CS 201: Algorithms, Mid Semester Examination, 20 September 2022

Note: Answer all of the questions. **Make you bound as tight as possible.** Maximum time 2 Hours. Total Marks 50.

1. The master method provides a "cookbook" method for solving recurrences of the form

$$T(n) = aT\left(\frac{n}{b}\right) + f(n)$$

Why the value of b should be greater than 1. [2]

2. Solve the following recurrence relations. [$2 \times 3 = 6$]

   (a) $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$

   (b) $T(n) = 4T\left(\frac{n}{3}\right) + n \log n$

   (c) $T(n) = T\left(\frac{n}{3}\right) + T\left(\frac{2n}{3}\right) + n$

3. Consider there are N positive integers which are stored in an array. We want to sort this array in ascending order using Quick sort. Assume median element is selected as a pivot in Quick sort each time. Let the time complexity to find the median element in an array of size N be $\mathcal{O}(N \log N)$. Compute the time complexity of this type of Quick sort. [4]

4. In merge Sort, we need to merge two sorted arrays. Let the time complexity to merge two sorted arrays of size $N/2$ is $\Theta(\sqrt{N})$. Obtain the time complexity of this kind of merge sort. [4]

5. Given an array A of size N. Let all the elements of A be negative. Write a $\mathcal{O}(N)$ pseudo code to find the sub-array with maximum sum. You need to find the following three pieces of information: [5]

   - Maximum sum
   - Start index of the sub-array
   - End index of the sub-array

6. Given a matrix A. Let the start and end index of row is `lowR` and `highR` respectively. Similarly, let the start and end index of column is `lowC` and `highC` respectively. Assume that $\texttt{highR} - \texttt{lowR} + 1 = \texttt{highC} - \texttt{lowC} + 1 = N = 2^k$ where $k \in Z^+$. Write the pseudo code to divide this matrix A into four sub-matrices A_1, A_2, A_3, A_4 of size $2^{k-1} \times 2^{k-1}$ such that combining these four sub-matrices gives you the original matrix A. Also, obtain the time complexity of this process. [5]

7. You have been given four matrices A_1, A_2, A_3 and A_4, as shown below. Parenthesize $A_1 \times A_2 \times A_3 \times A_4$ using the tabular format of dynamic programming such that the number of scalar multiplications to multiply these four matrices is minimum. *Note:* Pseudo-code/Algorithm is not required, and only the parenthesis pattern is needed, not the final result of matrix multiplication. [5]

$$A_1 = \begin{bmatrix} 1 & 2 \\ 2 & 4 \\ 1 & 3 \end{bmatrix} \quad A_2 = \begin{bmatrix} 1 & 2 & 2 & 1 \\ 2 & 4 & 5 & 3 \end{bmatrix} \quad A_3 = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 1 \\ 4 & 2 \end{bmatrix} \quad A_4 = \begin{bmatrix} 1 & 5 & 1 & 1 & 2 \\ 2 & 6 & 1 & 1 & 2 \end{bmatrix}$$

8. Sachin wants to travel from Bongora to Kolkata via road in his Tata Nano car, which has a fuel capacity of C litres. Assume that there is a single path between these two cities. Sachin has prepared a list of petrol pumps on his path with their distance from Bongora. The set of petrol pumps are represented with $P = \{p_1, p_2, \ldots, p_n\}$ and their corresponding distance set with $D = \{d_1, d_2, \ldots, d_n\}$. Note that the list contains petrol pumps at starting point p_1 with $d_1 = 0$ and the ending point p_n also. Sachin wants to minimize the number of stops (for filling petrol) during his journey.

   (a) Design an efficient algorithm to solve Sachin's problem by selecting the minimum number of petrol pumps. Your algorithm should have a time complexity of $\mathcal{O}(n \log n)$.

   (b) Analyze time complexity for each step of the proposed algorithm.

   Example: Consider Figure 1. The vertical lines on the x-axis represent the petrol pumps. Here, Sachin will have to fill petrol at places marked with X. Hence, the solution will be 7 including the starting point. [7+3]

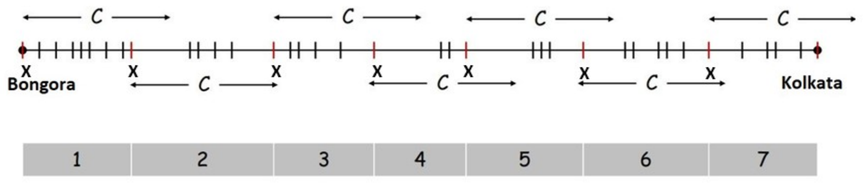

Figure 1: Example: Travelling Problem

9. You are consulting for a small computation-intensive investment company, and they have the following type of problem that they want to solve over and over. A typical instance of the problem is the following. They're doing a simulation in which they look at n consecutive days of a given stock, at some point in the past. Let us number the days $i = 1, 2, \ldots, n$; for each day i, they have a *price p(i)* per share for the stock on that day. Suppose during this time period, they wanted to buy 1,000 shares on some day and sell all these shares on some (later) day. They want to know: When should they have bought and when should they have sold in order to have made as much money as possible? (If there was no way to make money during the n days, you should report this instead.)

   For example, suppose $n = 5$, $p(1) = 9$, $p(2) = 1$, $p(3) = 2$, $p(4) = 5$, $p(5) = 2$. Then you should return "buy on 2, sell on 4" (buying on day 2 and selling on day 4 means they would have made Rs 4 per share, the maximum possible profit for that period).

   Your investment friends were hoping to find the solution in $\mathcal{O}(n \log n)$ time. Devise such an algorithm (give pseudo-code) and analyze the time complexity of each step. [6+3]