CSE 551 Assignment 2

14th September, 2022

Submission Instructions: Deadline is **11:59pm on 09/20/2022**. Late submissions will be penalized, therefore please ensure that you submit (file upload is completed) before the deadline. Additionally, you can download the submitted file to verify if the file was uploaded correctly. **Please TYPE UP YOUR SOLUTIONS and submit a PDF** electronically, via *Canvas*. Furthermore, please note that the graders will grade 2 out of the 4 questions randomly. Therefore, if the grader decides to check questions 1 and 4,and you haven't answered question 4, you'll lose points for question 4. Hence, please answer all the questions.

1. The Fibonacci series can be computed as follows,

$$F(n) = F(n-1) + F(n-2) \tag{1}$$

In class, we showed how this can be done in $\mathcal{O}(log\ n)$ computation time. Now suppose that the definition is changed in the following way,

$$F'(n) = F'(n-1) + F'(n-2) + F'(n-3) + F'(n-4) \tag{2}$$

Can $F'(n)$ be computed in $\mathcal{O}(log\ n)$? If yes, please show how it can be done. If no, show a counterexample where this fails. Please provide your rationale for both. Assume that $F'(0) = 0, F'(1) = 1, F'(2) = 1, F'(3) = 1$. **[25 Points]**.

ANS:

Given,

$F'(n) = F'(n-1) + F'(n-2) + F'(n-3) + F'(n-4)$

Therefore,

$[F'(n-3)\ \ F'(n-2)\ \ F'(n-1)\ \ F'(n)]$

$= [F'(n-3)\ \ F'(n-2)\ \ F'(n-1)\ \ F'(n-1)+F'(n-2)+F'(n-3)+F'(n-4)]$

$= [F'(n-4)\ \ F'(n-3)\ \ F'(n-2)\ \ F'(n-1)]$ x A where

$$A = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$= [F'(n-5)\ \ F'(n-4)\ \ F'(n-3)\ \ F'(n-2)]$ x A^2

. . . .

$= [F'(0)\ \ F'(1)\ \ F'(2)\ \ F'(3)]$ x A^{n-3}

$= [0\ \ 1\ \ 1\ \ 1]$ x A^{n-3}

Since the dimensions of $A = (4 \times 4)$,i.e., a constant value, the matrix multiplication A^{n-3} can be done in O(log n) time (**Refer to class slides**).

2. In class we have discussed the Quick Sort algorithm. Answer the following questions regarding quick sort.

   - When does the worst case for Quick Sort occur?
     Ans: The worst case for quick sort occurs when every time the pivot element picked would be the largest or smallest element in the array. This would make the array into only one part of size $n-1$.

   - What would be the worst case recurrence for Quick Sort? Solve the recurrence and get its time complexity in worst case.
     Ans: The worst case recurrence for quick sort is $T(n) = T(n-1) + T(1) + O(n)$. Here is the solution:
     $T(n) = T(n-1) + T(1) + O(n) and T(1) = O(1)$
     $T(n) = T(n-1) + O(n)$
     $T(n) = [T(n-2) + O(n-1)] + O(n)$
     $= T(n-3) + (n-2) + (n-1) + O(n)$
     .
     .
     .
     $= T(1) + (n-(n-2)) + ... + (n)$
     $= 1 + 2 + 3 + ... + n = n(n-1)/2$
     $T(n) = O(n^2)$

   - What would be the best case recurrence for Quick Sort? Solve the recurrence and get its time complexity in best case.
     Ans: In best case of quick sort every time the array would be split into two equal halves. So the recurrence would be $T(n) = 2T(n/2) + O(n)$. We can solve the recurrence using Master theorem and we get the time complexity of $O(nlogn)$

- Suppose you have an algorithm which can give the median of a set on numbers in O(n) time. How can this be used to improve the worst case time complexity of Quick sort?

  Ans: When we have a algorithm to find median in O(n) time, in every iteration of the quick sort we can use this to get the median of the numbers ans use it as the pivot element. This would make the recurrence $T(n) = 2T(n/2) + O(n)$ solving which we get $T(n) = O(nlogn)$

- In the execution of Quick sort suppose every time the $n/5^{th}$ smallest element is picked as the pivot element among n elements, what would be the recurrence for quick sort? What would be its time complexity?

  Ans: If the n/5th element is picked as the pivot element after one split one bucket has $n/5$ elements and other has $4n/5$ elements.

  The recurrence becomes $T(n) = T(n/5) + T(4n/5) + O(n)$. We can use recurrence tree method to solve this. The sum of nodes in each level would be n and the height of the tree would be at most $log_{5/4}n$. So the time complexity would be $\leq nlog_{5/4}n = O(nlogn)$.

3. If k is a non-negative constant, then prove that the recurrence: **[25 points]**

$$T(n) = k \text{ , for } n = 1 \text{ and}$$
$$T(n) = 3T(\tfrac{n}{2}) + kn, \text{ for } n > 1$$

has the following solution (for n a power of 2):

$$T(n) = 3kn^{log_2 3} - 2kn$$

Ans: Here is the solution to the recurrence.

$T(n) = 3T(n/2) + kn$
$T(n) = 3[3T(n/4) + kn/2] + kn$
$= 9T(n/4) + 3kn/2 + kn$
$= 9[3T(n/8) + kn/4] + 3kn/2 + kn$
$= 27T(n/8) + 9kn/4 + 3kn/2 + kn$

After i levels we get $T(n) = 3^i T(n/2^i) + kn + \Sigma_{j=0}^{i-1}(3/2)^j$

Since there are $log_2 n$ levels substituting i we get

$T(n) = 3^{logn} * k + 2kn * ((3/2)^{logn} - 1)$ which on simplifying gives $T(n) = 3kn^{log_2 3} - 2kn$

4. Design an algorithm to compute the 2nd smallest number in an unordered (unsorted) sequence of numbers $\{a_1, a_2, ..., a_n\}$ in $n + \lceil log_2(n) \rceil - 2$ comparisons in the worst case. If you think such an algorithm can be designed,

then show how it can be done. If your answer is no, then explain why it cannot be done. [**25 points**]

ANS:

Let us consider a tournament. Assume that you have a list of n players (denoted by unique integers). One tournament rule could be the following: the smallest number of two numbers, wins. Now, we can use the Find-MinMax algorithm taught in class to find out the smallest number in n - 1 comparisons. Realize that, in such a tournament the smallest number (best player) is guaranteed to play the second smallest number (second best player), if we follow our rule. Therefore, once we have discovered the best player, we need to examine *the subtree from where the best player originated*. This is because, the best player could have played the second best player at any level in the tournament (and not just the final round). This analysis takes $\lceil log_2(n) \rceil - 1$ comparisons. Therefore, in total, we have at most $n + \lceil log_2(n) \rceil - 2$ comparisons to find the 2nd second best player in the tournament.

Another similar way to solve this to construct a min heap. Construction of the min heap would take $O(n)$. This would entail that the minimum value in the sequence is present in the root. Remove the minimum and replace it with the rightmost element in the tree, following level order traversal. Heapify the new tree and you will have the second smallest element as the root now. Heapify takes $O(logn)$ time. Hence the total time to find the second smallest element is less than $n + \lceil log_2(n) \rceil - 2$.