Venkatesh Kadali
AD22232014.

Homework 3 (CS5050)

Problem 1: An array $A[1---n]$ (unimodal array)

Input: A unimodal array $A[1--n]$

Output: peak entry p

Time: $O(\log n)$

Algorithm: Binary Search

First, find the value $A[n/2]$, $A[n/2-1]$ & $A[n/2+1]$

Because with $A[n/2]$ itself, we cannot predict whether p lies before or after $n/2$.

Now, we have 3 possibilities, as the elements ~~even~~ in the array A are distinct.

Case 1: if $A[n/2-1] < A[n/2] < A[n/2+1]$,
then entry $n/2$ must come [illegible] before p, then we can continue recursively with $n/2+1$ to n

Case 2: if $A[n/2-1] > A[n/2] > A[n/2+1]$,
then entry $n/2$ must come after p, then we can continue recursively with 1 through $n/2-1$.

Case 3: if $A[n/2] > A[n/2+1]$ & $A[n/2] > A[n/2-1]$
Then p is equal to $n/2$.

In all the above cases, we can see that the sub array of A which is nearly of half size is pruning. This pruning happens recursively on the remaining subarrays. Hence running time

$T(n) = T(n/2) + O(1)$, After solving recurrences $T(n) = O(\log n)$

Solving recurrence: $T(n) = T(n/2) + O(1)$

$$T(n) = T(n/4) + 1 + 1$$
$$= T(n/8) + 1 + 1 + 1$$
$$= T(n/16) + 1 + 1 + 1 + 1$$
$$\vdots$$
$$= T(n/2^K) + K$$

$$n/2^K = 1 \Rightarrow 2^K = n \Rightarrow \boxed{K = \log n}$$

---

problem 2:

If the input numbers are divided into the groups of seven. As prof said in the class, we will get

time: $T(n) = T(n/7) + T(5n/7) + n$

To prove whether it is working in a $O(n)$ running time we use given and verification method.

Given: $T(n) = O(n)$
$T(n) \leq Cn$

Assume : $T(n/7) \leq c \cdot \frac{n}{7}$

$T(5n/7) \leq c \cdot \frac{5n}{7}$

Then we will get

$$T(n) \leq c \cdot \frac{n}{7} + c \cdot \frac{5n}{7} + n$$

$$T(n) \leq c \cdot \frac{6n}{7} + n \overset{??}{\leq} cn$$

Now, to find c and n_0 st $T(n) \leq cn$ for $n \geq n_0$.
To prove $T(n) \leq cn$, we will prove
$c \cdot \frac{6n}{7} + n \leq cn$.

$n \leq \frac{c}{7} n$, if $c = 7$ & $n_0 = 1$
then $n \leq \frac{c}{7} n$.

Hence proved and our guess $T(n) = O(n)$ is correct problem 3:

Given : No. of wells = n

$p_i = (x_i, y_i) \rightarrow$ coordinates of wells.

Output: optimal location for the main pipeline.

(or) y-coordinate of mainpipeline.

To find the optimal location of the main pipeline, we have to find the medians of y-coordinates of ~~spo~~ wells. The x-coordinates are irrelevant, so we are considering only y-coordinates.

① If n is even and the pipeline passes ~~through~~ between two oil wells whose coordinates are lower and upper medians. If we move the pipeline ~~vertically~~ of distance 'd', then $n/2$ of wells become 'd' farther from pipeline and $n/2$ of wells become 'd' closer from pipeline. Then sum after moving the pipeline is

$$S' = S + \frac{dn}{2} - \frac{dn}{2} = S.$$

So, when n is even, an optimal placement of pipeline is anywhere on or between two medians.

② If n is odd and the pipeline goes through the oil well, whose y-coordinate is median. Now, if we move the pipeline by a distance 'd'.

ⓐ All oil wells below median becomes 'd' units farther from pipeline = atleast $(n+1)/2$ ~~pipelines~~ oilwells.

ⓑ All oil wells above median becomes 'd' units closer to pipeline = atmost $(n-1)/2$ oilwells

So, sum of pipelines distance after moving is

$$S1 = S + d(n+1)/2 - d(n-1)/2 = S+d > S.$$

Therefore, moving the pipeline up from the oil well at the median increases total spur length. and moving the pipeline down from the median also increases total spur length.

So the ~~p~~ location of pipeline is on Median.

~~A~~

③ If n is even and pipeline goes through the oil well whose y-cordinate is upper median, If we increase y-coordinate

* oilwells below upper medians = $n/2 + 1$ (d units farther)
* oilwells above upper medians = $n/2 - 1$ (d units closer)

So, $S1 = S + d(n/2+1) - d(n/2 - 1)$

$= S + 2d > S$

So, moving pipeline up from oil well at upper median increases total spur length. Similarly pipeline going through oil well whose y-coordinate is lower median, then spur length increases.

Finally, as we know we are looking for the median, we can use linear-time median finding algorithm to compute an optimal location for main pipeline which is taking $O(n)$ time.

problem 4:

Input: an array $A[1 \ldots n]$

Output: find k_i-th smallest number

(a) Time: $O(n \log n)$

First, sort the elements of an array A.
The time for sorting the elements using mergesort is $O(n \log n)$.
We can find the k_ith smallest number by scanning the sorted list.
It takes $O(n \log n)$ time.

(b) Time: $O(nm)$

If we use linear-time selection algorithm to find the k_i-th smallest number of A for each $1 \le i \le m$, then it takes $O(nm)$ time.

(c) Time: $O(n \log m)$

By using the linear-time selection algorithm and divide and conquer technique, we can achieve this $O(n \log m)$ time.

Let $a[i]$ be the k_i-th smallest number of A.
Our main aim is to find ~~[illegible]~~.
$a[1], a[2] \ldots a[m]$

First, find $a[m/2]$ by using selection algorithm.
So, A_1 = elements smaller than $a[m/2]$
A_2 = elements larger than $a[m/2]$

Second, compute A_1 and A_2 in linear time by comparing each element of A with $a[m/2]$.
then, $A_1 = a[1], a[2] ---, a[m/2 - 1]$
$A_2 = a[m/2 + 1], ---- , a[m]$

Third, continue to ~~comp~~ find $a[1], --- a[m/2 - 1]$ in A_1 recursively and find $a[m/2 + 1] --- a[m]$ in A_2 recursively. for each $1 \le i \le m/2 - 1$, $a[i]$ is still the k_i-th smallest number in A_1 and for each $m/2 - 1 \le i \le m$, $a[i]$ is the $k_i - (|A_1| + 1)$-th smallest number in A_2.

So, The algorithm has $O(\log m)$ levels of recursive steps and each level takes $O(n)$ time in total. So, the total time is $O(n \log m)$.