ECE368 Mid-Term Exam 2
2019

- **Before the exam:** This is a closed book exam. Calculator, smart devices, and cheatsheets are **NOT** allowed. In choosing your seat, maximize your distance from other students, e.g. seat every other row and/or line.
- **This exam consists of 9 pages** (including this cover sheet). Please check to make sure that all of these pages are present before you begin. Credit will not be awarded for pages that are missing. It is your responsibility to make sure that you have a complete copy of the exam.
- **No questions are allowed** during the exam in order to maintain order and fairness. We only respond to logistics requests. If you have technical doubts that may impact your answers, clearly write down your assumptions as "**ASSUMPTION:**..". We will consider your assumptions in grading.
- **For questions that come with answer boxes:** only answers written inside the boxes will be considered.
- **When time is up:** Stop writing immediately; Form a line to sign out and turn in the exam; Do not chat before you leave the classroom.

  We strictly enforce these rules. Any violation will be recorded and reported.

IMPORTANT: Write your user (login) ID at the TOP of EACH page. Also, be sure to ***read and sign*** the ***Academic Honesty Statement*** that follows:

"In signing this statement, I hereby certify that the work on this exam is my own and that I have not copied the work of any other student while completing it. I understand that, if I fail to honor this agreement, I will receive a score of ZERO for this exam and will be subject to possible disciplinary action."

Printed Name: Sample Solution

login:

Signature:

DO NOT BEGIN UNTIL INSTRUCTED TO DO SO.

1. (15 points) Trees.

1(a). (10 points): Calculate the minimum and maximum number of nodes in a height-balanced binary tree (i.e., AVL tree) of height h, for `h = 1, 2, 3, 4, 10`. Use your results to fill out the following table.

Note: Think carefully about the definition of the tree height.

h	*Minimum*	*Maximum*
1	2	3
2	4	7
3	7	15
4	12	31
10	232	2047

Wrong answer receives 0 point.

1(b). (5 points): Suppose you wanted to insert the following elements into a binary search tree of integers: `90, 3, 16, 7, 12, 38, 50`. Which of the following insertion orders will result in a *complete* binary tree? Assume that the BST is empty to begin with.

Write down the letter of your choice: a

(a) 16, 7, 3, 12, 50, 38, 90

(b) 16, 38, 3, 12, 50, 90, 7

(c) 7, 16, 3, 90, 12, 50, 38

(d) 16, 3, 12, 50, 90, 7, 38

(e) More than one of the above orders will result in a complete binary tree.

(f) None above.

2. (15 points): In class, we performed partitioning and quicksort as follows:

```
Partition_lec(r[], lb, ub):
  pivot ← r[lb]
  down ← lb
  up ← ub
  while down < up {
    while r[down] ≤ pivot and down < ub {
      down++
    }
    while r[up] > pivot {
      up--
    }
    if down < up {
      r[down] ↔ r[up]
    }
  }
  r[lb] ← r[up]
  r[up] ← pivot
  return up

Quicksort_lec(r[], lb, ub):
  if lb ≥ ub {
    return
  }
  pivot_idx ← Partition_lec(r[], lb, ub)
  Quicksort_lec(r[], lb, pivot_idx-1)
  Quicksort_lec(r[], pivot_idx+1, ub)
```

2(a). (3 points): What is the time-complexity of the above quicksort procedure in `O()` notation when executed on an array of size `N` that has all identical elements?

Your choice of the big-O complexity (write down one letter): d

(a) O(n)

(b) O($n * logn$)

(c) O($n * \sqrt{n}$)

(d) O(n^2)

(e) O(n^3)

(f) None above

2(b). (12 points): Here is the pseudo code of a new quicksort algorithm that uses the average of each sub-array as the pivot. Note that the average may not be an element of the subarray.

Fill in the blanks of the pseudo code. If you believe no code is needed for a blank (e.g. for line 22), write down "N/A".

```
1  Partition(R[], avg, lb, ub)
2           down = lb
3           up = ub
4           while (down < up)
5                   while (R[down] ≤ avg and (down < ub))
6                           down++
7                   while (R[up] > avg)
8                           up--
9                   if down < up  /* anything here? */
10                  R[down]↔R[up]   /*swap*/
11          return up;
12
13 Quicksort(R[], lb, ub)
14          if lb ≥ ub {
15                  return
16          }
17          sum = 0
18          for i ← lb to ub
19                  sum += R[i]
20          avg = sum/(ub-lb+1)
21          pivot_idx = Partition(R[], avg, lb, ub)
22          N/A   /* anything here? */
23          Quicksort(R, lb, pivot_idx);
24          Quicksort(R, pivot_idx + 1, ub);
```

3. (8 points): Consider a max-heap, H, whose array representation is given below:

`31, 28, 20, 19, 15, 17, 18, 12, 16, 14`

3(a). (4 points): Show the **array representation** of the max-heap, H, after the execution of an *ExtractMax* operation on it. Heapify must be done. Only showing the tree representation will receive 0 credit.

Your answer here: {28, 19, 20, 16, 15, 17, 18, 12, 14}

3(b). (4 points): Show the **array representation** of the max-heap, H, after the execution of an *Insert(32)* operation on it. Start with the original max-heap and NOT the result of 3(a). Heapify must be done. Only showing the tree representation will receive 0 credit.

Your answer here: {32, 31, 20, 19, 28, 17, 18, 12, 16, 14, 15}

4. (12 points): Top K. The goal is design the fastest algorithm that you can.

4(a). (3 points): Design an algorithm to obtain the k-th largest element in an unsorted array. Describe the algorithm as **a short list of no more than 5 bullet points (okay if you have fewer than 5 points)**.

Your answer here:

- Randomly pick an element as pivot
- Partition use pivot; small→left, larger → right
- If pivot.index == k, return pivot
- If pivot.index < k, recurse on right
- If pivot.index > k, recurse on left

(3 points): State the average-case time complexity (in O() notation) of your algorithm in terms of n and/or k. Your answer: O(n)

4(b). (3 points): Suppose there exists an algorithm that can return the k-th largest element in an unsorted array in O(n) time. Design an algorithm to obtain a sorted list of elements ranked k-th and higher (i.e. the ranks of these elements are smaller than or equal to k). Describe the algorithm as **a list of no more than 5 bullet points (okay if you have fewer than 5 points)**.

- Use O(n) selection algorithm to select k-th element
- Iterate through whole array and pick out all elements ranked higher than the k-th element and save them to a new array
- Heap/quick sort the new array
-
-

(3 points): State the average-case time complexity (in O() notation) of your algorithm in terms of n and k. Your answer: O(n + klgk)

This Page is Intentionally Left Blank.

For grading purposes only. Do not write on this page.

Question 1:	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Question 2:	1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
Question 3:	1 2 3 4 5 6 7 8
Question 4:	1 2 3 4 5 6 7 8 9 10 11 12
TOTAL:	/50