University of Maryland, College Park
Dept. of Computer Science
CMSC132
Midterm #2

First Name (PRINT): _____

Last Name (PRINT): _____

University ID: _____

Section/TAName: _____

I pledge on my honor that I have not given or received any unauthorized assistance on this examination.

Your signature: _____

Instructions

- *This is a closed-book and closed-notes exam.*
- ***Total point value is 70 points.***
- *Thus is a 50 minute exam.*
- ***WRITE NEATLY**. If we cannot understand your answer, we will not grade it (i.e., 0 credit).*

Grader Use Only

Page 2	(15)	
Page 3	(15)	
Page 4	(10)	
Page 5	(8)	
Page 6	(10)	
Page 7	(12)	
Total	**(70)**	

1. (15 points) For each of the following operations, circle the algorithmic complexity as specified. Choose the smallest (best) category that you can. Do not make any assumptions about the data structures beyond what is explicitly stated.

   a. Search on an array. (Average case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   b. Search on a well-balanced binary search tree. (Average case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   c. Search on a well-balanced binary tree. (Average Case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   d. Search on a binary search tree. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   e. Search on a Hash Table with uniformly distributed elements. (Average case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   f. Search on a Hash Table. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   g. Accessing an element of an array by index number. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   h. Accessing an element of a linked list by index number. (Average case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   i. Insert at the front of an array. (The front is the side with index 0.) (Average case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   j. Insert at the head of a linked list. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   k. Add an element to a heap. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   l. Heap sort. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   m. Tree sort using a binary search tree. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   n. Remove the tail element of a doubly-linked list. (Worst case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

   o. Remove the tail element of a linked list (when maintaining a tail reference). (Average case.)

      O(1) O(log n) O(n) O(n * log n) $O(n^2)$

2. (4 points) For each of the following, choose the Java collection that *best* satisfies the requirement listed. (Assume that other considerations are less important.)

   a. You need very fast searches and don't care about anything else.

      ArrayList LinkedList HashSet LinkedHashSet TreeSet

   b. You need to iterate over the collection in sorted order.

      ArrayList LinkedList HashSet LinkedHashSet TreeSet

   c. After adding the elements, you will need to access them by their insertion positions. (E.g.: fast access to the element that was the 7^{th} one added or the 100^{th} one added.)

      ArrayList LinkedList HashSet LinkedHashSet TreeSet

   d. You want fast searches, but also will need to iterate over the elements in the order in which they were inserted.

      ArrayList LinkedList HashSet LinkedHashSet TreeSet

3. (2 points) Using the letters `h` for "hash code", `p` for "a large prime", and `t` for "table size", carefully state the formula that is used to determine the position of an element in a hash table.

4. (2 points) What data structure is most commonly used to implement a heap?

5. (1 point) What does MVC stand for?

6. (1 point) What does GUI stand for?

7. (1 point) Which of the three components in the MVC design pattern is *not* part of the GUI?

8. (2 points) Assume that `Fish` is a subclass of `Animal`.

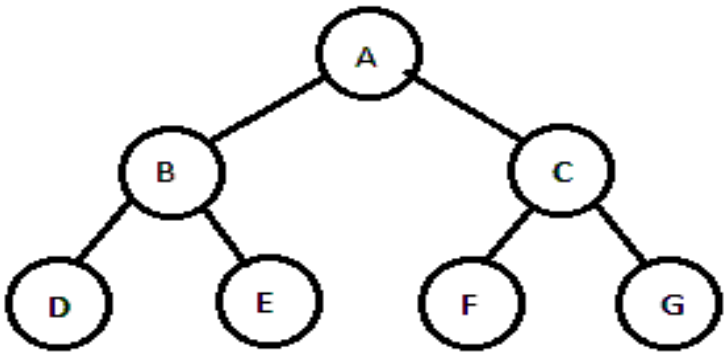
   a. `Fish[]` is a subclass of `Animal[]`. TRUE FALSE

   b. `ArrayList<Fish>` is a subclass of `ArrayList<Animal>`. TRUE FALSE

9. (2 points) If you choose to override the `equals` method, what other method must you also override?

10. (6 points) Using the theorem from class about limits, prove the correct relationship between $log_7 n$ and $log_{13} n$. (For full credit you must write the correct limit, evaluate the limit, and carefully state your conclusion.)

12. (4 points) Consider the binary tree below.



List the order in which the elements would be visited when performing a *post-order* traversal.

13. (8 points) Imagine that you start with an empty min-heap and add the following values one-by-one, in the order shown:

15, 20, 5, 12, 17

a. Draw the resulting heap as a TREE (not an array).

b. Show how the tree above would look after `removeSmallest()` is called once,

14. (10 points) Consider the class below, which represents a traditional linked list (using `null` to designate the end of the list).

```
public class LinkedList<T> {
   private class Node {
         private T data;
         private Node next;

         private Node(T data) {
               this.data = data;
         }
   }
   private Node head;
}
```

Write a method with the prototype you see below, which is part of the `LinkedList` class. The `value` must be inserted immediately after the first occurrence of the `target`. If the `target` is not in the list, then your method should do nothing. **You may NOT use recursion for this question.** Recursive solutions will earn 0 points.

```
public void insertValueAfter(T value, T target) {
```

15. (12 points) Consider the class below, which represents a traditional binary search tree (using `null` to represent non-existent children).

```
public class BinarySearchTree<K extends Comparable<K>, V> {
   private class Node {
         private K key;
         private V value;
         private Node left, right;
         public Node(K key, V value) {
               this.key = key;
               this.value = value;
         }
   }
   private Node root;
}
```

Fill in the instance method below, which is part of the `BinarySearchTree` class. The method will return the `value` of the node with the given `key`. If there is no node with this key, the method should return `null`. You may write an auxiliary helper method if you find it useful. **You must use recursion for this question.** If your solution is not recursive, you will receive 0 points. If your solution traverses extra nodes unnecessarily you will receive very few points.

```
public V get(K key) {
```