OWLS Practice Final Solution Key
January 2022

This practice midterm has many questions for you to use to study for the exam. Note, this exam is way longer than the exam Professor Geitz is going to give, we have done that intentionally so that you have many questions to practice with.

Good Luck! You got this!

1. Here is a list of data: **[4,2,18,1,3,7,9,0,6]** For each of the following structures I will walk through the list in order, add each item to the structure and then go into a loop in which I remove elements one at a time from the structure and print them as I remove them. **In what order do I print the items for**

   (a) A stack
   Solution: 6,0,9,7,3,1,18,2,4

   (b) A queue
   Solution: 4,2,18,1,3,7,9,0,6

   (c) A priority queue
   Solution: 0,1,2,3,4,6,7,9,18

   (d) Forming a graph: each value is nodes, and there are **edges from any value to any value except itself that it divides evenly into**. In the removal stage I do a topological sort of the graph.
   Solution: Multiple Answers are correct! One such answer would be: 0,1,7,2,4,6,3,9,18

   (e) Forming a hash table of size 9 with linear open addressing using each data value as its own hash code (so the hash value is the remainder when we divide the value by 9). In the remove stage I remove and print the data at index 0, then the data at index 1, then index 2, etc.
   Solution: 18,1,2,3,4,9,0,7,6

2. Here is the start of a class that represents binary search trees that hold integer values:

```
class BST {
    Node root;
   class Node {
        int data;
        Node left, right;

        Node(int d) {
            data = d;
            left = null;
            right = null;
        }
    }
    BST( ) {
        root = null;
    }
}
```

As you can see, the BST class has a nested Node class; there is a constructor for Nodes. Give a BST method

boolean insert(int x)

If x is already in the tree your method should make no changes and return false. If x is not in the tree your method should insert it (preserving the Binary Search Tree property) and return true.

Solution

```
1. Start from the root.
2. Compare the inserting element with root, if less than root, then recurse for left,
else recurse for right.
3. After reaching the end, just insert that node at left(if less than current) else
right.
```

3. I have an ArrayList<String> and I want to sort it by increasing string length. The ArrayList class has a method sort (Comparator<String> comp). Give a class that implements the Comparator<String> interface that I can use to sort my ArrayList.

Solution:

```
class comp implements Comparator<String> {

    // override the compare() method
    public int compare(String s1, String s2)
    {
        if (s1.len() == s2.len())
            return 0;
        else if (s1.len() > s2.len())
            return 1;
        else
            return -1;
    }
}
```

4. I have a priority queue with 11 entries stored as a heap in the following array:

0	1	2	3	4	5	6	7	8	9	10	11	12
	10	15	20	30	40	25	20	30	40	45	40	...

The digits on top are the array indices. I want to poll the queue to remove its smallest value. What is the sequence of entries in the array after this removal? Your answer should look like this: 15 20 20 25 30 30 40 40 40 45, though that isn't the correct answer.
Solution: 15,30,20,30,40,25,20,40,40,45

5. I have n integer values stored in an ArrayList that I want to use as a priority queue. Instead of the heaps that we used in class, I am going to just sort the ArrayList and use that as the basis for my priority queue. For each of the following methods give an algorithm in English and a worst-case Big-Oh estimate of how long this takes.

   (a) **peek( ):** Return the smallest value in the sorted ArrayList without changing the list.
   Solution:
   Return index 0

   Big Oh: O(1)

   (b) **poll( ):** Remove and return the smallest value in the sorted ArrayList
   Solution;
   Return index 0
   Go through the each index, starting at 0 and setting ArrayList[i] = ArrayList[i+1]. When the last element is reached, set it to 0.

   Big Oh: O(n)

   (c) **offer(x):** Add element x to the sorted array list.
   Solution:
   Go through the Arraylist until value is larger than x.
   Save index to a variable
   Start at the back and move every element up one spot until the index is reached
   Insert x in front of the one that was bigger than it, which should be duplicated.

   Big Oh: O(n)

6. I have a directed graph with N nodes. How can I tell if the graph has a cycle? Your answer should be an algorithm in English. It is nice but not essential for you to give the name of the algorithm. You should describe how the algorithm works.
   Solution: Depth First Search

7. Oh, no! You fell asleep during the CS 151 final and now you find yourself inside a video game. You are in a gigantic house with many rooms and doors connecting the rooms. The name of each room is painted on the ceiling so you know where you are; you just have to find your way out. Instead of a map you have a file that lists all of the doors of the house; a line in the file such as

   Dining Room : Kitchen

   means there is a door connecting the Dining Room and the Kitchen. You need to find your way to the room called Exit. To make matters worse, all of the doors in the house are guarded by trolls. To go through a door you have to distract its troll by giving it a cookie. You have a small bag of cookies and no way to get any more. If you can find your way out before you run out of cookies, you will wake up in the wonderful Land of Internships. If you don't find your way out, those trolls look very hungry

   Give an algorithm, in English, for how to find your way out.
   Solution:

```
Create a Map of all rooms with an edge between rooms that have a door.
Use a shortest path algorithm to find the shortest path from start to exit
Take the shortest path and give each troll a cookie
```

8. In each part give a Big-Oh estimate of the worst-case time it takes to find a specific entry in the given structure:

   (a) An unsorted Linked List with n entries
   Solution: O(n)

   (b) A sorted Array List with n entries
   Solution: O(logn)

   (c) A sorted Linked List with n entries
   Solution: O(n)

   (d) A Binary Search tree with n entries
   Solution: O(n)

   (e) An AVL Tree with n entries
   Solution: O(logn)

   (f) A Heap (Priority Queue) with n entries
   Solution: O(n)

9. We might have a binary tree structure with integer nodes defined as:

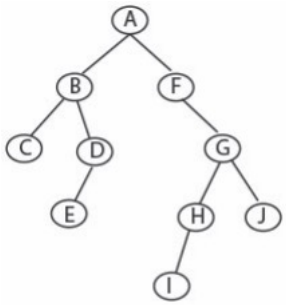
```
class Tree {
    int data;
    Tree left, right;
}
```

Let's assume missing or empty trees are represented as null and a leaf is a tree that has null left and right children. Write a method **int height( )** for this Tree class that returns the height of the tree rooted at the current node. Leaves should have height 0, a tree with leaves for both children should have height 1, and so forth. Note that this method is slightly different from the code you wrote for Lab 5 because there is no EmptyTree class; you have to handle the base case of the recursion yourself.

Solution:

```
int height( ) {
    if ((left == null) && (right == null))
        return 0;
    else if (left == null)
        return 1+right.height();
    else if (right == null)
        return 1+left.height();
    else
        return 1+Math.max(left.height, right.height);
}
```

10. Here is a picture of a binary tree with 10 nodes.



I am going to run an algorithm that uses a "structure" that might be a stack or a queue. I start by putting the root node A into the structure. Then at each step I remove the next node from the structure, print its letter, add its left child into the structure and then add its right child into the structure. This continues until the structure is empty.

(a) In what order are the nodes printed if the structure is a stack? An answer might look like "AGHBCDFEIJ" (though that's not the right answer).
Solution: AFGJHIBDEC

(b) In what order are the nodes printed is the structure is a queue?
Solution: ABFCDGEHJI

11. I want to build a Hashmap in an array of size 10 (so the array indices are 0 through 9). I will use linear open addressing to resolve collisions. Here is a table of 9 entries and their hash values:

Entry	A	B	C	D	E	F	G	H	I
Hash Value	0	8	9	6	1	3	7	5	6

I add the data in the order: D B C I G A E F H
List the entries in the order they appear in the array: index 0 first, then index 1, and so forth

Solution: G A E F H D I B C

12. We might have a Node class defined as

```
class Node {
    private int data:
    private Node next;
}
```

and a List class with an empty sentinel node Head at the front and a variable Tail pointing to the last node in the list.

Write a method List reverse(List L) that will build the reversal of List L. So if L starts as the list shown, reverse(L) will produce the list 30, 20, 10. It is acceptable to destroy L in the process of making its reversal (for example, you might want to make the Head node point to the node containing 30, and that node point to the one containing 20, and so forth.

Solution:

```
List reverse(List L) {
    if (L.size( ) < 2)
        return L;
    else {
        Node p = L.Head;
        Node q = p.next;
        Node r = q.next;
        q.next = null;
        Head.next = Tail;
        Node newTail = q;
        while (p != L.Tail) {
            if (p == L.Head)
                q.next = null;
            else
                q.next = p;
            p = q;
            q = r;
            if (r != null)
                r = r.next;
        }
        L.Head.next = L.Tail;
        L.Tail = newTail;
}
```

The following questions are not from Professor Geitz and are written by us or the previous OWLs!!

1. Below is a fill in the gap:
   (a) The java syntax "implements" is used for **interfaces** while "extends" is used for **abstract classes**
   (b) One main difference between an array and an ArrayList is **(Multiple Answers OK)[Array has a fixed length]...**
   (c) **Bellman Ford** algorithm can be used to find the shortest path in a graph with negative edge weights.
   (d) We perform **siftup** when we add an element to a heap and **siftdown** when we remove an element from a heap.
   (e) The **pop** operation is used to remove an element from a stack
   (f) The element removed from a queue is the **first** element added to the queue.
   (g) Dijkstra's algorithm produces a **shortest path**
   (h) If you're looking for the next empty location in an array and you search in a squared number sequence, then you are using a hashing technique known as **quadratic probing**
   (i) An IF statement with 3 nested for-loops in one of its conditions has a big-oh runtime $\mathbf{O}(n^3)$
   (j) Once the load factor exceeds its predefined value, the hash map must be **resized**

2. Put the following Big-O runtimes in order (Indicate if two are equivalent): $O((1/5)n), O(logn + n^2), O(log(n)), O(2^n), O(n^6), O(nlogn), O(x^n)$

Solution:
$O(logn), O((1/5)n), O(nlogn), O(logn + n^2), O(n^6), O(2^n), O(x^n)$
The last two are equivalent.

3. Write the Big-O runtime of the following functions:

(a)
```
public int size(){
    return size;
}
```

Solution: O(1)

(b)
```
public int fact(int n){
    int x = 1;
    for (int i = 1; i <= n; i++){
        x = x*i;
        if (n>1){
            n--;
        }
    }
    return x;
}
```

Solution: O(n)

(c)
```
public int ILoveOWLS(int n){
    int result = 0;
    while (n > 1){
        n = n//2;
        result += 1;
    }
    return result;
}
```

Solution: O(n)

(d)
```
public int IWillMissThisClass(int n){
    int result = 0;
    for (int i = 0; i < n; i++){
        for (int j = 0; j < n-1; j++){
            for (int k = 3; k < n-5; k++){
                result ++;
            }
        }
    }
}
```

Solution: O(n^3)

4. Give the expected Big-O runtime for the following operations using the given data structures. Ignore the cost of doubling the array, or consider it ammortized throughout all the add calls

Operation	ArrayList	Singly Linked List (with head pointer and no tail pointer)
add(0,x)	O(n)	O(1)
add(x)	O(1)	O(1)
get(i)	O(1)	O(n)
remove(0)	O(n)	O(1)

5. Stack vs Queue Assume that you are implementing a class that can use either a Stack or a Queue to manage a worklist of items that need to be processed. If the following method calls occur in the order given below, complete the table showing: 1) The changes being made to the worklist. 2) What the method call will return. If the method does not return anything, write "null".

Solution:

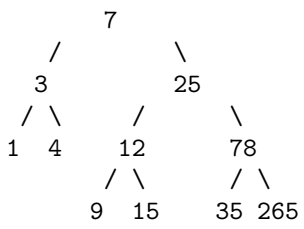
Stack:

Method	Worklist	Return value
push("a")	a	null
push("b")	ba	null
peek()	none	b
push("c")	cba	null
pop()	ba	c
pop()	a	b

Queue:

Method	Worklist	Return value
enqueue("a")	a	null
enqueue("b")	ab	null
head()	none	a
enqueue("c")	abc	null
dequeue()()	bc	a
dequeue()()	c	b

6. Draw the binary search tree that will result from inserting the following numbers in the following order: 7, 25, 12, 78, 35, 3, 1, 4, 15, 9, 265
   Solution:



7. Heaps

   (a) Describe the structure of a min-heap, how do each nodes compare to their children?
   Solution: The smallest element is at the root and each parent is smaller than their children.

   (b) Fill in the runtimes of the min-heap operations

      i. GetMin()
      Solution: O(1)

      ii. RemoveMin()
      Solution: O(1)

      iii. Insert()
      Solution: O(1)

   (c) Fill in the array index locations of each node (When relevant, relative to node

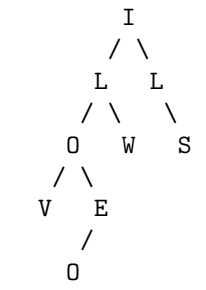
      i. The root
      Solution: A[0]

      ii. Node i's Left Child
      Solution: A[2i+1]

      iii. Node i's Right Child
      Solution: A[2i+2]

      iv. Node i's Parent
      Solution: A[(i-1)/2]

8. Show and label the results of inorder, postorder, level order, and preorder traversals of the tree below. Label which traversals are depth-first and which are breadth-first.



Solution:
Inorder: VOOELWILS
Postorder: VOEOWLSLI
Levelorder: ILLOWSVEO
Preorder: ILOVEOWLS

9. Why, in a binary search tree with n nodes, are there exactly n-1 different possible opportunities for single rotations? (Think about the structure of a rotation, whats needed?)

   Solution: The root doesn't have a parent to rotate on, which means we have one less opportunity to rotate for the entirety of our tree.

10. Give recursive algorithms that perform preorder and postorder tree traversals in O(n) time on a tree of n nodes.

    Solution: Call each algorithm on the root of the tree.

```
INORDER(x):
    if x != null
        INORDER(x.left)
        print(x.node)
        INORDER(x.right)

PREORDER(x):
    if x != null
        print(x.node)
        PREORDER(x.left)
        PREORDER(x.right)

POSTORDER(x):
    if x != null
        POSTORDER(x.left)
        POSTORDER(x.right)
        print(x.node)
```

11. ArrayList Implementation

(a) Implement the resize function to create a new array that is 3 times the length of the old one, copies all the data into the new one, changes this.data in the relevant way.
Solution:

```
Create a new array that is three times the size of the current one using data.length
Copy all of the old data into the new array
Set this.data to be your new array
```

(b) Implement a new function, shuffle that shuffles the array by swapping the first and last element, the n-2 and 1st element, and so on. Include the runtime of your function.
Solution: Put all the contents into a queue, dequeue one element at a time into a new array.

12. Give the Big-Oh for the following:

    (a) Bubble Sort:
    Solution: $O(n^2)$

    (b) Selection Sort:
    Solution: $O(n^2)$

    (c) Heap Sort:
    Solution: $O(nlogn)$

13. Sort the following list using merge sort [15, 6, 4, 22, 3, 11, 2, 9]. Show all of you steps.