Data Structures and Algorithms  
Dr. Gregory Safko

Rowan University

# Review: Final Exam

# Exam to take place on Monday, Aug 28

# Time: 9:30 to 11:40AM, James 2113

### Note: The focus of this exam will rely mostly on material from the second half of this term.

### In addition to knowing all of the topics so far, and all of the material discussed in lecture and lab, you should also know the following:

### General Knowledge

1. Know your quizzes, and what you answered incorrectly.
2. Know how your labs worked. Understand any errors along the way
3. Know (conceptually) how to add and delete elements from the four types of linked lists.
4. Know the difference between a stack and queue, and why you would use one over the other.
5. ~~Know the difference between sets, bags, and tuples.~~
6. Know your basic tree terminology
7. Know how to create and traverse trees.
8. Know how to recognize and write a recursive function
9. Know how to sort a list using: bubble sort, insertion sort, and selection sort
10. Know how to run a pre-, post-, and in-order traversal of a tree.
11. Know your basic algorithm efficiencies such as: O(n!), O(nn), O(2n) O(n2), O(n *lg* n), O(n), O(*lg* n)
12. Know how to analyze either a segment of code, or an algorithm, and apply the appropriate Big-O classification to it.
13. Know how to create heaps, how to remove elements, and how to heapify a complete tree.
14. Know how to create graphs. Know how to create a Cost Adjacency Matrix from a graph, and vice versa
15. Know how to examine a Cost Adjacency Matrix and determine if the graph is a digraph (directed graph)
16. Know how to use a Deque object

Sample Questions:

How can you recognize that an algorithm will have a Big-O of O(lg(n)) ?

* If the algorithm’s complexity is logarithmic then the algorithm grows exponentially; happens most often in binary search scenarios.

How can you recognize that an algorithm will have a Big-O of O(n lg(n)) ?

* If the algorithm’s complexity is linearithmic; happens most often in

sorting algorithms like merge sort and heap sort.

What algorithm has a Big-O of O(n3) ? O(n2.71) How are they related?

Given an array, an Array List, and a Linked List, all of size n, and the front is at index 0, and the back is at index size() -1, what is the Big O for each of these data collections given the following methods:

PushFront(val) O(n) usually, sometimes O(1)

Consists of creating a new node and updating references, can be done in a constant time.

PushBack(val) O(1) usually, sometimes O(n)

Adding an element to the back usually takes constant time unless a collection needs to be resized.

InsertInMiddle(index, val) O(n)

Requires the shifting of elements through traversal.

RemoveItem(val) O(n)

Requires traversal and shifting of elements.

FindItem(val) O(n)

Requires traversal of elements.

CountNegatives() O(n)

Requires traversal of elements.

A pre-order traversal of a tree gives the following list:

[400, 297, 148, 199, 362, 501, 620, 840, 711]

Draw the tree that represents this list

Here's the step-by-step process:

1. Start with the root node: 400

2. Move to the next element: 297 < 400, so it's the left child of 400.

3. Move to the next element: 148 < 297, so it's the left child of 297.

4. Move to the next element: 199 > 148, so 199 is the right child of 148.

5. Move to the next element: 362 > 297, so 362 is the right child of 297.

6. Move to the next element: 501 > 400, so 501 is the right child of 400.

7. Move to the next element: 620 > 501, so 620 is the right child of 501.

8. Move to the next element: 840 > 620, so 840 is the right child of 620.

9. Move to the next element: 711 < 840, so it's the left child of 840.

The resulting binary search tree:

400

/ \

297 501

/ \ \

148 362 620

\ / \

199 711 840

What are the leaf nodes of the tree you just drew? Can you make any inferences about how you can look at this list and determine the leaf nodes?

Leaf nodes are nodes with no children so: 199, 362, 711, 840

How would you write a recursive function that counts the leaf nodes, starting from the root of a binary tree?

public int countLeafNodes(TreeNode node) {

if (node == null) {

return 0;

}

if (node.left == null && node.right == null) {

return 1;

}

return countLeafNodes(node.left) + countLeafNodes(node.right);

}

Given a full array of integers called numbers of size n > 0, write code to output the index of the "first" negative number in the array. If there are not any negative numbers it should print -1.

For example if the array contains {125, 3, 45, -321, 99, -12} the output would be “3”.

If the array contains {22, 83, 21, 37, 61, 58, 12} the output would be “-1”.

Public int firstNegative()

{

for(int i =0; i < arr.size() – 1; i++)

{

if(i < 0)

{

int negative = i;

}

}

return negative;

}

Given the following UML

|  |
| --- |
| **Patient** |
| -name: String  -ID: int  -ailment:String  -Priority: int |
| +Patient(name:String, ID: int, ailment: String, Priority: int)  // setters & getters |

Create an ArrayList of type Patient. Index 0 = back, Index size() – 1 = front

ArrayList<Patient> pq = new ArrayList<>();

// Push patients, adjusting for priority

// Print ArrayList after each addition

Push patients on the array list. If their priority is less than the patient(s) ahead of them, move them up until they go to the position closest to the front without going past a patient with an even lower priority (Yes, this is a priority queue)

PushBack the following patients (in order):

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Id** | **Ailment** | **Priority** |
| Einstein | 32979 | Indigestion | 20 |
| Jack | 23231 | Broken Arm | 50 |
| Jill | 323216 | Concussion | 10 |
| Beethoven | 77735552 | Heart Attack | 1 |
| Curie | 920340 | Uranium Poisoning | 25 |
| Presley | 7777777 | Heart Attack | 1 |
| Von Neumann | 1234321 | Uranium Poisoning | 25 |

Print out the ArrayList after each added patient, to show that it works