

Prelim 2

CS 2110, 24 April 2018, 7:30 PM

	1	2	3	4	5	6	7	Total
Question	Name	Short answer	Heaps	Tree	Collections	Sorting	Graph	
Max	1	16	10	20	11	18	24	100
Score								
Grader								

The exam is closed book and closed notes. Do not begin until instructed.

You have **90 minutes**. Good luck!

Write your name and Cornell **NetID**, **legibly**, at the top of **every** page! There are 6 questions on 8 numbered pages, front and back. Check that you have all the pages. When you hand in your exam, make sure your pages are still stapled together. If not, please use our stapler to reattach all your pages!

We have scrap paper available. If you do a lot of crossing out and rewriting, you might want to write code on scrap paper first and then copy it to the exam so that we can make sense of what you handed in.

Write your answers in the space provided. Ambiguous answers will be considered incorrect. You should be able to fit your answers easily into the space provided.

In some places, we have abbreviated or condensed code to reduce the number of pages that must be printed for the exam. In others, code has been obfuscated to make the problem more difficult. This does not mean that it's good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam. I will not talk about the exam with anyone in this course who has not yet taken Prelim 2.

(signature)

1. Name (1 point)

Write your name and NetID, **legibly**, at the top of **every** page of this exam.

2. Short Answer (16 points)

(a) **True / False (8 points)** Circle T or F in the table below.

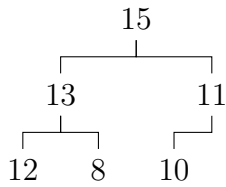
(a)	T	F	All search algorithms take $O(n^2)$ time in the worst case.
(b)	T	F	If a class implements <code>Iterator</code> , it must also implement <code>Iterable</code> .
(c)	T	F	It is best to store sparse graphs in an adjacency matrix and dense graphs in an adjacency list.
(d)	T	F	<code>ArrayList<Node></code> is not a subtype of <code>ArrayList<Object></code> even if <code>Node</code> extends <code>Object</code> .
(e)	T	F	An algorithm's time complexity can be both $O(n^3)$ and $O(n^2)$.
(f)	T	F	<code>ArrayList<double> myList= new ArrayList<>();</code> is valid Java.
(g)	T	F	Consider an undirected graph with set of edges E and set of vertices V . It is a tree iff $ E = V - 1$.
(h)	T	F	Any planar graph can be topologically sorted.

(b) **GUI (3 points)** What three steps are required to listen to an event in a Java GUI?

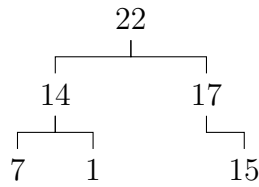
(c) **Red-Black Trees (5 points)** What is the full red-black tree invariant?

3. Heaps (10 Points)

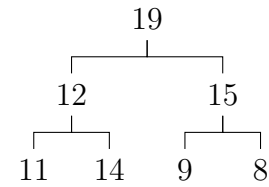
(a) **6 points** State whether each tree below is a valid max-heap, in which the priorities are the values. If it is not, state which invariant is unsatisfied.



1.



2.



3.

(b) **4 points** The int array `b` below corresponds to a min-heap of integers in which the values are the priorities. Suppose the heap has size 7 and `b[0..6]` contains these 7 integers.

`b = [3 8 4 9 11 5 10]`

Draw the state of `b` after calling `b.poll()`. Give your solution as an array.

4. Trees (20 Points)

(a) **2 points** What is the worst case time complexity for determining if a value is contained in:

- a Red Black Tree with n nodes?
- a Binary Search Tree with n nodes?

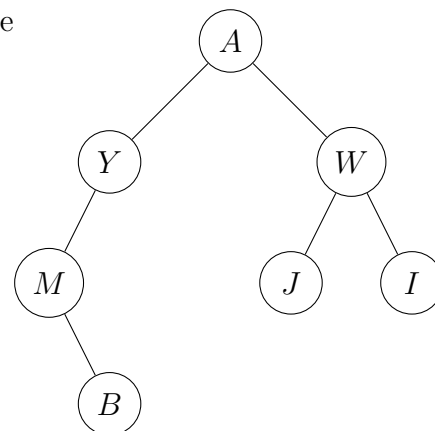
(b) **4 points**

Write down the order in which this tree's nodes are printed in preorder, inorder, and postorder traversals.

Preorder:

Inorder:

Postorder:



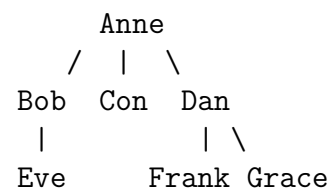
(c) 4 points Draw the binary search tree that results from inserting the following nodes, one by one, into an initially empty tree: 7, 4, 5, 1, 9, 8.

(d) 10 points

Class Ppl (for People), to the right, represents a person and their kids. In the tree to the right, Anne is the first generation. Her kids are Bob, Con, and Dan; they are Anne's generation 2. Bob has 1 kid, Eve, and Dan has 2 kids, Frank and Grace. Eve, Frank, and Grace are Anne's generation 3. Et cetera.

Complete method `maxGen` below.

```
public class Ppl {
    private String name;
    private Set<Ppl> kids;
    ...
}
```



```
/** Return the maximum number of generations that this Ppl can trace
 * forward through their kids.
 *
 * For example, in the tree to the right above,
 * Anne.maxGen() = 3 // Tree contains some of Anne's kid's kids
 * Bob.maxGen() = 2 // Bob's Tree contains Bob's kid
 * Con.maxGen() = 1 // Con's tree contains only Con
 * Frank.maxGen() = 1 // Frank's tree contains only Frank */
public int maxGen() {
```

```
}
```

5. Collections and Interface (11 Points)

Class **Multiset**(E), declared below, implements a multiset —like a set but elements can be in it more than once. Implement methods `contains`, `add`, and `remove`. Assume all other methods required by `Collection` have already been implemented.

```
/** An instance is a multiset --an unordered collection in which there can be
 * multiple instances of the same element. */
public class Multiset<E> implements Collection<E> {
    private int size;                // Number of elements in this multiset
    private Map<E, Integer> elemNum; // Mapping of element in this multiset to
                                    // its number of occurrences (which is > 0)

    /** Constructor: An empty multiset. */
    public Multiset() { elemNum= new TreeMap<E, Integer>(); }

    /** Return whether this multiset contains ob. */
    public boolean contains(Object ob) {

    }

    /** Add e to multiset. Return whether elemNum was modified in any way. */
    public boolean add(E e) {

    }

    /** Remove ob from the multiset if present. Return whether it was removed. */
    public boolean remove(Object ob) {

    }

}
}
```

6. Sorting (18 Points)

(a) **4 points.** Consider the following class `Author`. Order the list of `Author` in decreasing order of `booksWritten`. If two authors have written the same number of books, order by decreasing number of copies sold. Complete method `compareTo(...)`.

```
/** An instance represents a comparable author object*/
public class Author implements Comparable<Author> {
    private String name;
    private int booksWritten;
    private int copiesSold; // total number of copies of all the books sold.
    ...
    /** Compare this with ob as in standard compareTo specification. */
    @Override public int compareTo(Author ob) {

    }
}
```

(b) **6 points.** Using method `compareTo()`, complete method `selectionSort()`, below. That is, complete the comment that begins the body of the for-loop as a high-level statement saying what is done to ensure that the loop invariant remains true. Then implement the high-level statement.

```
public void selectionSort(Author[] b) {
    // inv: b[0..i-1] is sorted
    for (int i= 0; i < b.length; i= i+1) {

    }
}
```

(c) **2 points.** What is the worse-case time and expected time of selection sort?

(d) **6 points** State the tightest expected *time* complexity of quicksort, mergesort, and insertion-sort. For quicksort, assume it is the version that reduces the space as much as possible.

quicksort:

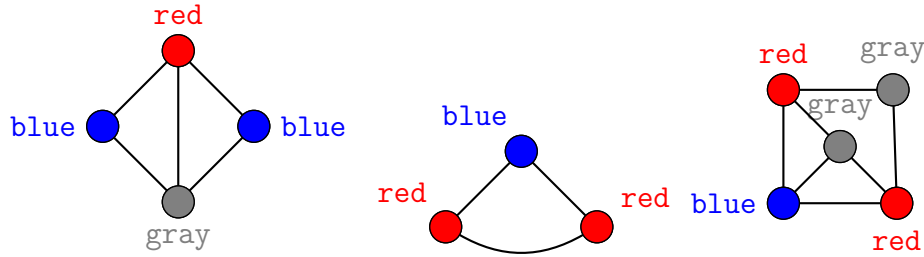
mergesort:

insertionsort:

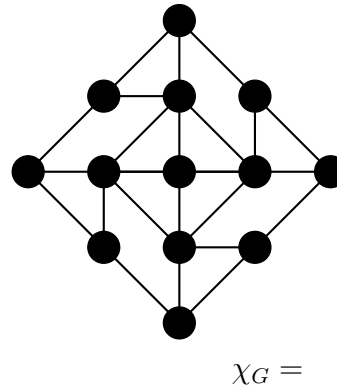
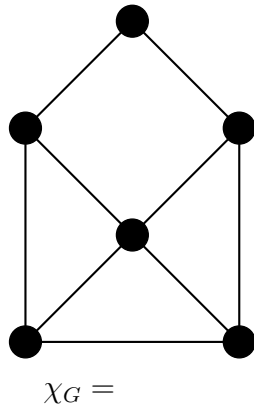
7. Graphs (24 Points)

A *coloring* of a graph is an assignment of colors to the nodes of a graph. An edge is *properly colored* if the two nodes it connects have different colors. A *proper coloring* is a coloring in which all edges are properly colored.

(a) 3 points Which of the following are proper colorings?



(c) 4 points The *chromatic number* of a graph G , written as (χ_G) , is the minimum number of colors needed to properly color the graph. For each of the graphs below, write down χ_G and properly color the graph using χ colors. To assign a color to a node, write the name of the color next to the node, as in part (a).



(d) 4 points Draw a graph whose chromatic number is 2 (there exists a proper coloring using only two colors). Use at least 4 nodes.

(e) 7 points Complete method `isProper`, below. The graph is undirected and connected. Starting with all nodes unchecked and `u` some node of the graph, the call `isProper(u)` will determine whether the graph is properly colored.

Use the English phrases `n was checked` and `check n` to test whether a node `n` has already been traversed and to mark it as checked, respectively. Use `n.color` for the color of node `n`. Use a loop for each neighbor `m` of `n`.

```
/** Let E be the set of edges that are reachable along unchecked paths from n.
 * Return true iff all edges in E are properly colored.
 * Precondition: all edges leaving a checked node are properly colored.
 * Precondition: n is unchecked. */
static public boolean isProper(Node n) {

}
}
```

(f) 1 point. Method `isProper` traverses the graphs in some fashion. It is similar to one of the following. Circle the one to which it is similar.

kruskal. prim. shortest path. dfs. bfs.

(g) 5 points. Two parts of the invariant of the shortest path algorithm are:

- For `f` in the frontier set, `d[f]` is the length of the shortest path to `f` that consists of red nodes except for `f`.
- There are no edges from the settled set to the far-off set.

State (1) the third part of the invariant and (2) the theorem that is proved about the invariant.