NAME: ____ NETID: __________________________

CS2110 Spring 2015 Prelim 2.
Solutions
April 21, 2013. 7:30

	0	1	2	3	4	Total
Score	/1	/20	/44	/15	/20	
Grader						

1. True-false questions (20 points)

a	[✓ T]	F	Dijkstra's algorithm does not compute the "all pairs" shortest paths in a directed graph with positive edge weights because, running the algorithm a single time, starting from some single vertex x, it will compute only the min distance from x to y for all nodes y in the graph.
b	[✓ T]	F	Keyword **this** can be used only in non-static methods.
c	[✓ T]	F	A non-abstract class that implements an interface must provide definitions for all the methods of that interface.
d	[✓ T]	F	Some directed acyclic graphs have more than one topological sort.
e	T	[✓ F]	Using a hash table to implement a set of ints provides slower look-up time than using a binary search tree in practice.
f	T	[✓ F]	If the invariant of a loop is not true when the loop starts, its first iteration will make the invariant true.
g	[✓ T]	F	If the same element is inserted several times into a Java **Set**, the list will contain only a single instance of that element.
h	T	[✓ F]	Selection sort of an array of size n has worst-case running time O(n*n), but sometimes it will run in linear time.
i	[✓ T]	F	Breadth-first search uses a queue.
j	[✓ T]	F	Autoboxing refers to the process of creating an object of a wrapper class to wrap a primitive-type value.
k	[✓ T]	F	Prim's and Kruskal's algorithms may compute different minimum spanning trees when run on the same graph.
l	T	[✓ F]	If a graph is drawn so that edges cross, the graph is not planar.
m	T	[✓ F]	The worst-case complexity of inserting a value into a binary search tree containing N items is O(log(N))
n	[✓ T]	F	If Quicksort is written so that the partition algorithm always uses the median value of the segment as the pivot, then the worst-case performance is O(n log n).
o	T	[✓ F]	Dijkstra's shortest-path algorithm is more like a depth-first search than a breadth-first search.
p	T	[✓ F]	LinkedList<Integer> is a subclass of LinkedList<Object>, just as Integer[] is a subclass of Object[].
q	T	[✓ F]	Nested for loops always result in complexity $O(N^2)$
r	T	[✓ F]	Suppose that X and Y are different objects of type A and that X.Equals(Y) is false. Then X.hashcode() != Y.hashcode(). That is, different objects have different hash codes.
s	[✓ T]	F	The default layout manager for a JFrame is BorderLayout.
t	T	[✓ F]	An int local variable has default value 0; an int field also has default value 0.

2. Short-answer questions (30 points).

(a) (5 points) Write the definition of asymptotic complexity; that is write down when f(n) is O(g(n)) for functions f(n) and g(n).

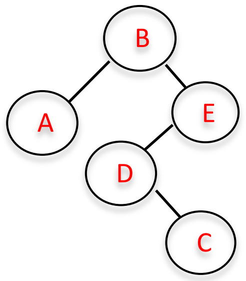
f(n) is O(g(n)) if there exist constants c > 0 and N > 0 such that
f(n) <= c*g(n) for all n >= N

(b) (5 points) Prove that m*m + 100*m is O(m*m)

Proof we start with f(m) and show that it is < c * g(n), finding c and N as we go

```
      f(m)
=         <use f(n) = m*m + 100*m>
      m*m + 100*m
<=        <if we use only m >= 100  —we can probably choose 100 for N>
      m*m + m*m
=         <m*m+ m*m = 2m*m>
      2m*m
          <choose c = 2, use g(m) = m*m>
      c*g(m)
```

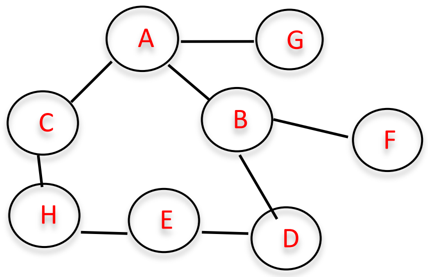
(c) (5 points) The inorder of a tree with 5 nodes is A B D C E. The preorder is B A E D C. Draw the tree:



(d) (5 points) Consider hashing using open addressing (not chaining, which uses linked lists). When removing an value, the array element it occupies is not set to null; instead, a flag is set to indicated that the element is no longer in the set. Explain why in one or two sentences (what would go wrong if we set the array element to null?)

If a removed element is set to null, then linear or quadratic probing will not work. Example. Suppose values a and b hash to the same value k, so that b[k..k+1] is {a, b}. Now suppose a is deleted and its array element set to null, so that b[k..k+1] is now {null, b}. Then a test whether b is in the set will look at b[k], find null, and conclude that b is not in the set.

(e) (5 points) List the sequence of nodes visited by BFS (the breadth-first search algorithm) starting at A of the following graph. When a choice of nodes has to be made, choose them in alphabetical order



Put the nodes in the order they are visited here: A B C G D F H E

(f) (5 points) For the graph in part (e), state the order of nodes visited by the recursive DFS (the depth-first search algorithm), starting at A. When a choice of nodes has to be made, choose them in alphabetical order.

Put the nodes in the order they are visited here: A B D E H C F G

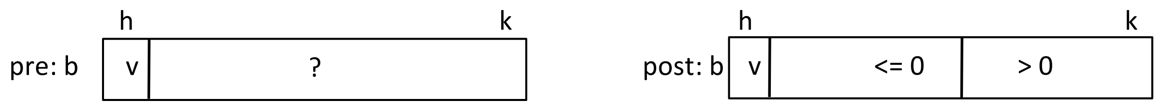
(g) (5 points) Insertionsort, to sort array c[0..n], has this invariant:

c[0..h-1] is sorted

Write the body of the loop. We expect it to be at the same level of abstraction as we have described it in lecture. We do *not* want to see an inner loop.

Push c[h] down to its sorted position in c[0..h].
h= h+1;

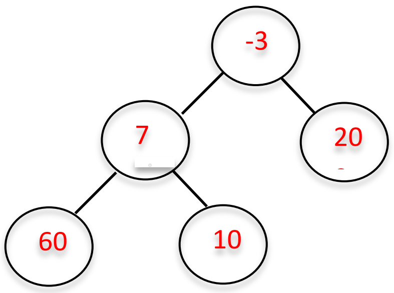
(h) (6 points) An array segment b[h..k] contains weights on edges of a graph. The weights can be negative, 0, or positive. We want a loop with initialization that swaps the values so that all the negative weights are to the left and all the positive weights to the right. Below is the pre- and post-condition for the loop. Below the pre- and post-condition, draw 3 of the 4 possible loop invariants that arise by combining the pre- and post-condition. *You do not have to put variables in to mark boundaries. You do not have to write the initialization or the loop*.



The answer is postcondition b but with a segment marked ? inserted in three different places. (1) between the v and <= 0 segment, between the <= 0 and > segments, and after the > 0 segment.

3. (15 points) Create a min-heap of ints (of maximum size eight) by inserting five values 60, 20, 10, 7, -3 in the order shown (leaving room for three more). Draw the resulting heap first as a tree (exactly as seen in class) and then as it is stored in an array.

(a) 5 points. The heap:



(b) 5 points. The corresponding vector (put a slash / through "empty" elements):

-3	7	20	60	10	/	/	/

(c) 5 points. Now show the vector after poll() is called once. Put the polled value here: -3

7	10	20	60	/	/	/	/

4. (20 points) Recursion and asymptotic complexity.
Please note the correct distribution of points on question 4: 6, 6, 5, 6.
(a) (6 points) Consider a doubly linked list in which nodes have fields *prev* and *next* pointing to the previous and next nodes in the list, respectively (or **null** if there is no such node). Unfortunately, the person who created the list inserted elements in the reverse order, so the value that should be last is now first in the list and vice versa. Write method *reverseList* below, whose use can rectify the situation. The method is initially called as reverseList(*head*, *tail*), where *head* and *tail* are the first and last nodes of the list, respectively. You can use the high-level statement "Swap values of b and c" to swap the values stored in nodes b and c. Your code should not contain any loops or change any *prev* or *next* fields.

```
/** If first !+ last, reverse the sequence of values in the list between nodes
  * first and last inclusive. */
public static void reverse(Node first, Node last) {
   if (first == last) return;   //This also takes care of an empty list, first and last being null
   Swap the values stored in first and last;
   if (first.next == last) return;   // Take care of case of a 2-node list
   reverse(first.next, last.prev);
}
```

(b) (6 points] For this question, consider trees whose nodes, of class TreeNode, contain three fields:
Suppose we have this function equals:

value:	the value at this node. Its type is some class ---not a primitive type
left:	left subtree (null if empty subtree)
right:	right subtree (null if empty subtree)

Write the body of function equals, given below.

```
/** Return true iff tree s equals tree t –meaning they have the same shape and same values
    in their corresponding nodes. */
public static boolean equals(TreeNode s, TreeNode t) {
    if (s == null || t == null) return  s == t;
    return  s.value.equals(t.value)  &&  equals(s.left, t.left)  &&  equals(s.right, t.right);
}
```

(c) (5 points) Suppose we use this function equals to test whether among M balanced BSTs there are any two that are identical. Assume each BST contains N nodes. What is the worst-case O() complexity of the resulting program, expressed in terms of N and M? Justify your answer by briefly explaining how you arrived at it.

The fact that the trees are balanced has nothing to do with the problem. In the worst case, it takes time O(N) to test where 2 trees with N nodes each are equal. In the worst case, O(M*M) pairs of trees have to be tested for equality. For example, let the trees be T1 to TM. Test T1 against T2..TM, then T2 against T3..TM, then T3 against T4..TM, each time finding they are not equal. Hence, the time is O(M*M*N).

(c) (6 points). Below is a version of breadth-first search, explained at a high level of abstraction. The specification is correct —it is what we want— but the algorithm has errors. Fix them. You may explain what is wrong below the algorithm and cross off things in it. Just keep things legible.

```
/** Node u is unvisited. Visit all nodes that are REACHABLE from u. */

public static void bfs(int u) {

   stack b= {u};

   while b is not empty do:

         u= b.popFirst();

         visit u;

         for each edges (v, u) entering u:

           b.appdn(v);

    }

    visit node u

}
```

The method body should be as below:

```
Queue b= {u};
while b is not empty do:
   u= b.popFirst();
   if u is not visited:
         visit u;
         for each edge(u, v leaving u):
            if v unvisited: b.append(v);
```