CSE 551 Assignment 2

February 24, 2021

Submission Instructions: Deadline is **11:59pm on 02/21**. Late sub-missions 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. Submit your answers electronically, in a single PDF, via *Canvas*. **Please type up the answers and keep in mind that we'll be checking for plagiarism**.
Furthermore, please note that the graders will grade 3 out of the 6 questions randomly. Therefore, if the grader decides to check questions 1, 2 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. Consider the following problem. You're given an array A consisting of n integers $A[1], A[2], ..., A[n]$. You'd like to output a two-dimensional $n \times n$ array B in which $B[i, j] (for\ i < j)$ contains the sum of array entries $A[i]$ through $A[j]$, i.e., the sum $A[i] + A[i+1] + ... + A[j]$. (The value of array entry $B[i, j]$ is left unspecified whenever $i \geq j$, so it doesn't matter what is output for these values.)

A simple algorithm to solve this problem illustrated in Algorithm 1:

```
Data: Input $A$ and $n$
Result: Return $B$
for $i$ = 1, 2, ..., $n$ do
    for $j$ = $i$ + 1, $i$ + 2, ..., $n$ - 1 do
        Add up array entries $A[i]$ through $A[j]$
        Store the result in $B[i, j]$
    end
end
Return $B$;
```

Algorithm 1: Algorithm for Q2

```
B[i][j] ← 0, $\forall i, j \leq n$
for $i$ = 1, 2, ..., $n$ do
    sum ← A[i]
    for $j$ = $i$ + 1, $i$ + 2, ..., $n$ - 1 do
        sum ← sum + A[j]
        B[i][j] ← sum
    end
end
Return $B$;
```

Algorithm 2: Improved Algorithm for Q3

- For some function f that you should choose, give a bound of the form $\mathcal{O}(f(n))$ on the running time of this algorithm on an input of size n (i.e., a bound on the number of operations performed by the algorithm). **[8 Points]**.

  ANS: The first two for loops each iterate n and $n-1$ times respectively. Therefore, we have n^2 computation for the for loops. The computation of the sum from $A[i]$ to $A[j]$, can take atmost n iterations. Storing the result takes constant time. Therefore, we can select a $f(n) = n^3$ and the upper bound is $\mathcal{O}(n^3)$.

- For this same function f, show that the running time of the algorithm on an input of size n is also $\Omega(f(n))$. (This shows an asymptotically tight bound of $\Theta(f(n))$ on the running time.) **[8 Points]**.

  ANS: Let us count the number of additions, suppose $i = 1$,
  when $j = 2$, we have 1 addition
  when $j = 3$, we have 2 additions
  when $j = 4$, we have 3 additions
  when $j = n$, we have $n - 1$ additions

  Therefore, the number of additions taking place $= 1 + 2 + (n-1) = n \times (n-1)/2$

  Now, i iterates n times, therefore select $f(n) = n \times n \times (n-1)/2 = (n^3 - n^2)/2$. You can now easily show $\Theta(n^3)$.

- Although the algorithm you analyzed in parts (a) and (b) is most natural way to solve problem-after all, it just iterates through the relevant entries of the array B, filling in a value for each - it contains some highly unnecessary sources of inefficiency. Give a different algorithm to solve this problem, with an asymptotically better running time. In other words, you should design an algorithm with running time $\mathcal{O}(g(n))$, where $lim_{n\to\infty} g(n)/f(n) = 0$. **[9 Points]**.

  ANS: One modification would be to pre-compute the values in the outer loop, to avoid re-computation. Consider the algorithm 2:
  This improves the time complexity to $O(n^2)$.

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

4. Let $n = 2p$, $V = (v_1, ..., v_n), W = (w_1, ..., w_n)$. Then we can compute the vector product VW by the formula:
$\sum_{1 \le i \le p} (v_{2i-1} + w_{2i}) \times (v_{2i} + w_{2i-1}) - \sum_{1 \le i \le p} v_{2i-1} \times v_{2i} - \sum_{1 \le i \le p} w_{2i-1} \times w_{2i}$

which requires $3n/2$ multiplications. Show how to use this formula for the multiplication of two $n \times n$ matrices giving a method which requires $n^3/2 + n^2$ multiplications rather than the usual n^3 multiplications. **[25 points]**

ANS:

Multiplications of two $n \times n$ matrices to compute the resultant $n \times n$ matrix C involves product of n^2 vector multiplications. The matrix A may be viewed as made up of n row vectors $V_1, V_2, ..., V_n$ and similarly B may be viewed as made up of n column vectors $W_1, W_2, ..., W_n$. The element C_{11} of C is $V_1 W_1$, C_{12} is $V_1 W_2$ and so on.

In order to compute C_{11} we will require 3p multiplications. In order to compute $C_{12}, ..., C_{1n}$ will require only 2p multiplications, as the second term in the formula involving V_1 is already computed while computing C_{11}.

Similarly, in order to compute $C_{21}, ..., C_{n1}$ we will require only 2p multiplications, as the third term in the formula involving W_1 is already computed while computing C_{11}.

In order to compute C_{22} to C_{2n} and C_{32} to C_{3n}, ..., C_{n2} to C_{nn}, $((n-1)^2)$ entries, only p multiplications will be needed as the second and the third terms of the formula need not be computed again. Thus the total number of multiplications that will be needed is:

$= 3p + (n-1) \times 2p + (n-1) \times 2p + (n-1)^2 \times p$

$= 3n/2 + (n-1) \times n + (n-1) \times n + (n-1)^2 \times n/2$

$= n^3/2 + n^2$

5. Find the solution to the following recurrence relation:

$T(n) = 16T(n/4) + n^2$,

for n a power of 4 (or, $n = 4^k$) and $n > 1$. **Show all your work. [25 points]**

$T(n) = 16T(n/4) + n^2$.

$$\begin{aligned}
T(n) &= n^2 + 16T(\frac{n}{4}) \\
&= n^2 + 16((\frac{n}{4})^2 + 16T(\frac{n}{4^2})) \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2T(\frac{n}{4^2}) \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2((\frac{n}{4^2})^2 + 16T(\frac{n}{4^3})) \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2(\frac{n}{4^2})^2 + 16^3T(\frac{n}{4^3}) \\
&\quad \ldots \\
&= n^2 + 16(\frac{n}{4})^2 + 16^2(\frac{n}{4^2})^2 + 16^3(\frac{n}{4^3})^2 + 16^4(\frac{n}{4^4})^2 + ... + 16^{\log_4 n}(\frac{n}{4^{\log_4 n}})^2 \\
&= \underbrace{n^2 + n^2 + n^2 + n^2 + n^2 + ... + n^2}_{\log_4 n+1} \\
&= n^2(\log_4 n + 1) \\
&= \Theta(n^2 \lg n)
\end{aligned}$$

6. Find the solution to the following recurrence relation:

   $T(1) = 1$
   $T(n) = 3^n T(n/2)$,

   for n a power of 2 (or, $n = 2^k$) and $n > 1$. **Show all your work. [25 points]**

   ANS:
   $T(n) = 3^n T(n/2)$
   $= 3^n * 3^{\frac{n}{2}} * T(n/4)$
   $= 3^n * 3^{\frac{n}{2}} * 3^{\frac{n}{4}} * T(n/8)$
   ....
   ....
   $= 3^n * 3^{\frac{n}{2}} * 3^{\frac{n}{4}} * ... * 3^{\frac{n}{n/2}} * T(n/n)$
   $= 3^n * 3^{\frac{n}{2}} * 3^{\frac{n}{4}} * ... * 3^2 * T(1)$
   $= 3^{n+\frac{n}{2}+\frac{n}{4}+...+2} * 1$
   For n a power of 2, we know: $n + \frac{n}{2} + \frac{n}{4} + ... + 2 + 1 = 2n - 1$
   $=> n + \frac{n}{2} + \frac{n}{4} + ... + 2 = 2n - 1 - 1 = 2n - 2$
   Therefore,
   $T(n) = 3^{2n-2} = \frac{3^{2n}}{9}$
   $T(n) \in O(3^{2n})$