CSE 100: Algorithm Design and Analysis
Midterm Exam 1

Spring 2018

Note: Please write down your name on *every* page. 9 pages in total including this cover. Time: 4:30-5:45

Problem	Points earned	Out of
1		20
2		10
3		15
4		5
5		10
6		10
7		10
8		10
9		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]** In the Random Access Model (RAM), there are multiple machines available for computation. Therefore, one can run each thread on a separate machine.
   True **False** **Sol.** False

   (b) **[2 points]** If an algorithm's worst case running time is $\Theta(n^2)$, it means that the running time is $\Theta(n^2)$ for all inputs of size n.
   True **False** **Sol.** False

   (c) **[2 points]** It takes at least $\Omega(n \log n)$ time for Mergesort (in the textbook) to sort any input of n elements.
   True **False** **Sol.** True

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

   (e) **[2 points]** If $f = O(g)$ and $g = O(h)$, then it must be the case that $f = O(h)$.
   True **False** **Sol.** True

   (f) **[2 points]** If $A(n) = 2A(n/2) + 1000n$ with $A(1) = 1$ and $B(n) = 2B(n/2) + n$ with $B(1) = 1$, then $A(n) = \Theta(B(n))$.
   True **False** **Sol.** True

   (g) **[2 points]** $\log^{15} n = \Omega(\sqrt{n})$.
   True **False** **Sol.** False

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

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

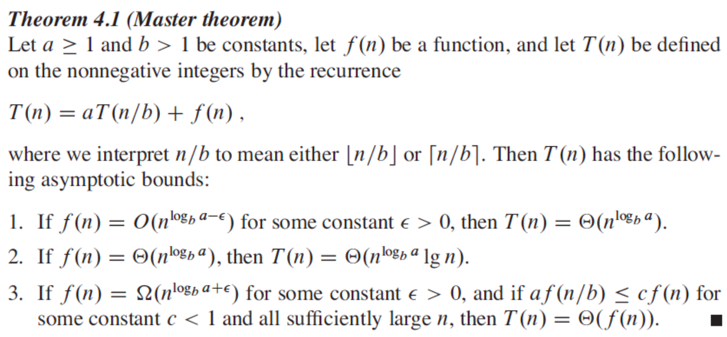
   (j) **[2 points]** $\lim_{n\to\infty} f(n)/g(n) = 0$, then $f(n) = O(g(n))$.
   True **False** **Sol.** True

2. **[10 points]** The following is a pseudocode of Insertion-sort. We would like to prove its correctness via loop invariant. Write down a loop variant. Complete the proof using Initialization, Maintenance, Termination.

```
INSERTION-SORT($A$)
for $j = 2$ to $A.length$
    $key = A[j]$
    // Insert $A[j]$ into the sorted
        sequence $A[1\,..\,j-1]$.
    $i = j - 1$
    while $i > 0$ and $A[i] > key$
        $A[i+1] = A[i]$
        $i = i - 1$
    $A[i+1] = key$
```

Sol. Loop invariant (5pts), inti. (1pt), Maintenance (3pts), Termination (1pt).

3. **[15 points]** Solve the following recurrences using the Master Theorem. You must state which case applies and set parameters (like a, b, c, ϵ). 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) $T(n) = 27T(n/3) + 10n^2$. **Sol.** Case 1. $a = 27$, $b = 3$, ϵ can be any positive constant ≤ 1. Therefore, $T(n) = \Theta(n^3)$.

(b) $T(n) = 2T(n/2) + n \log n$. **Sol.** N/A. Reason. $f(n) \neq \Theta(g(n) = n)$. Further, one is not polynomially larger than the other. No need to explain the reason.

(c) $T(n) = 9T(n/3) + 4n^2$. **Sol.** Case 2. $f(n) = 4n^2$ and $g(n) = n^{\log_3 9} = n^2$, thus $f(n) = \Theta(n^2 \log n)$.

4. [**5 points**] Formally prove that $n^2 = \Omega(2n^2 + 5)$ using the definition of $\Omega(\cdot)$. **Sol.** $\frac{1}{7}(n^2 + 100) \leq n^2$ for all $n \geq 1$.

5. [**10 points**] Solve the following recurrence using the *substitution method*: $T(n) = 2T(n/2)$ (no need to show the base case). **Sol.** We would like to show that $T(n) \leq n$. Assuming that this claim holds true for $1, 2..., n-1$, we have $T(n) = 2T(n/2) \leq 2(n/2) = n$, as desired.

6. **[10 points]** Solve the following recurrence using the *recursion tree method*: $T(n) = 4T(n/2)+n$. To get full points, your solution must clearly state the following quantities: the tree depth (be careful with the log base), each subproblem size at depth d, the number of nodes at depth d, workload per node at depth d, (total) workload at depth d.

   Sol. The tree visualization is omitted. (rough visualization: 1pts)
   For simplicity, let $T(1) = 1$. Tree depth $D = \log_2 n$. (1.5pts)
   Each subproblem size at depth d: $n/2^d$.(1.5pts)
   Number of nodes at depth d: 4^d (1.5pts)
   WL per node at depth d: $n/2^d$ (1.5pts)
   WL at depth d: $2^d n$. (1.5pts)
   So, $\sum_{d=0}^{D} 2^d n = \Theta(2^D n) = \Theta(n^2)$. (bottom level dominates) Final answer (1.5pts)

7. **[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 $n \times n$ matrix
3  if $n == 1$
4      $c_{11} = a_{11} \cdot b_{11}$
5  else partition $A$, $B$, and $C$ as in equations (4.9)
6      $C_{11}$ = SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{11}, B_{11}$)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{12}, B_{21}$)
7      $C_{12}$ = SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{11}, B_{12}$)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{12}, B_{22}$)
8      $C_{21}$ = SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{21}, B_{11}$)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{22}, B_{21}$)
9      $C_{22}$ = SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{21}, B_{12}$)
              + SQUARE-MATRIX-MULTIPLY-RECURSIVE($A_{22}, B_{22}$)
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.

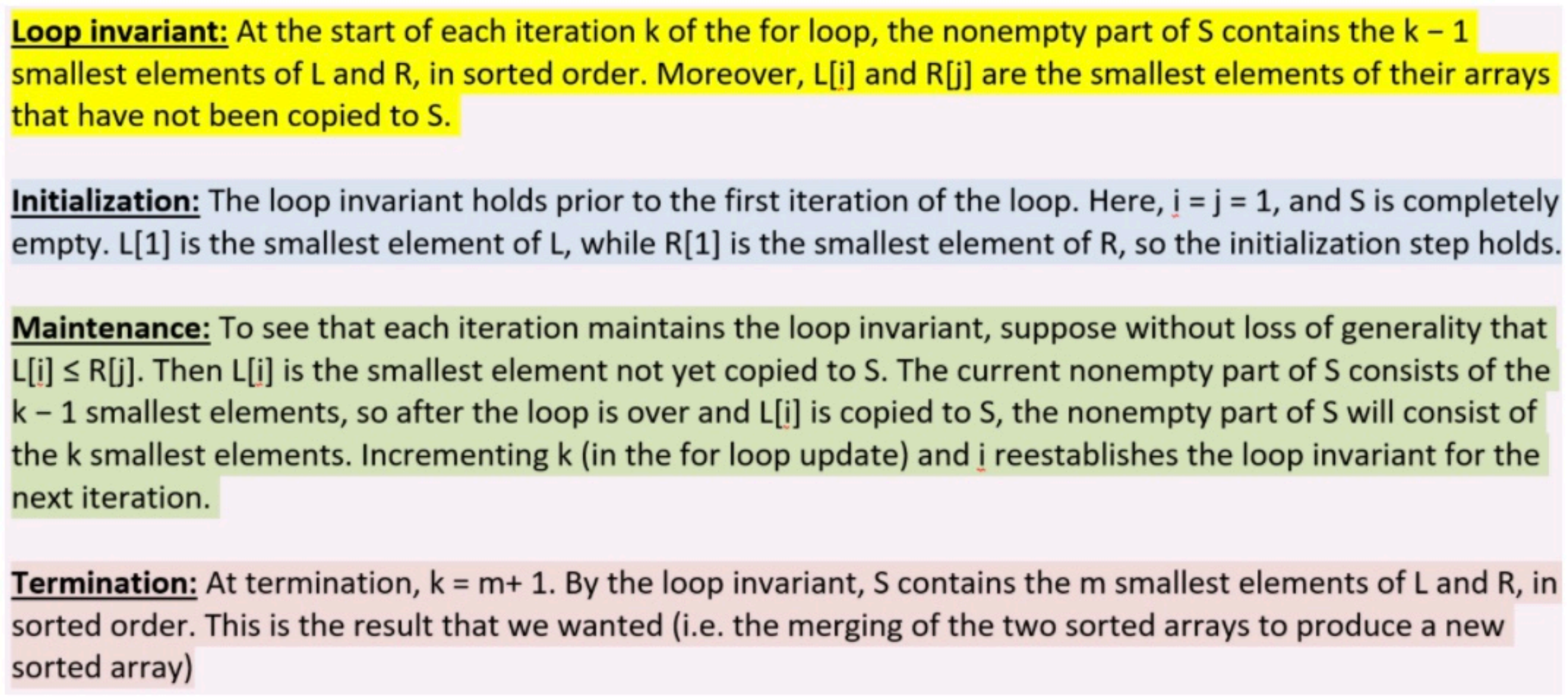
Sol. $T(n) = 8T(n/2) + \Theta(n^2)$. (2.5 pts)
$T(n) = \Theta(n^3)$. (2.5 pts)

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)

8. **[10 points]** Merge sort implementation. Merge-Sort(A, p, r) sorts the subarray $A[p...r]$ in increasing order. Give a pseudocode of Merge-Sort(A, p, r). You are allowed to use Merge(A, p, q, r), which merges two sorted arrays, $A[p...q]$ and $A[q+1...r]$, into a sorted array $A[p...r]$.

   Sol. See the textbook



Loop invariant: At the start of each iteration k of the for loop, the nonempty part of S contains the $k - 1$ smallest elements of L and R, in sorted order. Moreover, $L[i]$ and $R[j]$ are the smallest elements of their arrays that have not been copied to S.

Initialization: The loop invariant holds prior to the first iteration of the loop. Here, $i = j = 1$, and S is completely empty. $L[1]$ is the smallest element of L, while $R[1]$ is the smallest element of R, so the initialization step holds.

Maintenance: To see that each iteration maintains the loop invariant, suppose without loss of generality that $L[i] \leq R[j]$. Then $L[i]$ is the smallest element not yet copied to S. The current nonempty part of S consists of the $k - 1$ smallest elements, so after the loop is over and $L[i]$ is copied to S, the nonempty part of S will consist of the k smallest elements. Incrementing k (in the for loop update) and i reestablishes the loop invariant for the next iteration.

Termination: At termination, $k = m+ 1$. By the loop invariant, S contains the m smallest elements of L and R, in sorted order. This is the result that we wanted (i.e. the merging of the two sorted arrays to produce a new sorted array)

9. **[10 points]** In the max subarray problem, we are given as input an array $A[1...n]$ of n elements. Here, our goal is to compute the largest sum of all elements in any subarray of $A[1...n]$. In other words, the goal is to compute $\max_{1 \leq i \leq j \leq n} S[i, j]$, where $S[i, j] = A[i] + A[i+1] + ... + A[j]$. We learned a $O(n \log n)$ time algorithm based on divide and conquer for the max subarray problem.

   The algorithm required us to compute $\max_{1 \leq i \leq n/2 < j \leq n} S[i, j]$ in $O(n)$ time; for simplicity, here let's assume n is even. Describe such an algorithm. It must be clear why the running time is $O(n)$ from the description unless you explain why.

   Sol. We can compute $S[n/2, n/2]$, $S[n/2 - 1, n/2]$, ..., $S[1, n/2]$ in this order in $O(n)$ time. Take the maximum of these. Likewise, we can compute $S[n/2+1, n/2+1], S[n/2+1, n/2+2], ...S[n/2+1, n]$ in this order in $O(n)$ time. Choose the max of these and add up the two chosen maximum values.