CSE 100: Algorithm Design and Analysis
Midterm Exam 1
Time: 4:30-5:45pm

Spring 2024

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 your answers directly in the exam booklet

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 100.

Problem	Points earned	Out of
1		20
2		5
3		5
4		8
5		10
6		12
7		10
8		10
9		10
10		10
Sum		Max 100

Name _____

1. **[20 points]**. For each of the following claims, decide if it is true or false. *No* explanation is needed.

   (a) **[2 points]** $100 \log n = O(n)$.
   True **False** **Sol.** True

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

   (c) **[2 points]** if $f = O(g)$, then $g = O(f)$.
   True **False** **Sol.** False

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

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

   (f) **[2 points]** $n^2 + 5n + 100 = n^2 + O(n)$.
   True **False** **Sol.** True

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

   (h) **[2 points]** $4^n = \Omega(n^4)$.
   True **False**
   Sol. True

   (i) **[2 points]** Mergesort has an asymptotically better running time than Insertion-sort.
   True **False**
   Sol. True

   (j) **[2 points]** $\lim_{n\to\infty} f(n)/g(n) = 0$, then $f(n) = O(g(n))$.
   True **False**
   Sol. True

2. **[5 points]** Formally prove that $10n^2 + 5n + 2 = O(n^2 \lg n)$ using the definition of the O notation.
   Sol. $10n^2 + 5n + 2 \leq cn^2 \lg n$ for all $n \geq n_0$ where $(c, n_0) = (100, 2)$. There are many correct answers.

3. **[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.

4. **[8 points]** The following is a pseudocode for the Strassen algorithm for matrix multiplication. Here, partitioning a matrix means doing so into four $n/2$ by $n/2$ (sub-)matrices.

    1. Divide the input matrices A and B and output matrix C into $n/2 \times n/2$ submatrices, as in equation (4.9). This step takes $\Theta(1)$ time by index calculation, just as in SQUARE-MATRIX-MULTIPLY-RECURSIVE.
    2. Create 10 matrices $S_1, S_2, \ldots, S_{10}$, each of which is $n/2 \times n/2$ and is the sum or difference of two matrices created in step 1. We can create all 10 matrices in $\Theta(n^2)$ time.
    3. Using the submatrices created in step 1 and the 10 matrices created in step 2, recursively compute seven matrix products $P_1, P_2, \ldots, P_7$. Each matrix P_i is $n/2 \times n/2$.
    4. Compute the desired submatrices $C_{11}, C_{12}, C_{21}, C_{22}$ of the result matrix C by adding and subtracting various combinations of the P_i matrices. We can compute all four submatrices in $\Theta(n^2)$ time.

    (a) State the running time of each step.

    i. Running Time: $\Theta(1)$ (or $\Theta(n^2)$)

    ii. **[2 points]** Running Time:
    Sol. $\Theta(n^2)$

    iii. **[2 points]** Running Time:
    Sol. $7T(n/2)$

    iv. Running Time: $\Theta(n^2)$.

    (b) **[4 points]** Give the recurrence for the running time of Strassen's algorithm and solve it. No need to show how you solved it.

Sol. $T(n) = 7T(n/2) + \Theta(n^2)$.
$T(n) = \Theta(n^{\log_2 7})$. It's okay to use O instead of Θ.

5. **[10 points]**. Assume that you can use Merge(A, p, q, r), which merges two sorted subarrays $A[p \ldots q]$ and $A[q+1 \ldots r]$ into a sorted array $A[p \ldots r]$. Give a pseudocode of Merge-Sort(A, p, r), which is supposed to sort $A[p \ldots r]$.

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

6. **[12 points]** The following is a pseudocode of Insertion-sort. Prove its correctness via loop invariant. In other words, state the *loop invariant* and prove it using *Initialization (Base case), Maintenance (Inductive Step), and Termination (Conclusion).*

```
Input: A[1 ... n]
for j = 2 to A.length
  Find 1 <= i <= j-1 s.t. A[i] <= A[j] <= A[i+1] if such i exists; otherwise, i = 0
  Push A[i+1 ... j-1] to A[i ... j]
  while moving A[j] to A[i+1] (need to use a temp var.)
```

Sol. Loop Invariant: At the start of each iteration of the (first) for loop, the subarray $A[1..j\text{-}1]$ consists of the elements originally in $A[1..j-1]$, but in sorted order. See the lecture slides/textbook for the proof.

7. **[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 $2n, n, n^2$.)

$$n \log n \qquad \log^5 n \qquad (\log \log n)^2 \qquad 4^n \qquad 1000^{1000} \qquad n^3$$

Sol.

$$1000^{1000} \qquad (\log \log n)^2 \qquad \log^5 n \qquad n \log n \qquad n^3 \qquad 4^n$$

8. **[10 points]** Solve $T(n) = 2T(n/2) + \Theta(n^2)$ 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, except the tree depth, which is worth 1pt. The score will be capped at 10.

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

9. **[10 points]** Solve the following recurrences using the *Master Theorem*. If the theorem is not applicable, just say N/A. If it is, just give the answer. (No need to show how you solved it).

> ***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) $T(n) = 4T(n/2) + \Theta(n^2)$.

Sol. $T(n) = \Theta(n^2 \log n)$.

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

Sol. N/A

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

Sol. $T(n) = \Theta(n^3)$.

(d) $T(n) = 4T(n/2) + \Theta(n)$.

Sol. $T(n) = \Theta(n^2)$.

10. **[10 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.

Sol. The key observation is $\max_{1 \le i \le n/2 < j \le n} S[i, j] = \max_{1 \le i \le n/2} S[i, n/2] + \max_{n/2 < j \le n} S[n/2 + 1, j]$.

Then, we can compute $S[n/2, n/2]$, $S[n/2 - 1, n/2]$, $S[n/2 - 2, n/2]$, ... , $S[1, n/2]$, in this order in $O(n)$ time, and take the maximum of them. Thus, we can compute the first term, $\max_{1 \le i \le n/2} S[i, n/2]$ in $O(n)$ time. We can similarly compute the second term in $O(n)$ time.