Points off	1	2	3A	3B	4	5	Total off	Net Score

CS 307 – Final – Fall 2010

Name_____

UTEID login name _____

Instructions:

1. Please turn off your cell phones and all other electronic devices.
2. There are 5 questions on this test.
3. You have 3 hours to complete the test.
4. You may not use a calculator on the test.
5. You may add helper methods if you wish when answering coding questions.
6. When answering coding questions assume the preconditions of the methods are met.

1. (1.5 point each, 30 points total) Short answer. Place you answers on the attached answer sheet.

For questions that ask "what is the output":

- If the code contains a syntax error or other compile error, answer “Compiler error”.
- If the code would result in a runtime error or exception answer “Runtime error”.
- If the code results in an infinite loop answer “Infinite loop”.

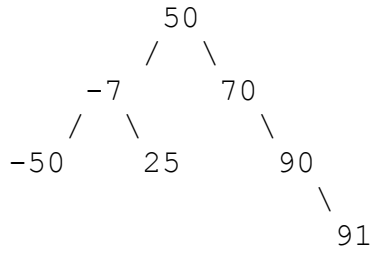
On questions that ask for the order (Big O) of a method or algorithm, recall that when asked for Big O your answer should be the most restrictive, correct Big O function. For example Selection Sort has is order $O(N^2)$, but per the formal definition of Big O it is correct to say Selection Sort is $O(N^3)$, $O(N^4)$, $O(2^N)$, and O(N!). Give the most restrictive, correct function. (Closest without going under.)

A. The following values are inserted one at a time in the order shown into an initially empty binary search tree using the traditional, naïve algorithm. Draw the resulting tree.

```
97   35   -59  93   2    95
```

B. What is the height of the resulting tree from part A? (Recall the height of a tree is the number of links from root to deepest leaf.)

Consider the following binary tree. `50` is the root of the tree.



C. What is the result of a pre-order traversal of the binary tree shown above?

D. What is the result of a in-order traversal of the binary tree shown above?

E. What is the result of a post-order traversal of the binary tree shown above?

F. Is the tree shown above a binary search tree?

G. Assume the tree shown above has all nodes colored black. (There are no red nodes in the tree.) Is the tree shown above a Red-Black tree? If not which Red-Black tree rule is violated?

H. What is the result of the following postfix expression?

```
5 2 + 100 200 * +
```

I. 2000 distinct (no repeats) integers that are in random order are inserted one at a time into a binary search tree using the traditional, naïve algorithm presented in class. What is the expected height of the resulting tree?

Give the **actual** number as you did on the experiments in assignment 11, NOT the order (Big O) of the height.

J. On the answer sheet, fill in the nodes of the given binary tree with integer values between 1 and 10 so that the tree is a binary search tree.

K. Consider the following methods:

```
public int myst(int[] data, int x) {
     return mystHelp(data, x, 0);
}

public int mystHelp(int[] data, int x, int y) {
     if(y == data.length)
          return 0;
     else if(data[y] == x)
          return 1 + mystHelp(data, x + 1, y + 1);
     else
          return mystHelp(data, x, y + 1);
}
```

What is output by the following code?

```
int[] vals = {4, -5, 5, 5, 4, 4, 6, 5, 3};
System.out.print( myst(vals, 4) );
```

L. What is the best case order (Big O) of the following method? N = `list.size()`.

```
public int alo(ArrayList<String> list, int tgt) {
     int result = 0;
     final int LIMIT = list.size();
     for(int i = 0; i < LIMIT; i++)
          if(list.get(i) != null && list.get(i).length() > tgt)
               result++;
     return result;
}
```

M. What is the best case order (Big O) of the following method? N = `list.size()`. `list` is a `LinkedList` from the Java standard library.

```
public int llo(LinkedList<Integer> list) {
     int result = 0;
     while(list.size() > 0) {
          result += list.get(0);
          list.remove(0);
     }
     return result;
}
```

N. What is the average case order (Big O) of the following method? The `BST` class uses the traditional, naïve algorithms for insertion and deletion of values.
N = `tree.size() = vals.length`

```
public int bst1(BST<Integer> tree, int[] vals) {
     int result = 0;
     for(int i = 0; i < vals.length; i++)
          if(tree.contains(vals[i]))
               result++;
     return result;
}
```

O. What is the worst case order (Big O) of the following method? The `BST` class uses the Red-Black tee algorithms for insertion and deletion of values.
N = `tree.size() = vals.length`

```
public int bst2(BST<Integer> tree, int[] vals) {
     int result = 0;
     for(int i = 0; i < vals.length; i++)
          if(tree.contains(vals[i]))
               result++;
     return result;
}
```

P. What is output by the following code?

```
Queue<Integer> q = new Queue<Integer>();
q.enqueue(3);
q.enqueue(1);
q.enqueue(2);
for(int i = 0; i < 5; i++) {
     int x = q.dequeue();
     q.enqueue(x);
     q.enqueue(x);
}
int total = 0;
while(!q.isEmpty())
     total += q.dequeue();
System.out.print(total);
```

Q. What is output by the following code?

```
Stack<Character> st = new Stack<Character>();
String name = "wirth";
int other = name.length() - 1;
for(int i = 0; i < name.length(); i++){
     st.push(name.charAt(0));
     st.push(name.charAt(other));
     other--;
}
while(!st.isEmpty())
     System.out.print(st.pop());
```

R. Consider the following method:

```
public void removeMany(Collection<Integer> col, int[] data) {
     for(int i = 0; i < data.length; i++)
          col.remove(data[i]);
}
```

Recall the `remove` method from the `Collection` interface:

> `boolean remove(Object o)`
> Removes a single instance of the specified element from this collection, if it is present.

The `Collection` named `col` contains N elements and the `int` array `data` contains N elements. Assume none of the elements in `data` are present in `col`.

Given the following timing data for method `removeMany` what type of `Collection` is being passed? Choose from `LinkedList`, `TreeSet`, and `HashSet`.

N	Time
100,000	1 second
200,000	4 seconds
400,000	16 seconds
800,000	64 seconds

S. Given this method:

```
public void manipObjects(Object ob1, Object obj2)
```

Briefly explain why the following method calls do not cause syntax errors:

```
manipObjects("Olivia", "isabelle");
manipObjects(new LinkedList<Integer>(), "kelly");
manipObjects(new ArrayList<Integer>(), new Rectangle());
```

T. What is output by the following code?

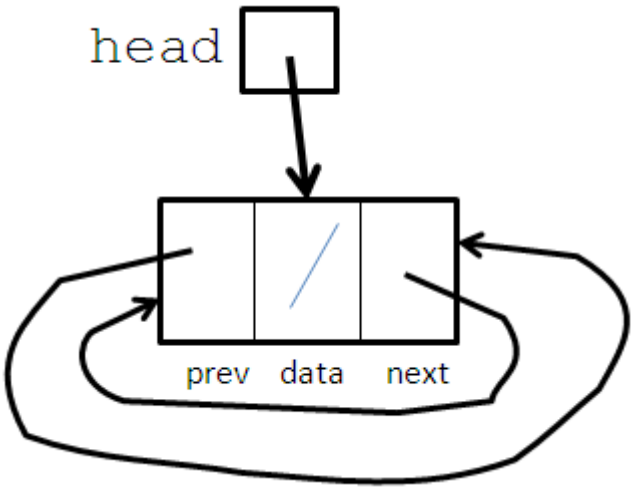
```
TreeSet<Integer> ts = new TreeSet<Integer>();
int val = 213555122;
while(val > 0) {
     ts.add(val % 10);
     val = val / 10;
}
for(int x : ts)
     System.out.print(x);
```

2. (Linked Lists, 15 points) Complete an instance method to remove the last occurrence of an element from a linked list.

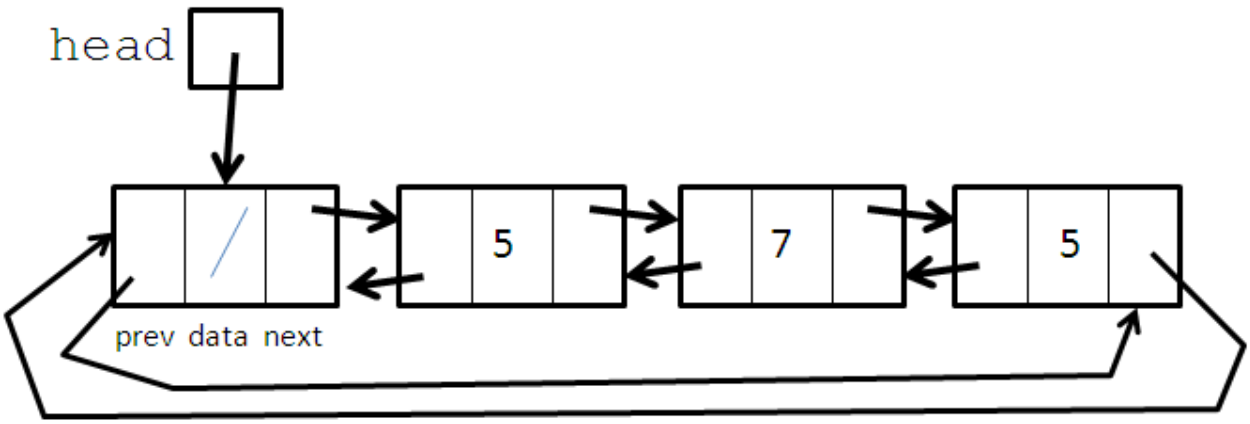
The linked list has the following properties:

- The nodes are doubly linked with references to the next node in the list and the previous node in the list.
- The `LinkedList` class keeps a reference to a dummy header node. The node does not contain any data and the instance variable `head` always refers to the same node.
- The list is circular. The previous reference of the dummy header node always refers to the last node in the list. The next reference of the last node of the list always refers to the dummy header node.
- The `LinkedList` class does not have an instance variable for its size.
- The `LinkedList` class achieves genericty by using Java's generics.

The structure of the internal storage container for an empty `LinkedList` is shown below. (Arrows indicate the references stored by variables.)



The structure of the internal storarage coinater for a list that contains the values `[5, 7, 5]` is shown below: (Nodes actually store references to the data objects, but the data is shown inside the node for clairty.)



Complete the method `boolean removeLast(E obj)` for the `LinkedList` class.

Imporant: Your method must be O(1) space meaning you can not use an auxiliary native array, `ArrayList`, or other data structure. Your method shall be O(N) time where N is the number of elements initially in the `LinkedList`. You may not use any other methods from the `LinkedList` class unless you write them yourself as part of your answer.

Examples of results of calls to `removeLast(E obj)` as seen by a client of the `LinkedList` class: (The return value is shown as well as the list after the call is completed.)

```
[A, B, A, C, B].removeLast(B) -> returns true, [A, B, A, C]
[A, B, A, C, B].removeLast(A) -> returns true, [A, B, C, B]
[A, B, A, C, B].removeLast(C) -> returns true, [A, B, A, B]
[A, B, A, C, B].removeLast(D) -> returns false, [A, B, A, C, B]
[].removeLast(A) -> returns false, []
[A].removeLast(A) -> returns true, []
[A, A, A].removeLast(A) -> returns true, [A, A]
```

The `LinkedList` class uses the following node class:

```
public class Node<E>{
    public Node(Node<E>, prev, E value, Node<E> next)

    public Node<E> getNext()
    public Node<E> getPrev()
    public E getValue()

    public void setNext(Node<E> newNext)
    public void setPrev(Node<E> newPrev)
    public void setValue(E obj)
}
```

Here is the `LinkedList` class.

```
public class LinkedList<E>{

    // Reference to the dummy header node
    private Node<E> head;

    public LinkedList() {
        head = new Node<E>();
        head.setPrev(head);
        header.setNext(head);
    }
```

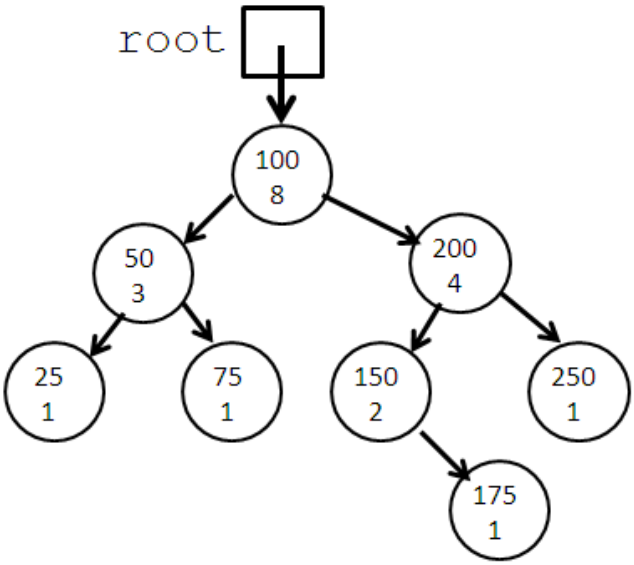
Complete the following `LinkedList` instance method:

```
// pre: obj != null
// post: Removes the last occurrence of obj in this list.
//     Returns true if this list changed as a result of this
//     method call, false otherwise.
public boolean removeLast (E obj) {
```

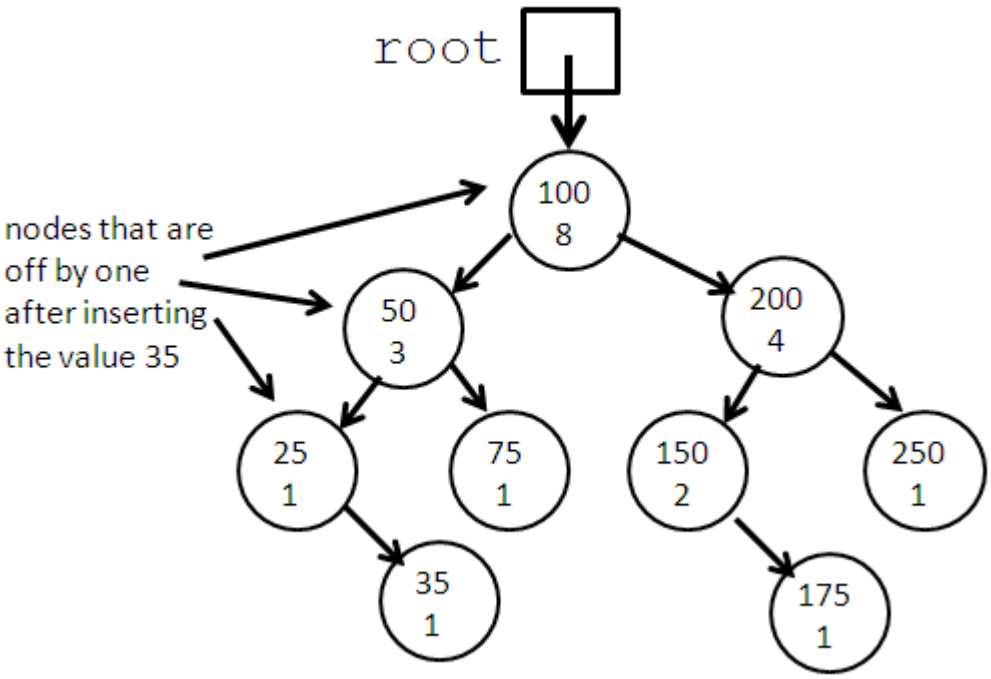
3. (Binary search trees, 25 points) This question has two parts.

An augmented data structure is a traditional data structure that has extra information added to make new operations more efficient or easier to implement.

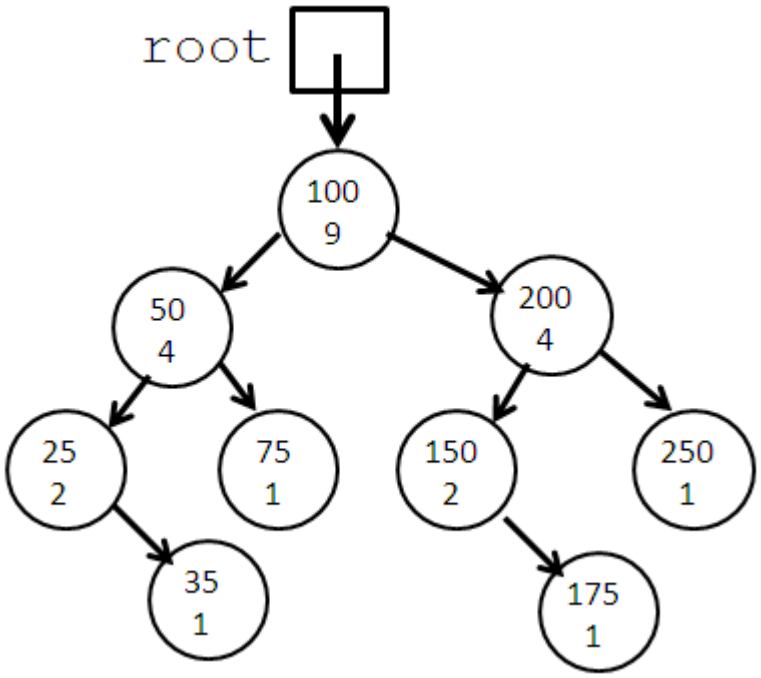
Consider a binary search tree where each node stores the number of descendants it has, including itself. In the picture below the top integer in each node is the actual data stored by the node and the bottom integer in each node indicates the number of descendants a node has, including itself.



When a node is added, the ancestors of that node must have their number of descendant nodes incremented by 1. For example, if the value 35 is added to the tree shown above, the add method creates the following structure.



Notice how the tree with the 35 added has several errors in it. All of the ancestors of the node that contains 35 (25, 50, 100) must have their number of descendants variable incremented by 1. The following picture shows the tree after the ancestors of the node that contains 35 have been fixed.



Part A (10 points): Write a private method in the `AugmentedBinarySearchTree` class that updates the number of descendants in each of the ancestor nodes of a value that has just be added to the tree. **You may not use any other classes in this question other than the `BSTNode` class. You method shall be O(D) time where D is the depth of the node that has been added to the tree.**

The `AugmentedBST` class uses the following node class:

```
public class BSTNode<E> {

     // create a new node with the given values
     public BSTNode(E value, BSTNode<E> left,
                    BSTNode<E> right, int numDescendants)

     public E getValue()
     public BSTNode<E> getLeft()
     public BSTNode<E> getRight()
     public int getNumDescendants()

     public void setValue(E newValue)
     public void setLeft(BSTNode<E> newLeft)
     public void setRight(BSTNode<E> newRight)
     public void setNumDescendants(int newNum)
}
```

Here is the `AugmentedBST` class:

```
public class AugmentedBST <E extends Comparable<? super E>> {

     private BSTNode<E> root;
     private int size;

     public boolean add(E obj) {
          int oldSize = size;
          root = addHelp(obj, root);
          assert size == oldSize || size == oldSize + 1;
          if(oldSize != size)
               fixNumDescendants(obj); // method to complete
          return oldSize != size;
     }
```

Assume if the value `obj` is not already present in the tree then the `addHelp` method correctly inserts a new leaf node that contains `obj` and has its number of descendants variable set to 1 and increments `size` by 1. If `obj` is already present the tree is unchanged.

Complete the `fixNumDescendants` method to correctly update all of the ancestor nodes of the node that contains the newly inserted value. This is an instance method in the `AugmentedBinarySearchTree` class. **You may not use any other classes in this question other than the `BSTNode` class. You method shall be O(D) time where D is the depth of the node that has been added to the tree.**

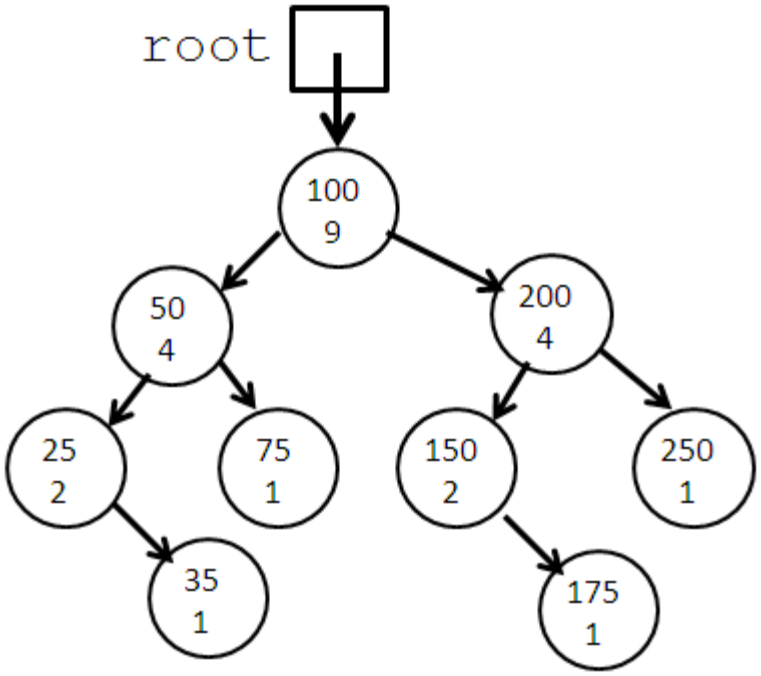
```
// This method is only called by the add method after obj has been
// inserted into the tree.
private void fixNumDescendants(E obj) {
```

```
// more room on next page if needed
```

```
// more room for fixNumDescendants method
```

Question 3, part B, 15 points: One operation that is slow, O(N) , for traditional binary search trees is *getting the kth largest item*. If k is equal to 1, the smallest value in the Binary Search Tree is returned. If k is equal to 2, the second smallest value in the Binary Search Tree is returned, and so forth.

For example given the following augmented binary search tree:



`getKth(1)` returns 25
`getKth(2)` returns 35
`getKth(3)` returns 50
`getKth(4)` returns 75
`getKth(5)` returns 100
`getKth(6)` returns 150
`getKth(9)` returns 250

In a traditional binary search tree the getKth method is O(N) where N is the number of nodes in the tree. In an augmented binary search tree with nodes that track the number of descendants they have (including themselves) the getKth method can be completed in O(D) time, where D is the depth of the node that contains the kth largest value. On average this will be O(logN), a substantial improvement over O(N).

On the next page complete the instance method `getKthHelper` for the `AugmentedBST` class. **You may not use any other classes in this question other than the `BSTNode` class. You method shall be O(D) time where D is the depth of the node that contains the kth largest value.**

```
public class AugmentedBST<E extends Comparable<? super E>> {

     private BSTNode<E> root;
     private int size;

     // pre: 0 < k <= size()
     public E getKth(int k) {
          assert 0 < k && k <= size;
          return getKthHelper(root, k);
     }

// complete the following instance method

private E getKthHelper(BSTNode<E> n, int k) {
```

4. (Using Data Structures. 20 points) Complete a method that implements the radix sort algorithm for an array of positive integers. The radix sort is a multi pass algorithm. The number of passes is equal to the number of digits of the largest value being sorted. For example if the largest integer in an array is 492 (3 digits) the algorithm makes 3 passes through the array to sort it.

The radix sort uses an auxiliary list of queues.

During each pass of the algorithm two steps are completed.

Step 1: Inspect a single digit from each value being sorted. Which digit is inspected depends on which pass of the algorithm is in progress. The first pass of the algorithms inspects the least significant digit of each value, the second pass inspects the second least significant digit, and so forth until the last pass which inspects the most significant digit.

During each pass values are placed into the queue corresponding to the resulting digit from that value. For example the first pass inspects the digit in the ones place. The value 36**5** is enqueued to the queue at position 5 of the list. (Actually the 6th queue due to 0 based indexing.) The value 21**0** is enqueued to the queue at position 0 of the list and so forth.

If a value has no digit at the current place value then the digit is considered to be a 0. For example, the third pass of the radix sort inspects the digit at the hundreds place, the third least significant digit. The value 95 has no explicit digit in the hundreds place, so its hundreds place digit is 0. During the third pass of the radix sort the value 95 is enqueued to the queue at position 0 in the list of queues.

Step 2: After all integers have been inserted into the appropriate queues, each queue, starting with the one at position 0 in the list, is emptied back into the array starting at index 0 in the array and going to the end.

Example: (data is the array of `int` values)

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
data	316	212	12	52	717	128	256

Pass 1, Step 1: examine digit in the ones place

List of Queues. Front of queue is the top

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
		212				316	717	128	
		12				256			
		52							

Pass 1, Step 2. Copy from list of queues, back into the array of ints

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
data	212	12	52	316	256	717	128

Pass 2, Step 1: examine digit in the tens place

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
	212	128			52				
	12				256				
	316								
	717								

Pass 2, Step 2. Copy from list of queues, back into the array of ints

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
data	212	12	316	717	128	52	256

Pass 3, Step 1: examine digit in the hundreds place

[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]
12	128	212	316				717		
52		256							

Pass 3, Step 2. Copy from list of queues, back into the array of ints

	[0]	[1]	[2]	[3]	[4]	[5]	[6]
data	12	52	128	212	256	316	717

The elements in data are now sorted in ascending order.

Complete a method that implements the radix sort algorithm.

```
// pre: data != null, data.length > 0, all elements in data >= 0
public void radixSort(int[] data) {
```

You may use the following method:

```
/*   pre: number >= 0, k > 0
     post: returns the kth digit of number,
     if number has fewer than k digits, 0 is returned
*/
public int getDigit(int number, int k)
```

Part of the algorithm requires you to find the max integer in the values being sorted and then determine how many digits that integer has. Recall a quick and dirty way to determine the number of digits a positive int has is to convert it to a String and take the length. (Do this only to the max value, not all the values, to avoid creating unnecessary String objects.)

```
int numDigits = (maxValue + "").length();
```

Use the following Queue class

```
public class Queue<E>
     // create an empty queue
     public Queue()

     public void enqueue(E obj)
     public E front()
     public E dequeue()
     public boolean isEmpty()
}
```

You will also use the ArrayList class.

ArrayList() **Create an empty ArrayList**	
boolean	**add**(E o) Appends the specified element to the end of this list.
E	**get**(int index) Returns the element at the specified position in this list.

You may not use any other methods or classes other than the ones mentioned in this question.

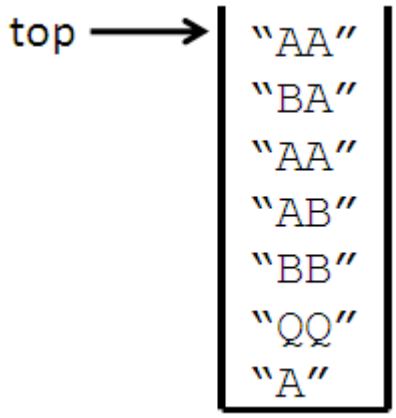
Implement the `radixSort` method on the next page.

```
// pre: data != null, data.length > 0, all elements of data > 0
// post: elements are sorted in ascending order (via the radix
//        sort algorithm)
public void radixSort(int[] data) {
```

5. (Stacks, 10 points) Complete a method that determines how many times a given String is present in a Stack of Strings. This method is **not** an instance method of the Stack class.

```
// pre: st != null, target != null
// post: return the number of times target is present in st.
//        st is not altered as a result of this method.
public int frequency(Stack<String> st, String target) {
```

For example, if `st` refers to the stack shown below:



and `target` refers to the String "AA" the method would return 2. If target refers to the String "Q" the method returns 0.

Use the following Stack class:

```
public class Stack<E> {

     // create an empty Stack
     public Stack()

     public void push(E obj)
     public E top()
     public E pop()
     public boolean isEmpty()
}
```

You may not use any other classes besides the Stack class and the equals method from the String class. You may not use native arrays. The Stack sent as a parameter may not be altered as a result of the method. (It may change during the method, but must be restored to its original state by the time the method completes.)

Implement the method on the next page.

```
// pre: st != null, target != null
// post: return the number of times target is present in st.
//        st is not altered as a result of this method.
public int frequency(Stack<String> st, String target) {
```

Name:_____

Answer sheet for question 1, short answer questions. **Put answers next to the letter.**

_____

A:

_____

B. _____

C._____

D._____

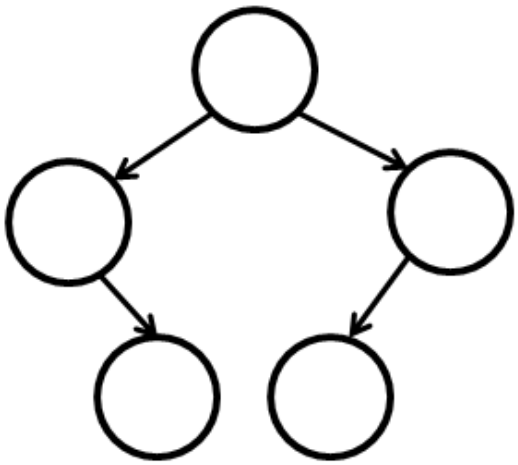
E._____

F. _____

G._____

H._____

I._____



J._____

K._____

L._____

M._____

N._____

O._____

P._____

Q._____

R._____

S._____

T._____