CSCI - 605 Foundations of Algorithms HW - 02

Authors - Sagar Kukreja, Rohan Shiroor

Problem 1:

a) $T(n) \leq \begin{cases} O(1), if\ n = 1 \\ 2\,T(floor(n/2)) + O(1),\ for\ n \geq 2 \end{cases}$

b) Using telescoping we can find the bound for $T(n)$,

$$
\begin{aligned}
T(n) &\leq 2T(n/2) + d \\
&\leq 2(\,2T(n/4) + d) + d \\
&\leq 4T(n/4) + 3d \\
&\leq 4(\,2T(n/8) + d) + 3d \\
&\leq 8T(n/8) + 7d \\
&. \\
&. \\
&. \\
&. \\
&\leq 2^kT(n/2^k) + (2^k - 1)d
\end{aligned}
$$

It will go on till $2^k = n, so\ that\ T(n/2^k)\ for\ n = 2^k\ is\ O(1)$

$\equiv k = log\, n$

Hence, $T(n) \leq n + (n-1)d$

$T(n) = O(n)$

c) lstreak = maximum consecutive occurrences of any element in left array (0-middle)
rstreak = maximum consecutive occurrences of any element in right array (middle+1, n-1)
maxstreak = maximum consecutive occurrences of any element in whole array

Problem 2:

a) $T(n) = 4T(n/2) + n^2$
As per Master's Theorem,
a = 4 , b = 2, $f(n) = n^2$
$log_2 4 = 2$, hence $n^{log_b a} = n^2$
n^2 vs $f(n) = n^2$
As, $f(n) = \theta(n^{log_b a})$, so case 2 of theorem holds .
Hence, $T(n) = \theta(n^2\, log\, n)$.

b) $T(n) = 2T(n/2) + \sqrt{n}$

As per Master's Theorem,

a = 2 , b = 2, $f(n) = \sqrt{n}$

$log_2 2 = 1$, hence $n^{log_b a} = n$

n vs $f(n) = \sqrt{n}$

As, $f(n) = O\ (n^{log_b a - \varepsilon})$, so case 1 of theorem holds .

Hence, $T(n) = \theta(n)$.

c) $T(n) = 7T(n/3) + n^2$

As per Master's Theorem,

a = 7 , b = 3, $f(n) = n^2$

$log_3 7 = 1.77$, hence $n^{log_b a} = n^{1.77}$

$n^{1.77}$vs $f(n) = n^2$

As, $f(n) = \Omega(n^{log_b a + \varepsilon})$, so case 3 of theorem holds .

Hence, $T(n) = \theta(f(n)) = \theta(n^2)$.

d) $T(n) = 2T(n/8) + n^{1/3}$

As per Master's Theorem,

a = 2 , b = 8, $f(n) = n^{1/3}$

$log_8 2 = 1/3$, hence $n^{log_b a} = n^{1/3}$

$n^{1/3}$vs $f(n) = n^{1/3}$

As, $f(n) = \theta(n^{log_b a})$, so case 2 of theorem holds .

Hence, $T(n) = \theta(n^{1/3}\ log\ n)$.

Problem 3:

Algorithm: For this problem, number of swaps required to bring entire class in order can be thought of as problem to count number of inversions. For this we use merge sort.

1) We first sort the array based on age of the persons using merge sort. This takes $O(n\ log\ n)$and returns the swaps needed to bring them in order.

   After step -1, the array is now sorted with teacher in middle and 7 years old on left side of teacher and 8 years old on right side of teacher.

2) We now divide the array into two parts: 1) array containing 7 year old students 2)array containing 8 year old student.

   And apply merge sort on both arrays individually to sort 7 years in ascending order of their height and 8 year old in descending order of their height.

   This takes $2O(n\ log\ n)$

3) And then we finally merge these two arrays in $O(n)$

Complexity Analysis:

Total Time complexity = $O(n \log n)$ + $2O(n \log n)$ + $O(n)$
= $O(n \log n)$

Proof of correctness:

This problem can be thought of as a counting inversion problem which states how far is the array from being sorted. This is solved using merge sort and counting the swaps required to sort the array which takes $O(n \log n)$

Here, we also apply merge sort in a sequential way to count the number of swaps and outputs them.

Problem 4:

For the problem of finding max number of pair of points that can be aligned by a line, we first try to find a line between all sets of points by finding the slope and the intercept. That operation is $O(n^2)$ operation because for n pair of points there are n^2lines that can be drawn between them. Then we sort those lines using merge sort which is $O(n^2 logn)$ operation as the size of data becomes n^2. So that is $O(n^2 logn^2)$ which is nothing but $O(2n^2 logn)$ which is $O(n^2 logn)$. Then we traverse through the data and find the common line with the max points which get aligned.This operation is again $O(n^2)$. So the total cost of running the algorithm is $O(n^2 logn) + O(n^2) + O(n^2)$. As $O(n^2 logn)$ is the dominating term the complexity of our algorithm is $O(n^2 logn)$.

Problem 5

Algorithm: For this problem, We use k-select algorithm to solve our problem.
We find pivot element using Improved Select algorithm.

1) We find the pivot element by dividing the array into groups of 5 and then come up with a pivot element. This step takes $O(n)$.

2) We then apply k-select algorithm and divide the array into 3 parts, large, equal, and smaller array based on pivot element.

3) If the equal array has n/2 or n/3 elements respectively, we terminate the algorithm with YES else we repeat step 1, 2 and 3 on large array or small array depending on the length of these two arrays, i.e. if their length is > n/2 or n/3.

Complexity Analysis:

The above algorithm is same as improved k-select algorithm which has complexity of $O(n)$. Our algorithm also does the same thing, except the conditions on which it recurse is different.

Total Time complexity = $O(n)$