Prelim 2 Solution

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. False. It's easy to write one that takes more.
(b)	T	F	If a class implements Iterator, it must also implement Iterable. False. Iterator is typically implemented by a private inner class, not the same class.
(c)	T	F	It is best to store sparse graphs in an adjacency matrix and dense graphs in an adjacency list. False. Storing a sparse graph in an adjacency matrix wastes space and makes processing the neighbors of a node take more time.
(d)	T	F	`ArrayList<Node>` is not a subtype of `ArrayList<Object>` even if `Node` extends `Object`. True. See a line in the entry for "generics" in JavaHyperText.
(e)	T	F	An algorithm's time complexity can be both $O(n^3)$ and $O(n^2)$. True. Any algorithm that is $O(n^2)$ is also $O(n^3)$.
(f)	T	F	`ArrayList<double> myList= new ArrayList<>();` is valid Java. False. Type parameters cannot be primitive types.
(g)	T	F	Consider an undirected graph with set of edges E and set of vertices V. It is a tree iff $\|E\| = \|V\| - 1$. False. There must also be no cycles.
(h)	T	F	Any planar graph can be topologically sorted. False. The graph must be acyclic.

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

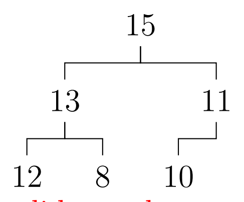
1. Have some class C implement an interface IN that is connected with the event.
2. In class C, override methods required by interface IN; these methods are generally called when the event happens.
3. Register an object of class C as a listener for the event. That object's methods will be called when event happens.

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

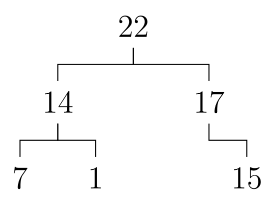
1. The tree is a binary tree.
2. Every node is either red or black.
3. The root is black.
4. If a node is red, then its (non-null) children are black.
5. For each node, every path to a descendant null node contains the same number of black nodes.

3. Heaps (10 Points)

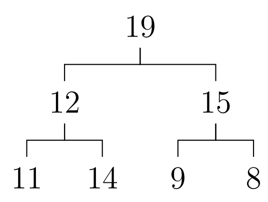
(a) 6 points State whether each tree below is a valid max-heap, in which the priorities are the values. If it is not, state which invariant is unsatisfied. No partial credit given.



valid max-heap



invalid: completeness invariant unsatisfied



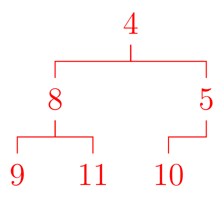
invalid: heap invariant unsatisfied

(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. (Points deducted if you did not write your solution as an array.)

[4 8 5 9 11 10]

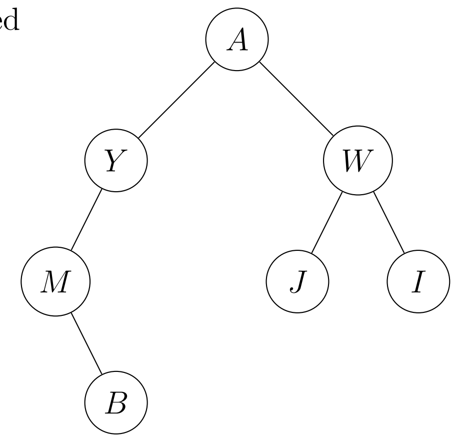


4. Trees (20 Points)

(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? O(log n)
- a Binary Search Tree with n nodes? O(n)

(b) 4 points
Write down the order in which this tree's nodes are printed in preorder, inorder, and postorder traversals.
Preorder: A Y M B W J I
Inorder: M B Y A J W I
Postorder: B M Y J I W A



(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. Two pts. off for a mistake; 0 if your answer does not resemble the correct tree.

```
    7
   / \
  4   9
 / \  /
1   5 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. Etcetera.

Complete method `maxGen` below.

```
public class Ppl {
  private String name;
  private Set<Ppl> kids;
  ...
}
          Anne
        /  |  \
     Bob  Con  Dan
      |         | \
     Eve      Frank Grace
```

```
 /** 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() {
  int mGen= 1;
  for (Ppl s : kids) {
      mGen= Math.max(mGen, 1 + s.maxGen());
  }
  return mGen;




}
```

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. Note: interpreting size as the number of entries in the map instead of the number elements in the multiset is a mistake. Suppose I gave you a bag containing ten dimes. Would you say its size is 1 or 10?

```
/** 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. */  (1 point)
    public boolean contains(Object ob) {
        elemNum.get(ob) != null;
    }

    /** Add e to multiset. Return whether elemNum was modified in any way. */  (4 points)

    public boolean add(E e) {
        if (contains(e)) {.             OR   Integer s= elemNum.get(e);
            elemNum.put(e, map.get(e)+1);    if (s == null) elemNum.put(e, 1);
        } else {                             else elemNum.put(e, s+1);
           elemNum.put(e, 1);                size++;
        }.                                   return true;
        size++;
        return true;
    }

    /** Remove ob from multiset if present. Return whether it was removed.*/  (6 points)

    public boolean remove(Object ob) {
        Integer s= elemNum.get(ob);
        if (s == null) return false;
        if (s == 1) elemNum.remove(ob);
        else elemNum.put((E)ob, s-1);
        size--;
        return true;
     }
}
```

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(...); it must implement the standard compareTo specification.

```
/** 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 object with ob for order. */
    @Override public int compareTo(Author ob) {  // there are other solutions
       if (ob.booksWritten == booksWritten)
             return ob.copiesSold - copiesSold;
       return ob.booksWritten - booksWritten;
    }
}
```

(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. (2 points) Then implement the high-level statement. (4 points)
We deducted points for an inner loop that swapped b[k] and b[j] many times instead of performing one swap after the index of the min value is found. No regrade request on this will get points back.

```
public void selectionSort(Author[] b) {
       // inv: b[0..i-1] is sorted  and  b[0..i-1] <= b[i..] (using compareTo)
       for (int i= 0; i < b.length; i= i+1) {
           // Swap b[i] with the minimum of b[i..]
           int k= i;
           // inv: b[k] is min of b[i..j-1]
           for (int j= i+1; j < b.length; j= j+1) {
              if (b[k].compareTo(b[j]) < 0) k= j;
           }
           // Swap b[i] and b[k]
           int t= b[i];  b[i]= b[k];  b[k]= t;
      }
  }
```

(c) 2 points. What is the worse-case time and expected time of selection sort?
$O(n^2)$, $O(n^2)$

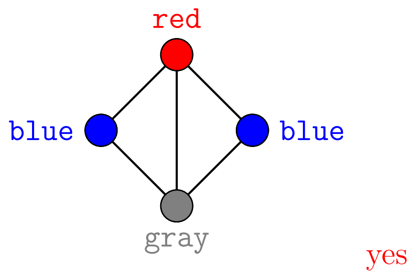
(d) 6 points State the tightest expected *time* complexity of quicksort, mergesort, and insertionsort. For quicksort, assume it is the version that reduces the space as much as possible.

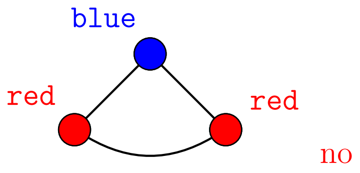
quicksort: $O(n\ log(n))$ mergesort: $O(n\ log(n))$ insertionsort: $O(n^2)$

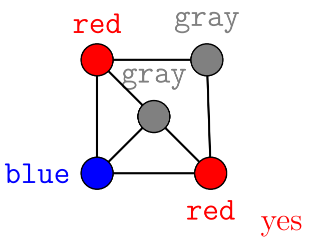
7. Graphs (24 Points)

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.

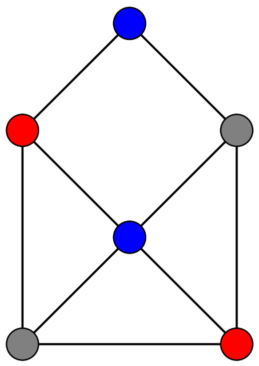
(a) 3 points Which of the following are proper colorings?



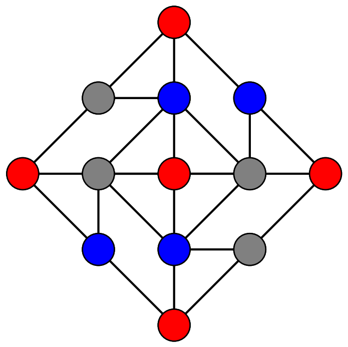




(c) 4 points The *chromatic number of a graph G*, written as (χ_G), is the minimum number of colors needed to properly color the graph. For each of the graphs below, write down χ_G and properly color the graph using χ colors. To assign a color to a node, write the name of the color next to the node,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



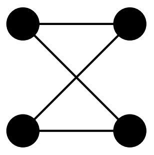
$\chi_G = 3$



$\chi_G = 3$

(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.

The graph must be a bipartite graph with at least four nodes.

For example: 

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`.
The precondition of any call implies that n should be checked only if its edges are properly colored. That is the reason for the first loop over the neighbors.

```
/** 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) {
    for each neighbor m of n {
        if (n.color == m.color) return false;
    }
    check n;
    for each neighbor m of n {
        if (m is unchecked  &&  !isProper(m)) return false;
    }
    return true;
 }
```

(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. The answer is dfs.

(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.
(1) For each node u in the settled set, d[u] is the length of the shortest path from the start node to node u.
(2) For f in the frontier set with minimum d-value (over nodes in the frontier set), d[f] is indeed the length of the shortest path from the start node to f.