CSE 450 Assignment 2

14^{th} 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]**.

Solution:

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. Find the closed form solution of the following recurrence relation and show the closed form solution in O, Θ, or Ω notation. **[25 points]**

$$T(n) = T(n-1) + \tfrac{n}{2},\ T(1) = 1$$

Solution:

$T(n-1) = T(n-2) + \frac{n-1}{2}$,

After n-2 tries

$T(n-(n-2)) = T(n-(n-2)-1) + \frac{n-(n-2)}{2}$

$= \mathrm{T}(1) + \frac{2}{2} = 1 + 1$

$T(2) = 1 + 1 = 2$

$\mathrm{T(n)} = 1 + \frac{n}{2} + \frac{n-1}{2} + \frac{n-2}{2} + \frac{n-3}{2} + \ldots\ldots + 1$

$= \frac{3}{2} + \frac{n}{2} + \frac{n-1}{2} + \frac{n-2}{2} + \frac{n-3}{2} + \ldots\ldots + \frac{1}{2}$

$= \frac{3}{2} + \frac{1}{2}$ (n + (n-1) + (n-2) + ... + 1)

$= \frac{3}{2} + \frac{1}{2} \times \frac{n(n+1)}{2}$

$= \frac{3}{2} + \frac{n(n+1)}{4}$

Therefore, the answer is $\Theta(n^2)$

3. You are to organize a tournament involving n competitors. Each competitor must play exactly once against each of his/her opponents. Moreover, each competitor must play exactly one match every day. If n is a power of 2, design a Divide-and-Conquer based algorithm to construct a time table allowing the tournament to be finished in $n-1$ days. Prove that the algorithm indeed completes the tournaments in $n-1$ days **[25 points]**

Solution:

$T(n) = T(\frac{n}{2}) + \frac{n}{2}$, T(2) = 1

$T(\frac{n}{2}) = T(\frac{n}{4}) + \frac{n}{4}$

After x-1 tries

$T(\frac{n}{2^{x-1}}) = T(\frac{n}{2^x}) + \frac{n}{2^x}$

$\frac{n}{2^x} = 1$

n = 2^x

x = $log_2 n$

$T(n) = 1 + \frac{n}{2} + \frac{n}{4} + \text{.....} + \frac{n}{2^x}$

$= 1 + \frac{n}{2}\ (\ 1 + \frac{1}{2} + \text{.....} + \frac{1}{2^{x-1}}\ )$

$= 1 + \frac{n}{2} \times (1 - (\frac{1}{2})^{x-1}) \div \frac{1}{2}$

$= 1 + n\ (\ 1 - \frac{1}{2^{x-1}})$ $1/2^{x-1} = 1/2^{\log_2 n - 1} = 2/n$

$= 1 + n - 2$

$= n - 1$

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]**

Solution:

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 $n + \lceil log_2(n) \rceil - 2$.