Name: NetID:

Prelim 2 Solution

CS 2110, April 26, 2016, 7:30 PM

	1	**2**	**3**	**4**	**5**	**Total**
Question	True/False	Complexity	Heaps	Trees	Graphs	
Max	10	30	20	20	20	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** at the top of **every** page! There are 5 questions on 10 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 its good style.

Academic Integrity Statement: I pledge that I have neither given nor received any unauthorized aid on this exam.

(signature)

Name: NetID:

1. True / False (10 points)

Circle T or F in the table below.

a)	T	F	A method that computes a result in $O(n^4)$ time will always run slower than a method that computes a result in $O(n^2)$ time.
b)	T	F	If a graph with n nodes has at least $n-1$ edges, it is connected.
c)	T	F	The vertices of a finite **directed** graph can be topologically sorted if and only if the graph is acyclic.
d)	T	F	If a graph has thousands of vertices, each having at most 5 neighbors, you should implement the graph using an adjacency matrix instead of an adjacency list.
e)	T	F	`LinkedList<String>` is **not** a subtype of `LinkedList<Object>`, even though `String[]` is a subtype of `Object[]`.
f)	T	F	Method `m()` processes a list of size n using nested for-loops. Therefore, the runtime of `m()` can't be $O(n^3)$.
g)	T	F	In the worst case, heap-sort and selection sort have the same run time.
h)	T	F	If a graph is bipartite, it is planar.
i)	T	F	Suppose `a` and `b` are objects. If `a.equals(b)` evaluates to `true`, then `a.hashCode() == b.hashCode()` must evaluate to `true`.
j)	T	F	Suppose `a` and `b` are objects. If `a.equals(b)` evaluates to `false`, then `a.hashCode() == b.hashCode()` must evaluate to `false`.

Aren't these the same?

If a and b are equal, the hash code for the two must be the same or hashing won't work (see the lecture notes for more on why). However, if they are not equal, they still might end up with the same hash code. For example, if we have a class Point which stores x,y coordinates, and hashCode returns x - y, The points (0, 0) and (1, 1) will both have the same hash code (0) even though they are not equal.

2. Complexity (30 points)

(a) 4 points For each of the functions f below, state the function $g(n)$ such that $f(n)$ is $O(g(n))$. $g(n)$ should be as simple and tight as possible. For example, one could say that $f(n) = 2n^2$ is $O(n^3)$ or $O(2n^2)$, but the *best* answer, the answer we want, is $O(n^2)$.

(i) $f(n) = 2n * (\log n + n)$ $g(n) = n^2$

(ii) $f(n) = 2^n + 3000n + 4$ $g(n) = 2^n$

(iii) $f(n) = 9n \log n + \frac{n}{2}$ $g(n) = n \log n$

(iv) $f(n) = 100 + 5n$ $g(n) = n$

(b) 4 points Recall that we proved in recitation that $f(n) = n + 6$ is $O(n)$. In a similar manner, prove that $f(n) = 17n^2 + 2n^3$ is $O(n^3)$.

```
           17n^2 + 2n^3       (this is f(n)
    <=       <assume n >= 17>
           n*n^2 + 2n^3
    =        <arithmetic>
           3n^3               (this is 3g(n)
```

```
So, with N = 17 and c = 3, for all n >= N, f(n) <= c g(n)
Q.E.D. (meaning Quit, End, Done)
```

(c) 4 points What is the simplest and tightest time-complexity class of the following function? For example, if the function takes $2n^2$ time, write $O(n^2)$. Be careful! Consider the complexity of *all* operations and function calls below.

```
/** Reverse list in place. */
public static void reverse(ArrayList<String> list) {
    int n= list.size();
    for (int i= 0; i < n; i++) {
        list.add(0, list.remove(i));
    }
}
```

The runtime of this function is $O(n^2)$. Note that `list.add(0, list.remove(i))` is an $O(n)$ operation because `add()` is $O(n)$ and `remove()` is $O(i)$ for an `ArrayList`

(d) 6 points For each of the following tasks, state the expected and worst-case time complexity. If the expected and worst-case time complexities are different, describe a situation in which the task will take the worst-case time.

(i) (2 points) Use merge-sort to sort an array of size n.

Expected run time: $O(n \log n)$ Worst-case run time: $O(n \log n)$

(ii) (2 points) Search for a value in a sorted array of size n.

Expected run time: $O(\log n)$ Worst-case run time: $O(\log n)$

(iii) (2 points) Check if a binary search tree of size n contains a particular value.

Expected run time: $O(\log n)$ Worst case run time: $O(n)$
Suppose the binary search tree has all of it's `left` pointers equal to `null`. Then in the worst case, we have to traverse all n nodes to check if an element is present.

(e) 12 points Suppose we need to choose a data structure to store a changing collection of values. For each of the usage patterns listed below, mark an "X" in the following table to indicate which data structure serves that usage pattern *best* in terms of average-case time complexity. There is exactly one best data structure for each usage pattern.

Usage Pattern	`ArrayList`	`LinkedList`	`HashSet`	Balanced BST	Heap
(i)		X			
(ii)			X		
(iii)				X	
(iv)	X				
(v)	X				
(vi)					X

The usage patterns: `Note: By LinkedList we mean a singly linked list`

(i) (2 points) Removing certain elements while iterating through all elements (and preserving the ordering of the remaining elements).
(ii) (2 points) Checking if the collection contains various values
(iii) (2 points) Processing the elements in sorted order
(iv) (2 points) Adding and removing elements from indices clustered near the end of the collection
(v) (2 points) Retrieving elements at several unpredictable indices
(vi) (2 points) Processing the elements in a priority order

(i) Removing items from the middle of the list takes O(1) time in a LinkedList when you are already at the node, since you just swap some pointers. For an ArrayList removing an item from the middle could take O(n) time because later items have to be shifted back.

(ii) Expected time O(1) for a HashSet

(iii) An inorder traversal of a BST will give you sorted order in O(n) time, faster than any other option here.

(vi) "Priority" only make sense for heaps

(v) Basically the same as the answer for (iv)

(iv) Indices only makes sense for ArrayList/LinkedList. For a singly linked list, accessing items at the end takes O(n), while for an ArrayList it is O(1)

3. Heaps (20 points)

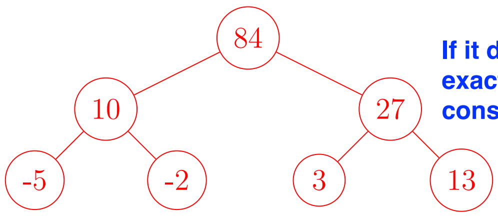
Consider creating a max-heap of `int`s by adding the following elements in the order presented:

-5, 10, 3, -2, 27, 13, 84

(a) 8 points Below, draw the resulting heap as a tree.
Note: If you provide a valid heap for these values but not the one that arises from adding the elements in order, you get **2 points** and can still get full credit for the subsequent problems.
Solution

Confused? Search for a Piazza post called "Prelim 2: Constructing a Heap".

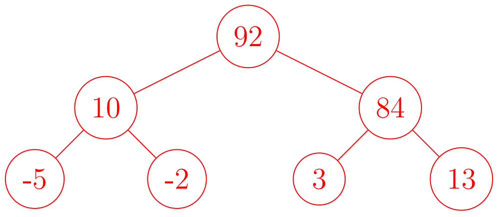
If it does not exist, create one with that exact name and ask how heaps are constructed.



(b) 6 points Draw the heap you provided as an array in the table below.
Solution

0	1	2	3	4	5	6
84	10	27	-5	-2	3	13

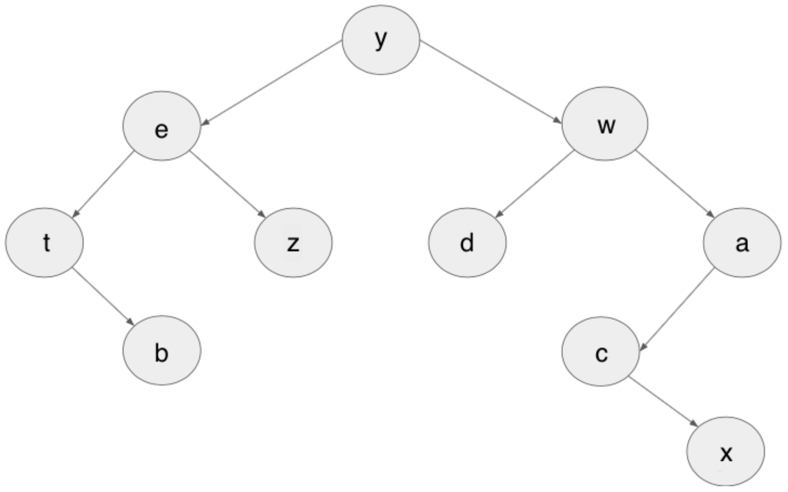
(c) 4 points Now we change the value of the element with value 27 to 92. Repair the heap you presented using the operations from class and draw the resulting heap as a tree below.
Note: If you provide a valid heap for these values but not the one that arises from performing this operation, you get **1 point** and can still get full credit for the subsequent problem.
Solution



(d) 2 points Draw the heap you provided as an array in the table below.
Solution

0	1	2	3	4	5	6
92	10	84	-5	-2	3	13

4. Trees (20 points)



(a) 2 points Is the tree above balanced?
The tree is **not** balanced

(b) 6 points Provide the pre-order, post-order, and in-order traversals of the tree above. Specifically, list the order in which the nodes of the tree are *processed* in each traversal. You should represent a node by the letter it contains, but please leave at least one space between letters for readability (e.g. a b c, *not* abc).

(i) (2 points) Pre-Order:
`y e t b z w d a c x`

(ii) (2 points) Post-Order:
`b t z e d x c a w y`

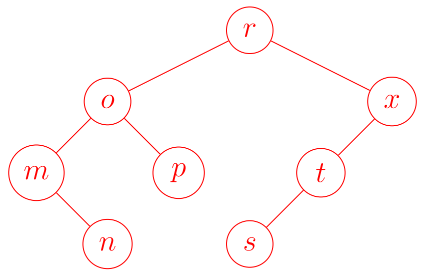
(iii) (2 points) In-Order:
`t b e z y d w c x a`

(c) 4 points Draw the BST (binary search tree) resulting from adding the following values one by one to an empty tree:

`r, x, o, t, m, s, p, n`

Note: If you provide a valid BST for these values but not the one that arises from adding the elements in order, you get **1 point**.

Solution

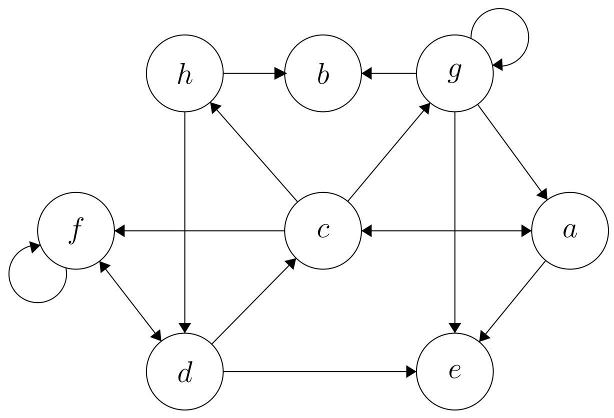


(d) 8 points Now that you have practiced adding values to a BST, implement method `add` of class `BSTNode` below.

```
/** An instance of this class represents a Binary Search Tree */
public class BSTNode {
  private int element; // the element of this BSTNode
  private BSTNode left; // left child of this BSTNode (null for an empty tree)
  private BSTNode right; // right child of this BSTNode (null for an empty tree)

  /** Constructor: a binary search tree containing only v */
  public BSTNode(int v) {
    element= v;
  }

  /** Add v to t and return t. If t is null return a new BSTNode containing v */
    * If v is already in t, do nothing and return t. */
  public static BSTNode add(int v, BSTNode t) {


    if (t == null) return new BSTNode(v);
    if (v == t.element) return t;
    if (v < t.element) {
      t.left = add(v, t.left);
      return t;
    }
    t.right = add(v, t.right);
    return t;


  }
}
```

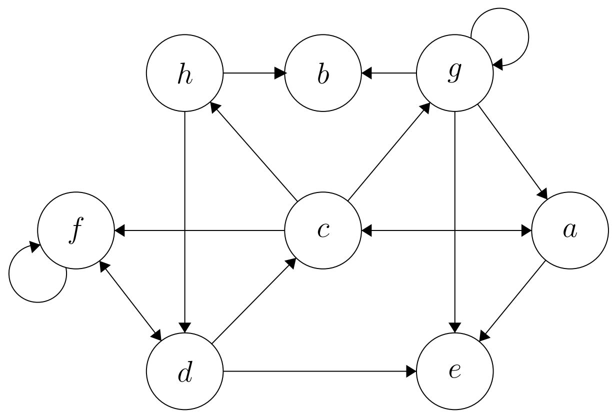
5. Graphs (20 points)

(a) 10 points You are given a (weighted) directed graph, duplicated in the first column below for each subproblem so that you may mark it up with notes (which will be ignored by the graders). For each subproblem, write your answer in the *second* column, NOT THE FIRST COLUMN! When you need to choose among vertices to visit first, choose in alphabetical order.



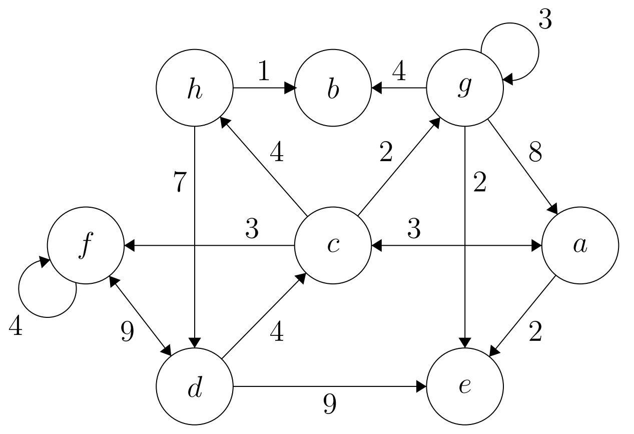
(i) (3 points) In what order are the nodes visited by DFS, starting from vertex a?

a c f d e g b h



(ii)(3 points) In what order are the nodes visited by BFS, starting from vertex a?

a c e f g h d b



(iii) (4 points) In what order are the nodes removed from the frontier in Dijkstra's algorithm, starting from vertex a?

a e c g f h b d

Questions? Go to piazza and search for a post named "P2 Spring 2016 7:30 5a iii". Ask your question there. IF the post does not exist, create one with this exact name. We don't want to answer the same question 30 times.

(b) 10 points The graph algorithms we have covered have applications beyond the problems for which they were originally intended. For example, although DFS and BFS are search algorithms, they are just ways to traverse a graph. So besides reachability, they can be used for distance calculation and cycle detection. And, topological sort can be used to determine shortest paths in DAGs.

For each problem listed below, mark an "X" in the following table to indicate which graph algorithm solves the problem *best* in terms of worst-case time complexity. There is exactly one best graph algorithm for each problem.

Problem	Topological sort	DFS	BFS	Dijkstra
(i)				X
(ii)			X	
(iii)		X		
(iv)	X			
(v)			X	

The problems:

(i) (2 points) You are on a bike trip in a park. Your map of the park shows you places to stop and rest, and you can calculate the distance between any two of them. You can bike only b miles without stopping at a resting place before you fall over, exhausted, and every mile takes you 10 minutes to bike. You are standing at the easternmost resting place. You want to determine how to most quickly reach the westernmost resting place without getting exhausted.

(ii) (2 points) You are intrigued by the concept of "six degrees of separation": any two people on the planet are connected through at most five intermediate acquaintances. You want to test this theory at Cornell. You have a database that tells you whether any two Cornell students are friends. You want to find a student who is "furthest away" from you, i.e. the minimum number of intermediate friends needed to relate that student to you is greater than or equal to that of any other student.

(iii) (2 points) You are a maze designer, and you want to make sure your preliminary design isn't too challenging. Write a program that, assuming there are no cycles, determines the length of the longest path from the maze's entrance that doesn't eventually lead to the maze's exit.

(iv) (2 points) You have a list of courses you want to take at Cornell. Some are prerequisites of others. You want to find a sequence in which to take these classes without violating any prerequisite requirement.

(v) (2 points) Ithaca has bicycle paths, but not on every road. For each pair of intersections A and B, your map of Ithaca tells you whether there is a bicycle path from A to B; call such a path a *direct bicicyle path*. You have to write a program to calculate the minimum number of direct bicicyle paths needed to get from one place in Ithaca to another.

(i) In this case, you have a graph of nodes (the oases) with edges (any path between nodes that is at most distance k) with weights (the distance between the nodes) and are trying to find the shortest path from one node to another. Dijkstra's shortest path algorithm is how we find shortest paths.

(ii) Breadth first search visits all nodes 1 away from the source, then all 2 away, then all 3 away, etc, so it can be used to determine which nodes are furthest away.

(iii) Depth first search allows us to keep track of how many nodes are on the path from the source to the current node, so we can use this information to find the longest path that doesn't lead to the node's exit.

(iv) Topological sort helps you find an order of the nodes such that all directed edges point forward. Since prerequisites can be modeled as arrows form the prereq to the course, topological sort will give us an order to take the courses where courses are always after their prereqs.

(v) We're trying to find the minimum number of steps from the source to some target. Since breadth first search visits nodes at distance 1, then 2, then 3, then 4, it is useful for finding the nodes with the fewest number of trips.