Problem Set 3

Due date: Electronic submission of the pdf file of this homework is due on **2/10/2023 before 11:59pm** on ecampus.

Name: **Ayushri Jain**

Resources. Below resources were used for reference -

- https://www.calculator.net/log-calculator.html
- Cormen, Leiserson, Rivest, Stein: Introduction to Algorithms, 3rd edition, The MIT Press, 2009 (or 4th edition)
- https://opendsa-server.cs.vt.edu/ODSA/Books/CS3/html/RecurrenceIntro.html
- https://www.programiz.com/dsa/master-theorem

On my honor, as an Aggie, I have neither given nor received any unauthorized aid on any portion of the academic work included in this assignment. Furthermore, I have disclosed all resources (people, books, web sites, etc.) that have been used to prepare this homework.

Signature: Ajain

Problem 1 (20 points).

Solution. To use mathematical induction, we need to have a base case, hypothesis step and then induction step. Let's get our base case first. It is given that $T(2) = 2$, we can multiply right hand side by $\log_2 2$ since it evaluates to 1. So, $T(2) = 2\log_2 2$ is our base case.
Let's assume that if $n = 2^k$ for some integer $k > 0$ then $T(n) = n\log_2 n$ is true. Substituting the value of n, we get $T(2^k) = 2^k \log_2 2^k$.
Now we need to prove that this holds true for some $n = 2^{k+1}$. We know that $T(n) = 2T(n/2) + n$ if $n = 2^k$ for $k > 1$,
Substituting $n = 2^{k+1}$, we get
$T(2^{k+1}) = 2T(2^{k+1}/2) + 2^{k+1} = 2T(2^k) + 2^{k+1}$
Replacing $T(2^k)$ by the value which we got before, we get
$T(2^{k+1}) = 2 * 2^k \log_2 2^k + 2^{k+1} = 2^{k+1}\log_2 2^k + 2^{k+1} = 2^{k+1}(\log_2 2^k + 1)$
We can now replace 1 with $\log_2 2$ on right hand side, we will get $2^{k+1}(\log_2 2^k + \log_2 2)$. Using logarithm rule of addition, we can simplify it as $2^{k+1}(\log_2(2^k * 2)) = 2^{k+1}(\log_2(2^{k+1}))$.
Therefore our equation becomes -
$T(2^{k+1}) = 2^{k+1}(\log_2(2^{k+1}))$.
We have proved that our assumption holds true for $k + 1$, hence by mathematical induction we can say that the solution of the given recurrence relation is $T(n) = n\log_2 n$.

Problem 2 (20 points).

Solution. In the proposed recursive procedure for insertion sort, there are two steps. First is to sort a sub-array $A[1..n-1]$ and then insert $A[n]$ into the sorted array $A[1..n-1]$. We need a base case which is easy to derive. When we only have 1 element in array, then it is already sorted. So, $T(1) = 1 = \Theta(1)$ becomes our base case. For arrays which have more than n elements, we will recursively sort the first $n-1$ elements and then insert the n^{th} element into this sorted array. In worst case, inserting this element will take $\Theta(n)$ time because we will need to look into entire array to find the position for inserting the element. Assuming array is of size n, let T(n) denote running time for this version of insertion sort. Then the recurrence relation for T(n) can be expressed as:

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1, \\ T(n-1) + \Theta(n) & \text{if } n > 1 \end{cases}$$

Problem 3 (20 points).

Solution. We know that to multiply two matrices A and B of size 2×2 using divide and conquer matrix multiplication algorithm, we will need 8 multiplications and some more evaluations (additions). In that case, we write our recurrence relation as $T(n) = 8T(n/2) + \Theta(n^2)$ and $T(2) = \Theta(1)$ is our base case. When we solve it, we get the time complexity as $T(n) = \Theta(n^{\log_2 8}) = \Theta(n^3)$. So, in general we can say that in case of divide and conquer matrix multiplication algorithm if we have m sub problems with k multiplications each then time complexity can be written as $T(n) = \Theta(n^{\log_m k})$. Using the above logic, the time complexity for the 3 ways discovered by V. Pan can be calculated as -

68×68 matrix, 132464 multiplications :
$T(n) = \Theta(n^{\log_{68} 132464}) = \Theta(n^{2.7951284873614})$

70×70 matrix, 143640 multiplications :
$T(n) = \Theta(n^{\log_{70} 143640}) = \Theta(n^{2.7951226897483})$

72×72 matrix, 155424 multiplications :
$T(n) = \Theta(n^{\log_{72} 155424}) = \Theta(n^{2.7951473910934})$
Clearly, the best asymptotic running time among the 3 methods is for 70×70 matrix $= \Theta(n^{2.7951226897483})$.
In Strassen's algorithm, 7 multiplications are used and its time complexity is $T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.8073549220576})$. As we can see $\Theta(n^{2.8073549220576})$ is bigger than $\Theta(n^{2.7951226897483})$, therefore the method discovered by V. Pan for 70×70 matrices is better than Strassen's algorithm.

Problem 4 (20 points)**.**

Solution. If we go with the basic process of multiplying two complex numbers, we will end up doing 4 multiplications : $(a+bi)(c+di) = ac+adi+bci+bdi^2 = ac + i(ad + bc) - bd = (ac - bd) + i(ad + bc)$. Multiplying two complex numbers using 3 multiplications can be tricky. However, it is clear that the real component will contain $ac - bd$ and imaginary will contain $ad + bc$. If we do $(a+b)(c+d)$, we get $ac+ad+bc+bd = X$ using 1 multiplication. Looking at our real component, let's do 2 more multiplications, $ac = Y$ and $bd = Z$. We can combine these 3 using addition and subtraction to get our required values. Our real component will then become $Y - Z = ac - bd$ and imaginary component will become $X - Y - Z = ac+ad+bc+bd-ac-bd = ad+bc$. Below is the algorithm:

given input : a, b, c, d

$X = (a + b)(c + d)$ # 1^{st} multiplication

$Y = ac$ # 2^{nd} multiplication

$Z = bd$ # 3^{rd} multiplication

real $= Y - Z$

imaginary $= X - Y - Z$

return real & imaginary

Problem 5 (20 points)**.**

Solution. According to master method, if there is a recurrence relation $T(n) = aT(n/b) + f(n)$ such that $a >= 1, b > 1$ and $f(n)$ is eventually positive then, the time complexity can be written in simpler terms for 3 cases :

$$T(n) = \begin{cases} \Theta(n^{\log_b a}) & \text{if } f(n) = O(n^{log_b a - \epsilon}), \\ \Theta(n^{\log_b a} \log n) & \text{if } f(n) = \Theta(n^{log_b a}), \\ \Theta(f(n)) & \text{if } f(n) = \Omega(n^{log_b a + \epsilon}) \text{ and } f(n) \text{ is regular,} \end{cases}$$

Given the recurrence relation $T(n) = T(n/2) + \Theta(1)$, $a = 1$, $b = 2$ and $f(n) = \Theta(1)$, $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$.
$f(n) = \Theta(1) = \Theta(n^0) = \Theta(n^{\log_2 1})$. Therefore, 2^{nd} case of master method is applicable here. Thus, $T(n) = \Theta(n^{\log_b a} \log n) = \Theta(n^{\log_2 1} \log n) = \Theta(\log n)$. Since base of logarithm can be anything as it won't affect the result, so we can finally conclude that $T(n) = \Theta(\log_2 n) = \Theta(\lg n)$ ($\log_2 x$ is written as $\lg x$).