CSE 100: Algorithm Design and Analysis
Midterm 1

Spring 2021

- This is a take-home exam with no proctoring. You can start at any time once you get access to this exam. You have to submit your solution by **5:30pm, 16-FEB**, through CatCourses (Midterm 1 under Assignments). You can resubmit any number of times until the deadline. If there are any technical issues with uploading, it is extremely important to immediately contact the instructor or the TA.
- This is an open-book exam. The **only** resources you can use during the exam are all **course materials** uploaded to CatCourses plus the **textbook**. In particular, this means that you are NOT allowed to search for solutions on the internet. No calculator is allowed. You have to take the exam by yourself.
- There are 10 problems in total. You can earn some partial points if you show progress even if you can't solve problems completely.
- 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.
- If you have questions, you can email the instructor and the TAs. However, we may not answer your questions on time as we can't be available all the time.

You're required to take the following honor pledge prior to taking the exam.

> *By completing this exam, I acknowledge and confirm that I will not give or receive any unauthorized assistance on this examination. I will conductmyself within the guidelines of the university academic integrity guidelines.*

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

   (a) (2 points) $n \log n = O(n^2)$.

   (b) (2 points) $\log \log n = O(\log n)$.

   (c) (2 points) $\log^{50} n = O(n^{0.1})$.

   (d) (2 points) $4^n = O(2^n)$.

   (e) (2 points) $100^{100} = \Theta(1)$.

   (f) (2 points) if $f = O(g)$, then $g = \Omega(f)$.

   (g) (2 points) if $n^3 = \Omega(n^2)$.

   (h) (2 points) If an algorithm's worst case running time is $O(n^2)$, it means that the running time is $\Theta(n^2)$ for all inputs of size n.

   (i) (2 points) $100 + 200n + 300n^2 = \Theta(n^2)$

   (j) (2 points) $\sum_{i=1}^{n} \log i = \Theta(n \log n)$.

2. (4 points) The following is a pseudocode of Insertion-sort. For instance $A[1 \dots 8] = \langle 7, 4, 2, 9, 4, 3, 1, 6 \rangle$, what is $A[1 \dots 8]$ just before the for-loop starts for $j = 5$?

```
Insertion-Sort(A)
1.  for j = 2 to A.length
2.    key = A[j]
3.    // Insert A[j] into the sorted sequence A[1...j - 1].
4.    i = j - 1
5.    while i > 0 and A[i] > key
6.      A[i + 1] = A[i]
7.      i = i - 1
8.    A[i + 1] = key
```

3. (a) (3 points) Give an instance of size n for which the Insertion-sort terminates in $\Omega(n^2)$ time.

   (b) (3 points) Give an instance of size n for which the Insertion-sort terminates in $O(n)$ time.

4. (5 points) The following is a pseudocode of the naive divide-and-conquer algorithm for matrix multiplication. Here, partitioning a n by n matrix means partitioning it 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.

5. (5 points) Let $T(n)$ denote the running time of Merge-sort on input of size n. In the following you can omit floor or ceiling.

```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```

   What is the running time of Line 3 (of Merge-Sort)?

   What is the running time of Line 4 (of Merge-Sort)?

   What is the running time of Line 5 (of Merge-Sort)?

6. (5 points) Formally prove that $50n + 15 = O(n^2)$ using the definition of $O(\cdot)$.

7. (10 points) Give a *pseudocode* of Heap-sort. For simplicity, you can assume that the heap A you're given is *already* a max-heap. You can use the function MAX-HEAPIFY(i) as a sub-procedure. Recall that the function MAX-HEAPIFY(i) makes the subtree rooted at node i a max-heap, *if both* the left and right subtrees of node i are max-heaps. You can use $A.heapsize$

to denote the current heap size. If you can't give a pseudocode, you can describe the Heap-sort algorithm in words, but you will lose some points.

8. (20 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.*

```
Insertion-Sort(A)
1.  for j = 2 to A.length
2.    key = A[j]
3.    // Insert A[j] into the sorted sequence A[1...j - 1].
4.    i = j - 1
5.    while i > 0 and A[i] > key
6.      A[i + 1] = A[i]
7.      i = i - 1
8.    A[i + 1] = key
```

9. (10 points) The following is a pseudo-code of Selection-Sort. We would like to prove its correctness via loop invariant. State a *loop invariant*. Note that your loop invariant must be strong enough to lead to the correctness of the algorithm. You only need to state the loop invariant. (*No* need to show Initialization, Maintenance, or Termination.)

```
Selection-Sort(A)
1.  n = A.length
2.  for j = 1 to n-1
3.   smallest = j
4.    for i = j+1 to n
5.      if A[i] < A[smallest]
6.        smallest = i
7.    exchange A[j] with A[smallest]
```

10. (15 points) Prove that the Merge-Sort pseudocode shown in Question 5 is correct. That is, prove that Merge-Sort(A, p, r) sorts the subarray $A[p...r]$ in increasing order. You can assume that Merge(A, p, q, r) successfully merges two sorted arrays, $A[p...q]$ and $A[q + 1...r]$, into a sorted array $A[p...r]$. Please do not forget the *base* case.