Session 1: You have 30 Minutes

1. The public method, removeLast, removes the last element in a list and returns true if the element is removed and false, otherwise.
   a. (15 pts.) Implement this method in the class SLL which implements the List interface using a singly linked list. State the running time of your method. You are not allowed to use any methods in List interface.

```
public boolean removeLast () {
      Node<E> curr = head;
      Node<E> prev = head;
      if (curr == null) return false;
      if (curr.next == null) {
            head = null;
            return true;
      }

      while (curr.next != null){
            prev = curr;
            curr = curr.next;
      }
      prev.next = null;
      return true;
}                                               13 pts.
T(n) = θ(n)                                     2 pts.
```

   b. (6 pts.) Implement this method in the class BC which includes a list named as myList (an instance of LinkedList class in Java). State the running time of your method. You should access myList through methods in the List interface.

```
public boolean removeLast () {
      if (myList == null || myList.size() == 0)
            return false;
      myList.remove(myList.size()-1);
      return true;
}                                               4 pts.
T(n) = θ(1)                                     2 pts.
```

2. (10 pts.) Assume an implementation of Stack interface using an instance of ArrayList class (as a class variable) in Java where the top of the stack is the first element. Write the code for two basic methods in Stack interface, i.e., push and pop. State the running time of your methods.

```
public void push(E data) {
      theList.add(0,data);
}                                               3 pts.
T(n) = θ(n)                                     2 pts.

public E pop() {
        if (theList.size() == 0) return null;
        return theList.remove(0);
}                                                           3 pts.
T(n) = θ(n)                                                 2 pts.
```

Session 2: You have 30 Minutes

3. (10 pts.) In the implementations of Queue interface using basic array structure in Java, we usually use the array as a circular array. Explain how this usage improves the performance of the implementation. (5 pts) Compare the performance of two implementations with a regular (non-circular) array and a circular array. (5pts.)

   (5 pts.) A regular array structure, can be used in several ways;

   - The head is always at position 0. Adding a new element using offer operation is always constant time (except when ta reallocation is needed). However, removing an element using poll operation takes linear time since shifting all the elements in the queue is required.
   - The head shifts one position to the right after each poll operation. Adding a new element requires shifting all the elements to the left or reallocation although the array is not full (several empty slots at the front of the array), since the tail is at the end of the array. So, the operation is not constant time.

   (5 pts.) On the other hand, when a circular array is used, both tail and head indices are wrapped to start from the index 0 when they are reached to the end of the array. So, empty slots at the beginning of the array is used without any shift operation and reallocation is not needed unless the array is really full.

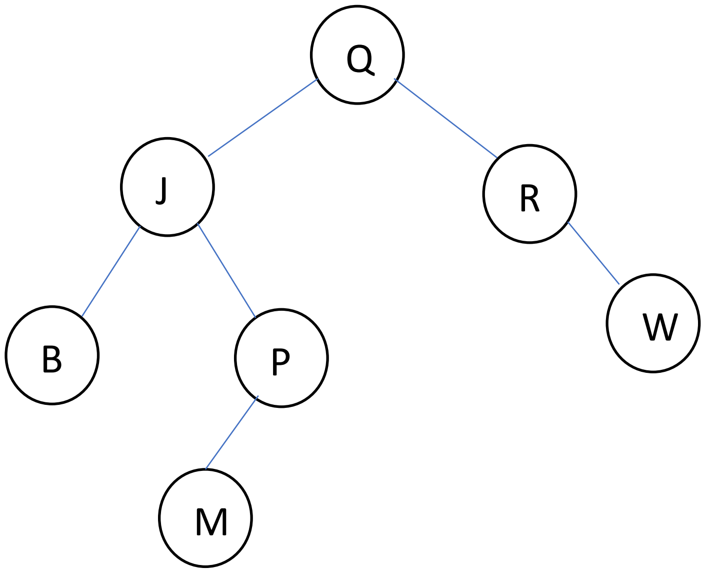
4. (20 pts.) We call a binary tree as balanced if the number of elements in the left subtree and the number of elements in the right subtree are equal for each node in the tree. Add a new public method isBalanced in class BinaryTree (defined in the book) to check whether the binary tree is balanced. You may use a helper method. You are not allowed to define any extra class variable. Analyze the running time of your method. The running time of your method should be O(n).

```
public boolean isBalanced () {
      if (isBalanced(root)> 0) return true;
      return false;
}
public int isBalanced(Node<E> root) {
      if (root == null) return 0;
      int numLeft = isBalanced (root.left);
      int numRight = isBalanced (root.right);
      if (numLeft>0 && numRight>0)
            if(numLeft == numRight)
                  return numLeft+numRight+1;
            else
                  return -1;
      return -1
}                                                 15 pts.
T(n) = θ(n)                                       5 pts.
```

Since each node is processed just once (in preorder) and takes constant time.

Session 3: You have 40 Minutes

5. Binary Search Tree.
    a. (4 pts.) Insert the letters Q, J, B, R, W, P, M (in the given order) into an initially empty binary search tree. Draw the resulting tree.



    b. (4 pts.) Insert the first three letters in your surname into the resulting binary search tree. Draw the resulting tree.
    c. (2 pts.) From the resulting tree, delete 'R'. Draw the resulting tree.
    d. (4 pts.) From the resulting tree, delete 'Q'. Draw the resulting tree. Write down the nodes traversed during the operation.
6. (25 pts.) Add a new public method numSmaller in the KWPriorityQueue class (defined in the book, implementing min-heap in an array). The method returns the number of nodes smaller than the item passed as parameter. Implement this method recursively. Note that you should not traverse the whole tree (or the array representing the tree). You may use a helper method. You are not allowed to define any extra class variable.

```
public int numSmaller (Comparable<E> item) {
      if (size()==0) return 0;
      return numSmaller (0, item);
}
public int numSmaller(int parent, Comparable<E> item){
      if (item.compareTo(table[parent])<= 0 ) return 0;
      int leftNum=0, rightNum=0;
      if (leftChild<size())
            leftNum = numSmaller(2*parent+1, item);
      if (rightChild<size())
            rightNum = numSmaller(2*parent+1, item);
      return (1+leftNum+rightNum);
}                                                 25 pts.
```