CSE 100: Algorithm Design and Analysis
Midterm Exam 1
Time: 3:00-4:15pm

Spring 2023

Note: This is a **closed** book examination. You have 75 minutes to answer as many questions as possible. The number in the square bracket at the beginning of each question indicates the number of points given to the question. Write all of your answers directly on this paper.

Please make your answers concise as well as precise. If there is something in the question that you believe is open to interpretation, then please go ahead and interpret, but state your assumptions in your answer. Or you can ask the TAs proctoring the exam for clarification. There is no penalty for attempting to answer a question with the wrong answer, so please answer as many questions as you can. Partial credit will be given even if the answer is not fully correct. If you run out of space you can ask for extra papers. Please write down your name on *every* page as some papers may fall off although it rarely happens. 10 pages in total including this cover. The maximum points you can earn is 101. The one point above 100 will be considered as bonus.

Problem	Points earned	Out of
1		12
2		3
3		5
4		5
5		10
6		10
7		10
8		10
9		12
10		12
11		12
Sum		Max 101

Name _____

1. **[12 points]**. For each of the following claims, decide if it is true or false. *No* explanation is needed.

   (a) **[2 points]** If $f = \Theta(g)$, then $g = \Theta(f)$.
   True **False** **Sol.** True

   (b) **[2 points]** If $f = \Theta(g)$, then $f = O(g)$.
   True **False** **Sol.** True

   (c) **[2 points]** If $n \log n + n + \log n = n \log n + \Theta(n)$.
   True **False** **Sol.** True

   (d) **[2 points]** The running time of Insertion Sort on the input $\langle 1, 2, 3, ..., , n-1, n \rangle$ is $\Omega(n^2)$.
   True **False** **Sol.** False

   (e) **[2 points]** The recurrence for the running time of Strassen's algorithm, which is used to multiply two n by n matrices, is $T(n) = 8T(n/2) + \Theta(n^2)$.
   True **False** **Sol.** False

   (f) **[2 points]** The running time of Merge sort is $\Theta(n \log n)$.
   True **False** **Sol.** True

2. **[3 points]** Suppose you run Insertion sort on the input $\langle 4, 3, 2, 5, 1 \rangle$. Show the array right after each iteration.
   Sol.

   (4, 3, 2, 5, 1); it's okay to have this line.
   3, 4, 2, 5, 1
   2, 3, 4, 5, 1
   2, 3, 4, 5, 1
   1, 2, 3, 4, 5

3. **[5 points]** Formally prove that $n^3+5n+2 = \Theta(n^3)$. You can directly use the formal definition of Θ. Or you can show the LHS is $O(n^3)$ and also $\Omega(n^3)$.
Sol. $(LHS) \geq n^3$ for all $n \geq 1$. so, it is $\Omega(n^3)$. Further, $(LHS) \leq 10n^3$ for all $n \geq 1$, so it is $O(n^3)$. Note that there are infinitely many possible solutions.

4. **[5 points]** Briefly explain the Random Access Model (RAM).
Sol. Single processor (no parallel computing). Assumes each basic operation takes $O(1)$ time. Simple memory structure (and random memory access). Full points if a student explains two out of these three correctly. If they get only one, they will earn 3 points.

5. **[10 points]** The following is a pseudocode for the naive divide-and-conquer algorithm for matrix multiplication. Here, partitioning a matrix means doing so into four $n/2$ by $n/2$ (sub-)matrices.

```
SQUARE-MATRIX-MULTIPLY-RECURSIVE(A, B)
 1  n = A.rows
 2  let C be a new n × n matrix
 3  if n == 1
 4      c11 = a11 · b11
 5  else partition A, B, and C as in equations (4.9)
 6      C11 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B11)
               + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B21)
 7      C12 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A11, B12)
               + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A12, B22)
 8      C21 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B11)
               + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B21)
 9      C22 = SQUARE-MATRIX-MULTIPLY-RECURSIVE(A21, B12)
               + SQUARE-MATRIX-MULTIPLY-RECURSIVE(A22, B22)
10  return C
```

Give the recurrence for the running time of Square-Matrix-Multiply-Recursive and solve it. We let $T(n)$ denote the running time when input matrices are n by n. No need to show how you solved it. Just state the final result along with the recurrence.

Sol. $T(n) = 8T(n/2) + \Theta(n^2)$. (2.5 pts)
$T(n) = \Theta(n^3)$. (2.5 pts)

The Strassen's algorithm improves upon this naive algorithm by reducing multiplications. Give the recurrence for the running time of Strassen's algorithm and solve it. As before, no need to show how you solved it.

Sol. $T(n) = 7T(n/2) + \Theta(n^2)$. (2.5 pts)
$T(n) = \Theta(n^{\log_2 7})$. (2.5 pts)

6. **[10 points]** Consider the pseudocode of $Merge(A, p, q, r)$, which is shown below. Given that $A[p...q]$ and $A[q+1...r]$ are both sorted, the function call merges the two sorted subarrays into a sorted subarray $A[p...r]$. We would like to prove the correctness of Merge. For simplicity, you can assume that $L[1...n_1] = A[p...q]$ and $R[1...n_2] = A[q+1...r]$., and $L[n_1+1] = R[n_2+1] = \infty$. (So you only need to consider from lines 10). You can assume that all elements stored in the array have distinct values. *State the loop invariant of the for loop Lines 12-17.* No need to prove the correctness.

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```

Sol. Loop Invariant: In the beginning of each for loop of iteration,

(a) $A[p...k-1]$ is a sorted subarray of elements from $L[1...i-1]$ and $R[1...j-1]$, where $i \leq n_1 + 1$ and $j \leq n_2 + 1$; and

(b) Any elements in $L[i...n_1+1]$ and $R[j...n_2+1]$ are greater than any elements in $A[p...k-1]$.

7. **[10 points]**. Assume that you can use Merge(A, p, q, r); see the previous page. Give a pseudocode of Merge-Sort(A, p, r), which is supposed to sort $A[p \ldots r]$.

```
Merge-Sort(A, p, r)
```

8. **[10 points]** Rank the following functions by order of growth; that is, find an ordering $g_1, g_2, \cdots, g_k$ (here k is the number of functions given) such that $g_1 = O(g_2)$, $g_2 = O(g_3), \cdots, g_{k-1} = O(g_k)$. (For example, if you are given functions, $n^2, n, 2n$, your solution should be either $n, 2n, n^2$ or $2n, n, n^2$.)

$$n^3 \qquad 2^n \cdot n^2 \qquad \log\log n + \log n \qquad \log^{99} n \qquad 10^{100}$$

Sol.

$$10^{100} \quad \log\log n + \log n \quad \log^{99} n \quad n^3 \quad 2^n \cdot n^2$$

9. **[12 points]** Solve $T(n) = 4T(n/2) + \Theta(n^2)$ using the recursion tree method. Cleary state the tree depth, each subproblem size at depth d, the number of subproblems/nodes at depth d, workload per subproblem/node at depth d, (total) workload at depth d – or they should be clear from your answer. If your answer is correct for all questions below, you will get full points.

   - Tree depth:
   - each subproblem size at depth d:
   - number of subproblems/nodes at depth d:
   - workload per subproblem/node at depth d:
   - (total) workload at depth d:
   - $T(n) =$.

   Sol. Each of the following is worth 2 pts.

   - Tree depth: $D = \log_2 n$.
   - each subproblem size at depth d: $n/2^d$.
   - the number of subproblems/nodes at depth d: 4^d
   - workload per subproblem/node at depth d: $\Theta(n^2/4^d)$
   - (total) workload at depth d: $\Theta(n^2)$.
   - $T(n) = \Theta(n^2 \log n)$.

10. **[12 points]** Solve the following recurrences using the *Master Theorem*. You must state *which case applies*. (No need to state what the value of a, b, ϵ are). If the theorem is not applicable, just say N/A.

Theorem 4.1 (Master theorem)
Let $a \geq 1$ and $b > 1$ be constants, let $f(n)$ be a function, and let $T(n)$ be defined on the nonnegative integers by the recurrence

$$T(n) = aT(n/b) + f(n) ,$$

where we interpret n/b to mean either $\lfloor n/b \rfloor$ or $\lceil n/b \rceil$. Then $T(n)$ has the following asymptotic bounds:

1. If $f(n) = O(n^{\log_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\log_b a})$.
2. If $f(n) = \Theta(n^{\log_b a})$, then $T(n) = \Theta(n^{\log_b a} \lg n)$.
3. If $f(n) = \Omega(n^{\log_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and all sufficiently large n, then $T(n) = \Theta(f(n))$. ■

(a) $T(n) = T(\frac{1}{3}n) + T(\frac{1}{4}n) + \Theta(n)$.

Sol. N/A. Each problem is worth 3pts.

(b) $T(n) = 6T(n/2) + \Theta(n^2)$.

Sol. Case 1. $T(n) = \Theta(n^2)$. (1 pt for stating which case).

(c) $T(n) = 4T(n/2) + \Theta(n^2 \log n)$.

Sol. N/A

(d) $T(n) = 2T(n/2) + \Theta(n^3)$.

Sol. Case 3. $T(n) = \Theta(n^3)$. (1 pt for stating which case).

11. **[12 points]** In the Max-Subarray problem, we designed a $O(n \log n)$ time algorithm using divide and conquer. In the algorithm and its analysis, we showed how we can compute $\max_{1 \le i \le n/2 < j \le n} S[i, j]$ in $O(n)$ time, where $S[i, j]$ is defined as $A[i] + A[i + 1] + \cdots + A[j]$; here we assumed n is even. Explain how we can compute it in $O(n)$ time. As you learned, you can describe your algorithm in English or using a pseudocode, or by a combination of the two.