Midterm #1

CIS 121—Fall 2017

In-class exam: Tuesday, October 3rd 2017.
Exam Starts: 10:30AM
Exam Ends: 11:50AM

This is a closed book exam. No computers or internet-connected devices are allowed during the exam. You are permitted to use one 8.5”x11” page with handwritten notes (these can be on both sides of the paper). If you need scratch paper, please get some from the front of the classroom. There are 10 questions on this exam, plus 2 extra credit questions, for a total of 12 questions. May the odds be ever in your favor.

Your name:

Your PennKey (i.e. ccb):
1 Analysis of Algorithms
10 points

For each of the statements below, please say whether it is true or false, and give a 1 sentence explanation of your answer.

1. Worst case analysis provides a running time bound that holds for every input of length $N$.

2. Worst case analysis is usually easier to establish than average case analysis.

3. We retain lower order terms in asymptotic analysis, since we are concerned with getting a very accurate estimate of running time.

4. Constant factors can depend on system architecture, choice of compiler or programming language.

5. To establish the bounds on the class of algorithms that solve a problem, we typically implement an algorithm to establish the lower bound, and rely on a proof to establish the upper bound.

6. Big Oh provides a good estimate of the average running time for an algorithm.

7. Asymptotic analysis is concerned with large values of $N$ and can be inaccurate for small $N$.

8. If an algorithm has a running time of $\Theta(N \log N)$ then: (say True/False for each of the items below).
   (a) It is $O(N \log N)$.
   (b) It is $\Omega(N \log N)$.
   (c) It is optimal.

9. If the lower bound on the class of algorithms that solve a problem is an algorithm $\Omega(N^2)$, and an algorithm in that class is $O(N^2)$ then:
   (a) The algorithm is $\Theta(N^2)$.
   (b) The algorithm is $\Omega(N^2)$.
   (c) The algorithm is optimal.

10. If two algorithms have the same running time in terms of Big Oh, then they will have equivalent running times in Tilde
2 Common Orders of Growth

10 points

The nature of software means that certain orders of growth are common in our algorithms. Enumerate the orders of growth that are typically encountered in computer science. Give their name, their expression in terms of $N$ and an example of how they arise in code. E.g.

- linear, $N$, a for loop from 0 to $N - 1$.

(Hint: you should think of at least 5 in addition to linear. You’ll get a little extra credit if you list a 6th that we haven’t talked about yet in class.)
3 Making sorting a priority
15 points

We would like to modify a simple sorting algorithm using one of the data structures that we learned in class, and see how it changes its performance.

1. What is the running time of the selection sort code given below? No need for a formal proof, just explain your answer in a sentence or two.

2. Re-implement the search for the smallest item with a priority queue. You can write in pseudocode for your answer, and you can assume you’re already given a priority queue implementation.

3. What is the running time of your new method? State any assumptions that you make about the priority queue operations.

For this problem, running time should be in Big Oh, not Tilde.

```java
public class Selection {
    public static void sort(int[] a) {
        int N = a.length;
        for (int i = 0; i < N; i++)
            int min = i;
        for (int j = i + 1; j < N; j++)
            if (a[j] < a[min])
                min = j;
        exchange(a, i, min);
    }
}

public class PQSelection {
    public void sort(int[] a) {
        int N = a.length;
        MinPQ pq = new MinPQ();
        for (int i = 0; i < N; i++)
            pq.insert(a[i]);
        for (int i = 0; i < N; i++)
            a[i] = pq.delMin();
    }
}
```
4 Picking the Right Sort
10 points

Insertion sort is $O(N^2)$ and merge sort is $O(N \log_2 N)$. Should we always prefer merge sort over insertion sort? Explain your answer. Give two reasons.

5 Impossible Running Time?
15 points

In class we discussed several implementations of a max priority queue that could sort Java Objects. We determined that the running time for the different implementations was the following:

<table>
<thead>
<tr>
<th>implementation</th>
<th>insert</th>
<th>delete max</th>
<th>find max</th>
</tr>
</thead>
<tbody>
<tr>
<td>unordered array</td>
<td>1</td>
<td>$N$</td>
<td>$N$</td>
</tr>
<tr>
<td>ordered array</td>
<td>$N$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>binary heap</td>
<td>$\log_2 N$</td>
<td>2 $\log_2 N$</td>
<td>1</td>
</tr>
<tr>
<td>$d$-ary heap</td>
<td>$\log_d N$</td>
<td>$d \log_d N$</td>
<td>1</td>
</tr>
</tbody>
</table>

We also alluded to more complex data structures which had better running times without discussing their details.

Prove or disprove: it is possible to have a max priority queue with an underlying data structure that supports constant time operations for insert, delete max, and find max. Please say whether it is possible or impossible, and give a justification for your answer. (Hint: think about the lower bound on comparison based sorting).
6 Symbol Tables and BSTs
5 points

For each of the statements below, please say whether it is true or false, and give a 1 sentence explanation of your answer.

1. Height of a binary search tree with $N$ elements will always be the floor of $\log_2 N$.

2. There is a single unique binary search tree for any input sequence.

3. A heap and a BST are both binary trees, but we implemented the heap using an array and a BST using a Java Object representing a tree.

4. Just like a heap, a BST is a complete binary tree.

5. Unlike a max priority queue, the ordered symbol table API (like the one we implemented with a BST) supports functions to get the minimum and get the maximum element.
7 Sorting in Java
15 points

We want to find good TAs for 121 next year, so we started writing an Object that stores information about a student (given below). Our goal is to sort the students by their CIS 121 scores, and email the top 20% of the students to ask them to apply to be TAs.

1. If we want to use one of Java’s built in sorting methods, what Java Interface must we implement? Write a short bit of code that implements the method from that interface, so that instances of the Student class will be sorted in descending order of cis121score. This means that students with the highest scores are ranked first.

2. Uh oh, CCB and Rajiv disagree about whether to sort by the 160 or the 121 scores. Luckily, Java overloads its sorting functions so that in addition to passing in an array to be sorted, you may also pass in another Object that allows a different sort order. What is that other Object? How would it simplify your code to satisfy CCB’s and Rajiv’s conflicting demands?

3. If we just email the 20% of the students who have the highest scores, then we might email graduating seniors too. Instead, someone proposes that we first use quicksort to rank the students by score, and then use quicksort again so that it re-groups the students by year. Then we could email the first 20% of the juniors, sophomores and freshman in the list. When we implement this idea, people with bad scores are going to be emailed. What went wrong and how do you fix it?

```java
public class Student implements Comparator<Student> {
    protected String name;
    protected int graduationYear;
    protected double cis121score;
    protected double cis160score;
    ...
}
```
8 Dynamically Resizing Arrays
15 points

In our implementation of a stack using a dynamically resizing array, when we try to push to the stack but the underlying array is full, we double the array’s size.

1. Why do we double the size of the array each time instead of just increasing the size of the array by 1? Discuss the worst-case and amortized running time of push() in your answer.

2. When we try to pop from the stack, we halve the size of the array when it is one-quarter full. Why not when it is half full? Think about what the worst case scenario would be.
9 Extra Credit: Middlest Priority Queue
10 points

You have a stream of $N$ integers coming in, and you can read one integer at a time. Design an $O(N \lg N)$ algorithm to output a length $N$ array, $A$, such that $A[i]$ is the median of the first $i$ elements of the stream. For the sake of this problem, $1 \leq i \leq N$. The median is defined as the element at index $\lceil \frac{N}{2} \rceil$ in a sorted ordering. Hint: use two heaps.

For example: Given $[5, 8, 1, 4, 3, 9]$, you should output $[5, 5, 5, 4, 4, 4]$.