

Prelim 2 Solution

CS 2110, April 26, 2016, 5:30 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)

1. True / False (10 points)

Circle T or F in the table below.

a)	T	F	A bipartite graph is always planar.
b)	T	F	Suppose <code>a</code> and <code>b</code> are objects. If <code>a.equals(b)</code> evaluates to <code>true</code> , then <code>a.hashCode() == b.hashCode()</code> must also evaluate to <code>true</code> .
c)	T	F	Suppose <code>a</code> and <code>b</code> are objects. If <code>a.equals(b)</code> evaluates to <code>false</code> , then <code>a.hashCode() == b.hashCode()</code> must also evaluate to <code>false</code> .
d)	T	F	The vertices of a finite graph can be topologically sorted if and only if the graph is acyclic.
e)	T	F	In the worst case, insertion sort and quick-sort have the same run time.
f)	T	F	<code>HashSet<String></code> is a subtype of <code>HashSet<Object></code> , just as <code>String[]</code> is a subtype of <code>Object[]</code> .
g)	T	F	Method <code>m()</code> processes a list of size n using nested for loops. Therefore, the run time of <code>m()</code> is $O(n^2)$.
h)	T	F	If a graph will have thousands of vertices, each having at most 5 neighbors, you should implement the graph using a matrix rather than adjacency lists.
i)	T	F	A method that computes a result in $O(n)$ time will always run faster than a method that computes a result in $O(n^3)$ time.
j)	T	F	A connected graph with n nodes has at least $n - 1$ edges.

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 $O(n^2)$.

$$(i) \quad f(n) = 5 \log n + n \quad g(n) = n$$

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

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

$$(iv) \quad f(n) = 100 + 5\sqrt{n} \quad g(n) = \sqrt{n}$$

(b) **4 points** Recall that we proved in recitation that $f(n) = n + 6$ is $O(n)$. In a similar manner, prove that $f(n) = 3n^2 + 200n$ is $O(n^2)$.

$$\begin{aligned} & 3n^2 + 200n && \text{(this is } f(n)\text{)} \\ \leq & \quad \langle \text{assume } n \geq 200 \rangle \\ & 3n^2 + n \cdot n \\ = & \quad \langle \text{arithmetic} \rangle \\ & 4n^2 && \text{(this is } 4g(n)\text{)} \end{aligned}$$

So, with $N = 200$ and $c = 4$, for all $n \geq N$, $f(n) \leq c g(n)$
 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.

```
/** Return the max of the values in list.
 * Precondition: list.size() > 0 */
public static int max(LinkedList<Integer> list) {
    int max= Integer.MIN_VALUE;
    int n= list.size();
    for (int i= 0; i < n; i++) {
        int current= list.get(i);
        if (current > max) max= current;
    }
    return max;
}
```

The runtime of this function is $O(n^2)$. Note that `list.get(i)` takes $O(i)$ time for a `LinkedList`

(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 quick-sort to sort an array of size n .

Expected run time: $O(n \log n)$ Worst-case run time: $O(n^2)$

Suppose the smallest element in the array segment is selected as the pivot each time. Now the depth of the recursion is n with each level taking $O(n)$ time.

- (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 hash set of size n contains a particular value.

Expected run time: $O(1)$ Worst case run time: $O(n)$

Suppose all of the elements in the set hash to the same value. Now every element is in the same linked list (if implemented with chaining) or every check will need to probe every occupied slot (if implemented with open addressing) to conclude an element is not 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:

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

3. Heaps (20 points)

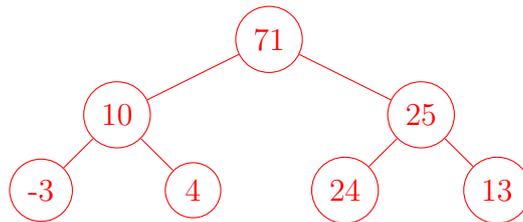
Consider creating a max-heap of `ints` by adding the following elements in the order presented:

24, 10, 25, -3, 4, 71, 13

(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



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

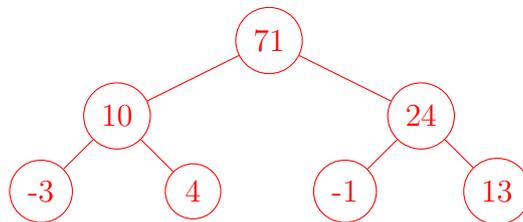
Solution

0	1	2	3	4	5	6
71	10	25	-3	4	24	13

(c) **4 points** Now we change the value of the element with value 25 to -1. 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

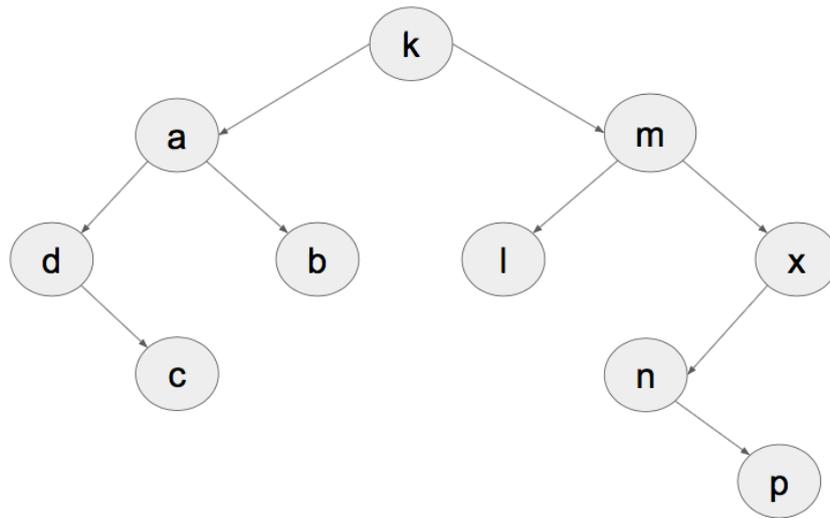


(d) **2 points** Draw the heap you provided as an array in the table below.

Solution

0	1	2	3	4	5	6
71	10	24	-3	4	-1	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:

k a d c b m l x n p

(ii) (2 points) Post-Order:

c d b a l p n x m k

(iii) (2 points) In-Order:

d c a b k l m n p x

Name:

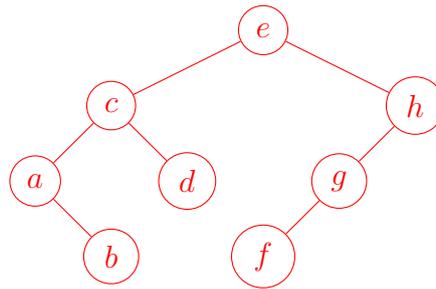
NetID:

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

e, h, c, g, a, f, d, b

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 value; // 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)

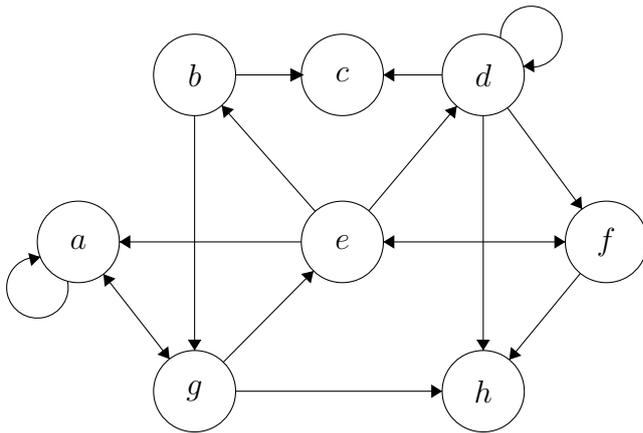
    /** Constructor: a binary search tree containing only e */
    public BSTNode(int e) {
        value= e;
    }

    /** Add e to this binary search tree.
     * If e is already in this tree, do nothing */
    public void add(int e) {

        if (e == value) return;
        if (e < value) {
            if (left == null) {
                left = new BSTNode(e);
                return;
            }
            left.add(e);
            return;
        }
        if (right == null) {
            right = new BSTNode(e);
            return;
        }
        right.add(e);
    }
}
```

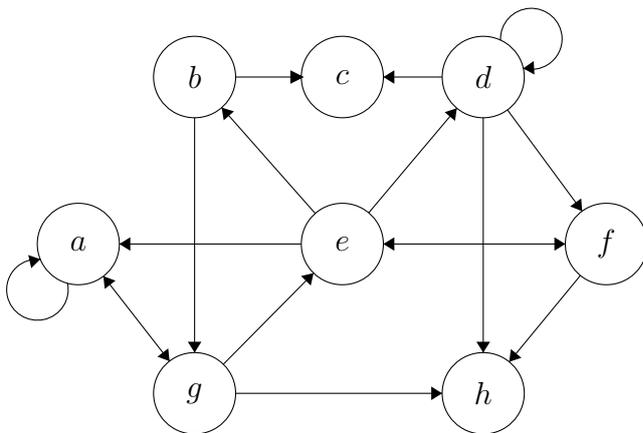
5. Graphs (20 points)

(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.



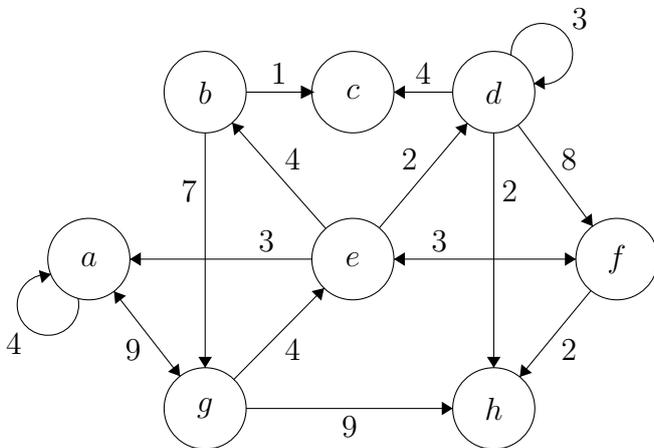
(i) (3 points) In what order are the nodes visited by DFS, starting from vertex *a*?

a g e b c d f h



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

a g e h b d f c



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

a g e d f b h c

(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 have a database of U.S. cities. For each pair of cities, it indicates whether a direct flight exists between them. You have to write a program to calculate the minimum number of connections needed to fly from one city to another. For example, you can't fly from Ithaca to San Francisco directly, but you can fly to Philadelphia and then to San Francisco, so the minimum number of connections is 1.
- (ii) (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.
- (iii) (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.
- (iv) (2 points) You are traveling on a desert. Your map gives the locations of oases, and you can calculate the distance between any two of them. You can walk only k miles without stopping at an oasis before you dehydrate, and every mile takes you an hour to hike. You are standing at the northernmost oasis. You want to determine how to most quickly reach the southernmost oasis without dehydrating.
- (v) (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.