CS:4310 Midterm Exam, Fall 2021

Your answers will be graded primarily for correctness, but clarity, precision, and conciseness will also be important.

1. In the *divide* step, Algorithm A starts by doing some work that results in five subproblems, each having size one-half of the size of the original problem. Algorithm A then recursively solves the five subproblems (in the *conquer* step) and *combines* the solutions of these subproblems to get a solution to the original problem. The divide and combine steps of Algorithm A run in $\log n$ time, where n is the input size of the problem.

   (a) Write the recurrence for the running time $T(n)$ of Algorithm A on input of size n. There is no need to specify base cases.

   (b) The Master Method cannot be directly used to solve this recurrence, but you can use the Master Method to obtain good upper and lower bounds on $T(n)$. Use the Master Method to obtain functions $f(n)$ and $g(n)$ such that $T(n) = \Omega(f(n))$ and $T(n) = O(g(n))$. Show all your work.

2. Here is a function called Unusual that appears in our textbook.

```
UNUSUAL(A[1..n]):
  if n = 2
      if A[1] > A[2]                    ⟨⟨the only comparison!⟩⟩
          swap A[1] ⟷ A[2]
  else
      for i ← 1 to n/4                  ⟨⟨swap 2nd and 3rd quarters⟩⟩
          swap A[i + n/4] ⟷ A[i + n/2]
      UNUSUAL(A[1..n/2])                ⟨⟨recurse on left half⟩⟩
      UNUSUAL(A[n/2 + 1..n])            ⟨⟨recurse on right half⟩⟩
      UNUSUAL(A[n/4 + 1..3n/4])         ⟨⟨recurse on middle half⟩⟩
```

   Find the asymptotic running time of this function. Show all your work.

3. Suppose you are given two sets of n points, one set $\{p_1, p_2, \ldots, p_n\}$ on the vertical line $y = 0$ and the other set $\{q_1, q_2, \ldots, q_n\}$ on the vertical line $y = 1$. Create a set of n line segments by connecting each point p_i to the corresponding point q_i. We now want to compute the number of pairs of these line segments that intersect. Let us call this the *Line Segment Intersection (LSI)* problem.

   You have already solved a problem – let us call this the *Mystery* problem – that is just the LSI problem in disguise. In 3-4 sentences, describe an $O(n \log n)$ time algorithm for LSI, by *reducing* LSI to the Mystery problem. In other words, your solution should have 3 parts:

   (i) How we can efficiently translate the input of LSI to the input of the Mystery problem,

   (ii) How we can efficiently solve the Mystery problem, and

   (iii) How we can take the solution of the Mystery problem and use it to obtain the solution of LSI.

   Part (ii) is trivial, since we have already solved the Mystery problem and you should just say so.

4. Here is a recurrence describing the solution of a problem $P(i, j)$, for $i = 2, 3, \ldots, m$ and $j = 1, 2, \ldots, n-1$ in terms of smaller versions of the same problem:

$$OPT(i,j) = \min \begin{cases} \min_{j+1 \le p \le n} OPT(i,p) + 1, \\ \min_{1 \le q \le i-1} OPT(q,j) + 10, \end{cases}$$

The bases cases of this problem are $P(1, j)$ for $j = 1, 2, \ldots, n$ and $P(i, n)$ for $i = 1, 2, \ldots, m$ and it takes $O(1)$ time to solve each of these base cases.

Suppose we define a memoization data structure `Table[1..m, 1..n]` to store the solutions to all problems $P(i, j)$.

(a) What is the order in which you will fill `Table`? Justify your answer in 1 sentence.

(b) What is the running time of an iterative algorithm that completely fills `Table`? Justify your answer in 1-2 sentences.

5. You are given matrices $A_1, A_2, \ldots, A_n$ and you want to compute the matrix product

$$A_1 \times A_2 \times \cdots \times A_n.$$

Each matrix A_i has dimensions $m_{i-1} \times m_i$. Note that the product is a matrix of dimensions $m_0 \times m_n$.

Because matrix multiplication is associative, one can perform the multiplications in any order. However, different multiplication orders can have very different costs. Recall that multiplying an $a \times b$ matrix and a $b \times c$ matrix in the elementary fashion takes $a \cdot b \cdot c$ multiplications and we will use this as a measure of the cost of multiplying the matrices. For example, suppose that A_1 has dimensions 50×20, A_2 has dimensions 20×1, and A_3 has dimensions 1×10. Then multiplying in the order $(A_1 \times A_2) \times A_3$ will have cost $50 \cdot 20 \cdot 1 + 50 \cdot 1 \cdot 10 = 1000 + 500 = 1500$. This is because we first multiply $A_1 \times A_2$ and this has cost $50 \cdot 20 \cdot 1 = 1000$. Then we multiply a 50×1 matrix (the product of A_1 and A_2) with a 1×10 matrix (A_3) and this has an additional cost $50 \cdot 1 \cdot 10 = 500$. Now note that multiplying in the other order, i.e., $A_1 \times (A_2 \times A_3)$ has cost $20 \cdot 1 \cdot 10 + 50 \cdot 20 \cdot 10 = 10200$. From this example it should be clear that performing $A_1 \times A_2$ first, before the other multiplication, is much cheaper than the other option of multiplying $A_2 \times A_3$ first.

This problem has you construct a dynamic programing algorithm that takes as input the matrix dimensions $m_0, m_1, \ldots, m_n$ and finds the cost of a cheapest ordering of the multiplications. Note that your algorithm is not actually multiplying the matrices, just figuring out the order in which the matrices are to be multiplied.

Let $MinCost(i, j)$ denote the cost of the cheapest order of multiplying $A_i \times A_{i+1} \times \cdots \times A_j$ for $1 \leq i < j \leq n$. Write a recurrence expressing $MinCost(i, j)$ in terms of costs of cheapest multiplication orderings for smaller subproblems and identify base cases.
