Name	UIN	Section	Seat

Question:	1	2	3	4	5	6	Total
Points:	20	20	20	20	35	65	180
Score:							

CS 125 Final Exam

16 Dec 2017

Please fill out your name and UIN.

This three-hour final exam consists of six questions broken into three types:

1. four 20-point short-answer questions;
2. one 35-point medium answer question; and
3. one 65-point long answer question.

Point values assigned to each question are intended to help you budget your time—so you should work on a 10-point question for about 10 minutes. Answer each question as **clearly** and **succinctly** as possible. Draw pictures or diagrams if they help. If you think that a question is unclear or ambiguous, state any assumptions you are making as part of your answer. **Please write clearly.** You may lose points if we cannot read your answer.

No aids of any kind are permitted. And you may not leave the exam room with any pages from this exam or written notes. Do not detach any pages from this exam.

There are 2 scratch pages at the end of the exam. If you use them, please clearly indicate which question you are answering.

I have neither given nor received help on this exam.

Sign and Date: _____

1. Sorting and Searching (20 Points)

(a) **8 points** We have studied four sorting algorithms this semester—selection sort, insertion sort, mergesort, and quicksort. Fill in the table below with the *best* and *worst* case runtime for each algorithm using $\mathcal{O}$ notation[1] (1 points each).

Algorithm	Best Case	Worst Case
Selection Sort		
Insertion Sort		
Mergesort		
Quicksort		

(b) **4 points** Using insertion sort I sorted (10^6) unordered email addresses in 4.0 seconds. Ignoring possible memory limitations of my machine, approximately how long will it take to sort (10^8) unordered email addresses?

◯ $<$ 3 minutes ◯ 3–10 minutes ◯ 11–59 minutes ◯ 1–23 hours
◯ 24–72 hours ◯ $>$ 72 hours

[1] What we have called "big O notation".

(c) **8 points** Each week the agro-computational "compu-corn-icans" monks write the date and crop sales onto a new heavy clay tablet which they archive with others along a wall edge. They want to re-sort their tablets by lifting and moving these so that the tablets are now ordered by increasing number.
Which CS 125 sorting algorithm would you recommend if comparing tablets is easy but moving tablets is hard? Justify your answer.

2. Algorithm Analysis (20 Points)

For each snippet below list the *worst case* runtime using $\mathcal{O}$ notation (4 points each).

```
void foo1(final int N) {
    int i = N * N;
    int j = N / 2;
    while (i > 0) {
        i = i - N;
        j = j + (int) Math.sqrt(640000);
    }
}
```

(a) **4 points** `foo1` _____

```java
/*
 * The length of data is N. Initially called with idx = 1.
 */
void foo2(final int[] data, final int idx) {
    if (idx >= data.length) {
        return;
    }
    foo2(data, 2 * idx);
    data[idx] = idx * idx + data[idx - 1];
}
```

(b) **4 points** `foo2` _____

```java
/*
 * The length of data is N.
 */
void foo3(final double[] data, final double threshold) {
    for (int idx = 1; idx < data.length; idx++) {
        if (data[idx] > threshold) {
            for (int j = 0; j <= idx; j++) {
                data[j]--;
            }
        }
    }
}
```

(c) **4 points** `foo3` _____

```java
/*
 * mysteryFoo(N) scales as O(N log N)
 */
void foo4(final int N) {
    for (int idx = N; idx > 0; idx--) {
        mysteryFoo(N);
    }
}
```

(d) **4 points** `foo4` _____

```java
int foo5(final int N) {
    if (N < 2) {
        return N;
    }
    return foo5(N % 2) + foo5(N / 2);
}
```

(e) **4 points** `foo5` _____

(a) Chuchu in full color.

(b) Chuchu posterized in 10 colors.

Figure 1: Example of posterization.

3. Posterization (20 Points)

While photographs can contain many colors, certain situations require reducing the number of colors in an image to a much small number. For example, certain T-shirt printers print only one color at a time, making the cost scale roughly as $\mathcal{O}(N)$ with the number of colors. This process is sometimes referred to as *posterization*, since apparently at one point it was used to create posters.

To answer this question, describe an algorithm to posterize an image into N different RGB colors which are provided as inputs. The output image should consist *only* of pixels in the given set. **Write your answer for Question 3 on the next page.**

(a) **5 points** First, provide a written description of your algorithm in English. You will need to consider and carefully define what it means for two colors to be similar. We will accept multiple definitions of color similarity, but ensure that yours is well-motivated and clearly explained.

(b) **10 points** Next, provide an implementation of your algorithm in either Java or pseudocode. In contrast to your English description for the first part, your answer here should be able to be translated to valid Java code with minimal additional work. Feel free to use shorthand for extracting color channels from pixels, rather than the bit operations that you used for MP4. For example, you can write `pixel.red` to refer to the red channel value of `pixel`.

(c) **5 points** Finally, indicate the running time of your algorithm using $\mathcal{O}$ notation, where N is the number of pixels (width $\times$ height). If your algorithm could be made more efficient, describe how. If your algorithm could *not* be made more efficient, argue why not.

Answer for Question 3

Answer for Question 3

4. `checkstyle` (20 Points)

```
1  public class Whatevs {
2      private static final String empty = "";
3
4      public static void doIt(String[] wawa) {
5
6          String a = empty, b = empty;
7          for (String c : wawa)
8          {
9              if (c.length() > 1024) {
10                 continue;
11             }
12             if (myMethod(b, c)) a = b = c;
13             else if (!myMethod(c, b))
14             a = a.concat("\n").concat(c);
15         }
16         // ready to print now... whee...
17         System.out.println(a);
18     }
19
20     static boolean myMethod(String a, String b) {
21         // this is definitely going to work
22         return a.length() > b.length() ? true : false;
23     }
24 }
```

(a) **10 points** First, identify *five* style problems with the code above (2 points per problem.) Assume we are using the same Sun Java style guidelines that we have been using all semester on the MPs, and apply the same guidance about code quality that we have repeated on MPs and in labs.

Line Number	Problem

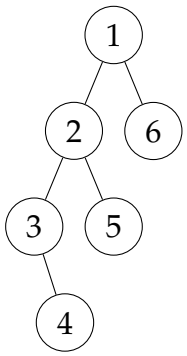
(b) **5 points** Next, rewrite the code to fix these problems. Your code should pass the `checkstyle` tests, and be concise and readable. Note that you *should not change the implementation* of the function—just fix the style errors.

(c) **5 points** Finally, in a few sentences explain why style guidelines are important and not just super annoying.

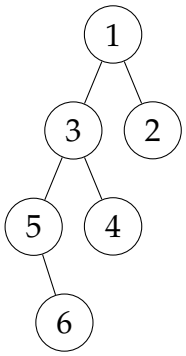
5. Tree Traversal (35 Points)

```
1  public class Tree {
2
3      /** Current node's parent. May be null if I'm the root of the tree. */
4      private Tree parent;
5
6      /** Current node's left child, or null if none. */
7      private Tree left;
8
9      /** Current node's right child, or null if none. */
10     private Tree right;
11
12     /** Current node's value. */
13     private int value;
14 }
```

Binary trees can be traversed in several different ways. A tree traversal visits every node in the tree in some order. One approach is known as *depth-first* traversal (DFT). It explores as far as possible along each branch in the tree before backtracking.



(2a) Visit order for DFT that visits left nodes first.



(2b) Visit order for DFT that visits right nodes first.

For example, assuming that a DFT visited all nodes in the following tree, they would be visited in the labeled order shown in Figure 2a. Note that these are not the *values* of the nodes—rather the order in which they are visited. This implementation visits left nodes first, but you could also visit right nodes first, at which point the order would be as shown in Figure 2b.

(a) **10 points** As a warm up, first, using our `Tree` data structure from MP6, write a recursive implementation of depth-first traversal. Descend to left nodes before right nodes, and print out the value of each node as you visit it.
Write a valid Java function to answer this question. You can assume that it would be run as an instance method of the `Tree` class defined above, and so can access its private instance variables. Use the instance that it is called on as the root to start your traversal.

(b) **20 points** Now, while recursion is a powerful programming technique, anything that can be done *with* recursion can also be done *without* recursion. Your next task is to (re)implement your depth-first traversal algorithm without using recursion. (No: your code cannot simply call your recursive implementation above. That counts as using recursion!) Follow the same guidelines as the previous part of the question: write a valid Java function, assume it is defined as a instance method, and use the instance it is called on to start your traversal.

You will need a data structure that maintains a list of `Tree` elements. You can assume it has the following interface—but note that you may not need all of these methods:

```
/*
 * Assume that first and last are valid initialized Tree objects.
 */

// Create a new empty list of Trees.
TreeList list = new TreeList();

// Prints true
System.out.println(list.empty());

// Add a value to the front of the list.
list.unshift(first);

// Prints 1
System.out.println(list.size());

/*
 * Remove and return the first value from the list.
 * Returns null if the list is empty.
 */
first = list.shift();

// Adds the value to the end of the list.
list.push(last);

/*
 * Remove and return the last value in the list.
 * Returns null if the list is empty.
 */
last = list.pop();
```

(c) **5 points** Finally, compare the operation of your recursive solution with your non-recursive solution. Specifically, you might want to describe what plays the role of the list data structure for the recursive implementation? Use the similarity to explain how anything that can be done with recursion can also be done without recursion.

Answer for Question 5

Answer for Question 5

6. Finding Friends (65 Points)

Write a Java class to track the location of your friends. It should allow you and your friends to move around and discover nearby friends. You can assume that this class only stores *your* friends, so every created Java friend object is a friend of yours. All data required by the problem should be stored by your new class, not somewhere else.

(a) **5 points** Your class should model your friends' name and location attributes. You should represent a friend's location as integer coordinates in two-dimensional space. Once created, a friend's name cannot change, but their position can.

(b) **10 points** You should provide at least two ways to create new friends (5 points each). One way allows you to create a new friend with a given name and location. The second allows you to create a new friend with a given name and at the same location as another friend.

Your friend class should enable the following functionality:

(c) **5 points** Friends should be able to update their location and move from place to place.

(d) **5 points** You consider two friends as equal if they are in the same location, even if their names are different.

(e) **10 points** You should be able to determine which friend is closest to you. You can compute distance as $d = Math.sqrt(x * x + y * y)$. Be sure to cast values appropriately to ensure that the return value is a floating point type.

(f) **10 points** You should be able to print out a list of all your friends and their current locations. You should implement this in part by utilizing the standard Java method allowing each friend to print their own name and location. However, this is not sufficient to print all of your friends.

(g) **20 points** Finally, provide a small example showing your class in action:

- Create several friends at different locations.
- Then they should all move to a new location.
- Create a new friend at the same location as one of your current friends.
- Now print all of their locations.
- Then, move yourself to the location of your closest friend.
- Verify that you and your friend are, at that point, equal to each other.

Write valid Java code for all your answers to this question. You will need to split your class definition across the pages that follow. Document your code carefully, and provide explanation as needed. However, you do not need to write your documentation in valid Javadoc format—although you can if you want.

Most of the points assigned to this problem will be for correctness, not for style. You do not need to worry about strictly following the `checkstyle` coding conventions. However, you should write clear and concise code, commented where needed.

Answer for Question 6

Answer for Question 6

Answer for Question 6

Answer for Question 6

Scratch. **Please indicate what question you are answering.**

Scratch. **Please indicate what question you are answering.**

Not Scratch. **Please do not use this as scratch space. Content here will not be graded.**

Congratulation on finishing CS 125!

It has been a pleasure teaching you this term. We hope that you have begun to appreciate the power and joy of computer science and computer programming. We wish you the best of luck with whatever the future holds. **Go forth and build good things!**

In the space below, please feel free to thank the course staff—including the teaching assistants and volunteers (doyens). Their hard work makes this course possible. And we hope that you will consider becoming a CS 125 volunteer yourself next semester!

BLANK