[ \times \times \times \]

\[ B \text{ [5...10]} \]

\[ B \text{ [10...15]} \]

\[ \ldots \]

\[ B \text{ [n...n]} \]

\[ \Theta(n) \]

Work.

2. do SELECT (\( A \text{ [1...} \left\lfloor \frac{n}{5} \right\rfloor , \left\lceil \frac{n}{10} \right\rceil \) )

\[ \text{output} \ A[p] \]

Do PARTITION \( (A \text{ [1...} n], A[p]) \) \( \Theta(n) \)

\[ \text{Now we show that with this } A[p] \]
\[ |A \text{ [low]}|, |A \text{ [high]}| \leq \frac{7}{10} n + 6 \]
The following illustration is only done to help in visualizing the process; no sorting is done in the algorithm at any point.

Imagine the elements of each group are arranged vertically in increasing order (going up) so that the median of each group is visualized to be in the middle. Further imagine that the groups are arranged so that their medians are in increasing order (going to the right).

\[ \text{Position Unknown in the above Visualization.} \]
Hence, recurrence for this algorithm is: 

\[ t(n) = t\left(\frac{n}{10}\right) + t\left(\frac{n}{5}\right) + cn \]

Using substitution method, show \( t(n) = O(n) \) 
(see page 222 of your book)

Hence \( t(n) = O(n) \)

Now we turn to some problems from geometry.
Ref: Ch 33 of your book, Lb 6.

**Convex Hull in \( \mathbb{R}^2 \) (Ch 33)**

**Def Convex Set:** \( S \) is NOT convex if \( \exists x', x^2 \in S \), such that \( \lambda x' + (1-\lambda)x^2 \notin S \) for some \( \lambda \in [0, 1] \).

**Bounded**

**Convex Polygon**

**Unbounded**

**Convex Polyhedral Set**

**Bounded Convex but not Polyhedral**
Points in input

Smallest convex set containing a set of points is called its convex-hull.

Problem statement
Input: Points \((P_1, P_2, \ldots, P_n) \in \mathbb{R}^2\).
\[ P_i = (x_i, y_i) \quad i = 1, \ldots, n \]

Desired output: \(CH\{P_1, \ldots, P_n\}\)
Operation: +, -, \times, \div, Comparison
Goal: An algorithm with w.c. complexity \(O(n \log n)\)
MACRO: Given an ordered triple \((p, p', p'')\) of \(\Theta(1)\) points, want to know whether \(p''\) is on, above, or below of directed line (left) (right) segment \((p, p')\)?
Think of drawing a straight line from p to p' and extending it in both directions. Assume you are at p, looking towards p'.

1. Any point on the line joining p and p' (and its extension in both directions) is on \((p, p')\).

2. Any point in the half-space obtained by removing \((p, p')\) and its extension that is to the "left" (when you look in the direction of p to p') is above or to the left of \((p, p')\).

To do this on a computer, calculate determinant

\[
\begin{vmatrix}
2 & y & 1 \\
x'_2 & y'_2 & 1 \\
x''_2 & y''_2 & 1
\end{vmatrix}
\]

\[
\text{of } \theta(6) \quad \begin{cases} 
> 0 & \Rightarrow \text{ above (left)} \\
= 0 & \Rightarrow \text{ on} \\
< 0 & \Rightarrow \text{ below (right)}
\end{cases}
\]

for \(p''\) with respect to \((p, p')\).
Now the Main algorithm.

To simplify our presentation, we will assume the input is \(\text{pre-sorted}\) so that \(x_1 \leq x_2 \leq \ldots \leq x_n\) (\(\Theta(n \log n)\)). [Presentation in the book does not do this and hence more complicated]

\[
\frac{\text{Small}}{\Theta(1)} \quad n \leq 3:
\]

Case 1

Convex hull: Vertices \(P_1, P_2, P_3\);

for each vertex, the other is both its predecessor and successor.

Case 2

\[
det \neq 0
\]

Vertices: \(P_1, P_2, P_3\);

\[
\text{Succ}(P_i) = P_{i+1} \pmod 3
\]

\[
\text{Pred}(P_i) = P_{i-1} \pmod 3
\]

Divide: Two parts \(\{P_1, \ldots, P_{\lfloor n/2 \rfloor}\}, \{P_{\lfloor n/2 \rfloor + 1}, \ldots, P_n\}\)

\[
L \quad R
\]

Call \(CH(L)\), \(CH(R)\) using Main alg on piece, \(t(\lfloor n/2 \rfloor)\), \(t(\lceil n/2 \rceil)\).
Now we need "merge" part. In most d-and-e algorithms, this is the most important part. You need to describe this with care and precision.

Q: If we want the entire algorithm to take $O(n \log n)$ time, what is the maximum time we can permit for "merge"

$$t(n) = t(\lceil \frac{n}{2} \rceil) + t(\lfloor \frac{n}{2} \rfloor) + ?$$

"Merge" Part:

CH(L) produces as its output, a list of vertices of the convex polygon (smallest) containing points on the "left" together with succ(v) (right) and pred(v) for each vertex.

Note "left", "right" are justified since points have been presented in increasing order of Xi and $L = \{1, \ldots, \lceil \frac{n}{2} \rceil\}, R = \{\lceil \frac{n}{2} \rceil + 1, \ldots, n\}$. So the "picture" looks like what is shown on next page.
We need two "common tangents" called upper and lower bridges. Upper bridge has all points "below" and touches both CH(L) and CH(R). Similar notions hold for lower bridge. Once we get these, we can get CH(LUR) easily as shown above. If we take f(n) time in doing this, our recurrence relation is

\[ t(n) = t(\lceil \frac{n}{2} \rceil) + t(\lfloor \frac{n}{2} \rfloor) + f(n) \]

In order to have \( t(n) = O(n \log n) \), we need \( f(n) = O(n) \) and no "larger". This gives us the "goal" for merge procedure.
We show how to get upper bridge in $O(n)$ time using Macro. First we make an initial "guess" $\overrightarrow{AB}$ where $A$ is the "right most" point (also a vertex of $CT(L)$) on the left and $B$ as the "left most" point on the right

$A = \lfloor \frac{n}{2} \rfloor$, $B = \lfloor \frac{n}{2} \rfloor + 1$, $\overrightarrow{AB}$ is our first guess for upper bridge.

At any step we ask (possibly) two questions (this process is repeated till we get "No" to both)

$Q_1$: Is $\text{Succ}(A)$ on or above $(A, B)$?
   If yes, do $A \leftarrow \text{Succ}(A)$, repeat $Q_1$
   If no, go to $Q_2$

$Q_2$: Is $\text{Pred}(B)$ on or above $(A, B)$?
   If yes, $B \leftarrow \text{Pred}(B)$, repeat $Q_2$
   If no, go to $Q_1$
\# of possible location for A + \# \cdots \# B \leq n.

In at least 2 Q, Either A Change, or B Change

Unless we have obtained upper bound:

\begin{itemize}
  \item Time Complexity for \( n \) \leq 2n \& Q's
  \item Each Q: \( \Theta(1) \) using MACRO
  \item Time Complexity: \( O(n) \)
\end{itemize}

Hence alg. has \( O(n \log n) \) Complexity

Add Preprocessing: Still \( O(n \log n) \)

We will later show can not beat this

Now we come to the last of and e example

that we will discuss in this class, it is also

the most complicated one.

**CLOSEST-PAIR in \( \mathbb{R}^2 \)**

**Input:** Points \((P_1, P_2, \ldots, P_n)\); \( P_i = (x_i, y_i) \; i = 1 \ldots n \)

in \( \mathbb{R}^2 \).

**Definition:**

\[
    d(P_i, P_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} \quad i \neq j
\]

( Euclidean distance )

**Desired Output:** indices \( i, j \), \( i \neq j \) such that

\[
    d(P_i, P_j) = \min_{k, l \in \{1, \ldots, n\}} d(P_k, P_l)
\]

We discuss this next.