IIIT Vadodara
CS203: Mid-Sem Remote Exam

January 15, 2022

Problem 1

[Marks:-4] Sort the functions in increasing order of asymptotic (big-O) complexity with explanation.
(a)$f_1 = n^{\sqrt{n}}$
(b) $f_2 = 2^n$
(c) $f_3 = 10000000n$
(d) $f_4 = \sum_{i=1}^{n}(i+1)$

Solution:- $f_4 = \sum_{i=1}^{n}(i+1) = \frac{n((n+1)+2)}{2} = \mathcal{O}(n^2)$
$f_1 = n^{\sqrt{n}} = (2^{lg_2\ n})^{\sqrt{n}}$ [Using property(a=b$^{lg_b a}$)]
=(2$^{\sqrt{n} lg_2\ n}$)
$f_2 = 2^n$

Function in increasing order:-f_3, f_4, f_1, f_2

Problem 2

[Marks:-3+3] Consider a recurrence relation $T(n) = \sqrt{n}T(\sqrt{n}) + n$. Prove that $\Theta(nlogn)$ and $\Theta(n)$ are NOT suitable bounds as per definition of Θ bound [Hint: You can use substitution method].

Solution:- Using Substitution method. Let $\theta(nlogn)$ is your bound. You had to prove that $nlogn$ is both your upper and lower bound.
First check for the upper bound.

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + n \\ &\leq \sqrt{n}c\sqrt{n}log\sqrt{n} + n \\ &= n.c.log\sqrt{n} + n \\ = n.c.\frac{1}{2}logn + n &\leq cnlogn \end{aligned}$$

The last inequality assumes only that $1 \leq c.\frac{1}{2}.logn$. This is correct if n is sufficiently large and for any constant c, no matter how small. The above proof says that our guess is correct for the upper bound.
Now, let us prove the lower bound for this recurrence.

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + n \\ &\geq \sqrt{n}k\sqrt{n}log\sqrt{n} + n \\ &= n.k.log\sqrt{n} + n \\ = n.k.\frac{1}{2}logn + n &\geq knlogn \end{aligned}$$

The last inequality assumes only that $1 \leq k.\frac{1}{2}.logn$. This is incorrect if n is sufficiently large and for any constant k. This guess is incorrect for the lower bound. Thus, we understood the $\theta(nlogn)$ is too big and not the correct θ bound.

Now assume $\theta(n)$ is your upper bound.

$$\begin{aligned} T(n) &= \sqrt{n}T(\sqrt{n}) + n \\ &\leq \sqrt{n}k\sqrt{n}\sqrt{n} + n \\ &= n.c + n \\ n(c+1) &\nleq cn \end{aligned}$$

Lower bound is easy to prove directly, that is true. Thus it seems $\theta(n)$ is too small and $\theta(nlogn)$ is too big. This proves our statement.

Problem 3

(**Marks:- 3+2+2**) Write the "Bottom-up iterative algorithm" that will find the length for Longest Common Subsequence (LCS) which contains only vowels.

- Show your algorithmic steps to above problem. Also write the algorithm that prints the LCS.
- Give justification for Time and Space complexity of your solution.
- Show all the steps of your solution as input strings X as your First Name and Y as your Second Name. Example: $X_1 = "RAKESH"$ and $Y_1 = "SINGH"$ then LCS would be ZERO. $X_2 = "RAKESH"$ and $Y_2 = "RAMESH"$ then LCS would be $"AE"$.

Solution:-

// function to find the length of longest common subsequence which contains all vowel characters

```
isVowel(char ch)
{
if (ch ==' a'||ch ==' e'||ch ==' i'||ch ==' o'||ch ==' u')
return true;
else
return false;
}

lcs(X, Y, m, n)
{
L[m + 1][n + 1];
i, j;
```

// Following steps build L[m+1][n+1] in bottom up fashion. Note:- that L[i][j] contains length of LCS of X[0..i-1] and Y[0..j-1]

```
for (i = 0; i <= m; i++) {
for (j = 0; j <= n; j++) {
if (i == 0 || j == 0)
L[i][j] = 0;

else if ((X[i - 1] == Y[j - 1])  isVowel(X[i - 1]))
L[i][j] = L[i - 1][j - 1] + 1;

else
L[i][j] = max(L[i - 1][j], L[i][j - 1]);
}
}
```

// L[m][n] contains length of LCS for X[0..n-1] and Y[0..m-1] which contains all vowel characters return L[m][n]; }
Time complexity: $O(mn)$
Space Complexity $O(mn)$
Example needs to be checked individually on his/her name.

Problem 4

(Marks:- 2+2+4) We have given a set of numbers in an array which is first increasing and then decreasing. For example:{3, 10, 20, 38, 120, 100, 60, 30, 7}. Maximum is 120.
(a)Write first the naive algorithm and the complexity which finds the maximum value from the list.

Solution:- We can traverse the array and keep track of maximum and element. And finally return the maximum element. **Simply check Linear search algo has been written or not.**
Time Complexity : $O(n)$

(b) Can you write the algorithm which can improve the complexity further?

Solution:- We can modify the standard Binary Search algorithm for the given type of arrays.
i) If the mid element is greater than both of its adjacent elements, then mid is the maximum.

ii) If mid element is greater than its next element and smaller than the previous element then maximum lies on left side of mid.

Time Complexity: $O(logn)$

(c) Show the steps of insertion of the above elements from array (one at a time) in the Heap. You can make either Max-Heap or Min-Heap.
Solution:- You had to make either the max-heap or min-heap on the above input and show the steps.

Problem 5

(Marks:- 4+2+2+2) An array $A[1\ldots n]$ is said to have a *majority element* if more than half of its entries are the same. Given an array, the task are following:

1. Design an efficient divide and conquer algorithm to tell whether the array has a *majority element*, and, if so, to find that element.
2. Give the recurrence relation of your solution.
3. Find the complexity of your solution using Recursion Tree method.
4. Give an example of your working solution to any input of your choice.

Solution: Many have done this question, so I am not writing this solution.