Exam 2 Summer 2021 Solutions

Q1. Size of List

```
//Node Class
public class Node {
    public int data;
    public Node next;
}

/**
* This method will return the number of nodes in a given linked list
* @param head A pointer of the beginning of the linked list.
* Note: DO NOT create a helper function
*/
public int sizeOfLinkedList(Node head) {
    if(head == null) {
        return 0;
    } else {
        return 1 + sizeOfLinkedList(head.next);
    }
}
```

Q2. Print a tree

```
//Tree Class
public class Tree {
  public class TreeNode {
    public TreeNode left;
    public int data;
    public TreeNode right;
  }

  private TreeNode root;

  public TreeNode getRoot() {
    return this.root;
  }

  /**
      5
     / \
    3   7
   / \ / \
  1  4 6  8

  * This method will return a string of all the data in the tree in post order.
  * For example, if t was the tree above, it would return “1436875”
  */
  public String postOrderToString() {
    return postOrderToStringHelper(root);
  }
```

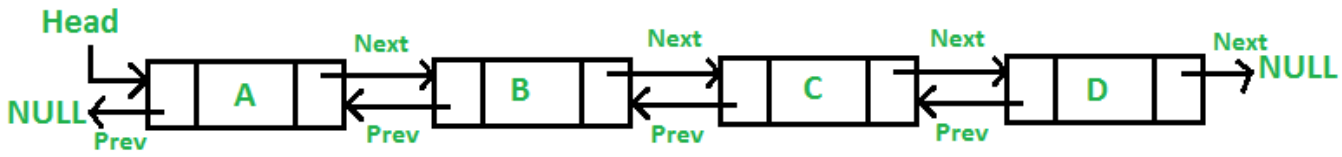
```
  private String postOrderToStringHelper(TreeNode n) {
    if(n == null) {
      return “”;
    }
    return postOrderToStringHelper(n.left) + postOrderToStringHelper(n.right) + n.data;
  }
}
```

Q3. Heap Sort

“Heapsort” is a sorting algorithm that involves inserting all the elements into a heap, and then removing them one at a time (where they will come in sorted order, from largest to the smallest).

```
public class MinHeap{
  //assume the following methods are implemented for you.
  public boolean isEmpty(){..}
  public int remove(){...}
  public void insert(int x){...}

  //fill in the blanks in heapsort
  public ArrayList<Integer> heapSort(ArrayList<Integer> in){
    ArrayList<Integer> arr = new ArrayList<Integer>();
    for(int elem : in){
        insert(elem);
    }
    while(!isEmpty()){
        arr.add(remove());
    }
    return arr;
  }
}
```

Q4. Delete Doubly Linked List

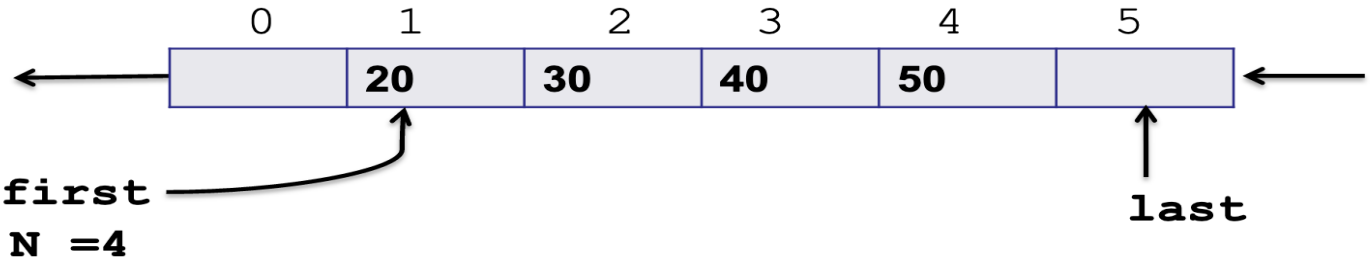



```
public class Node<E implements Comparable<E>> {
    E data;
    Node<E> next;
    Node<E> prev;
}

public boolean deleteFromDoublyLinkedList(E key) {
    Node<E> current = head; //head is the first node in the list
    //STEP 1: Find the value to be deleted
    while(current != null) {
        if(current.data.compareTo(key) != 0) { //Type E is comparable
```

```
            current = current.next;
        } else {
            break;
        }
    }

    if(current == null) return false;  //does not exist
    else {
        //STEP2: Remove the node with the data and rechain the linked list
        if(current.prev == null) { //Head of the list is being removed
           head = current.next;
           if(current.next != null) {
              current.next.prev = null;
           }
        } else { //Not the head of the list
            current.prev.next = current.next;
            if(current.next != null) { //Not last node in the list
                current.next.prev = current.prev;
            }
        }
        return true;
    }
}
```

Q5. Queue

Below is the code for enqueuing an item into a circular array-based queue. `N` is the number of items currently on the queue, `q` is the array representing the queue, `last` is the index of the next free position of the queue and `resize()` doubles the size of the queue and copies the contents over into the new array, maintaining queue order and setting `first` and `last` appropriately. You should only resize when the circular array is full. Here is the queue after we enqueue 10, 20, 30, 40, 50 and dequeue one item:




```
public void enqueue(E item) {
    //Check if resize is required
    if (first == last) {
      resize();
    }
    //Add the item to the queue
    q[last++] = item;
    //Wrap around
    if (last >= q.length) {
      last = 0;
    }
    N++;
}
```

Q6.

Find the two problems with the code provided below that can lead to incorrect behaviour or runtime errors (No compilation errors).

```
public void intArrayToString(int[] arr) {
    intArrayToStringHelper(arr, 0);
}

private void intArrayToStringHelper(int[] arr, int index) {
    //Print the value at index in the array
    System.out.println(arr[index]);
    //Make the recursive call to print the rest of the values in the array
    intArrrayToString(arr);
}
```

Q6.1.

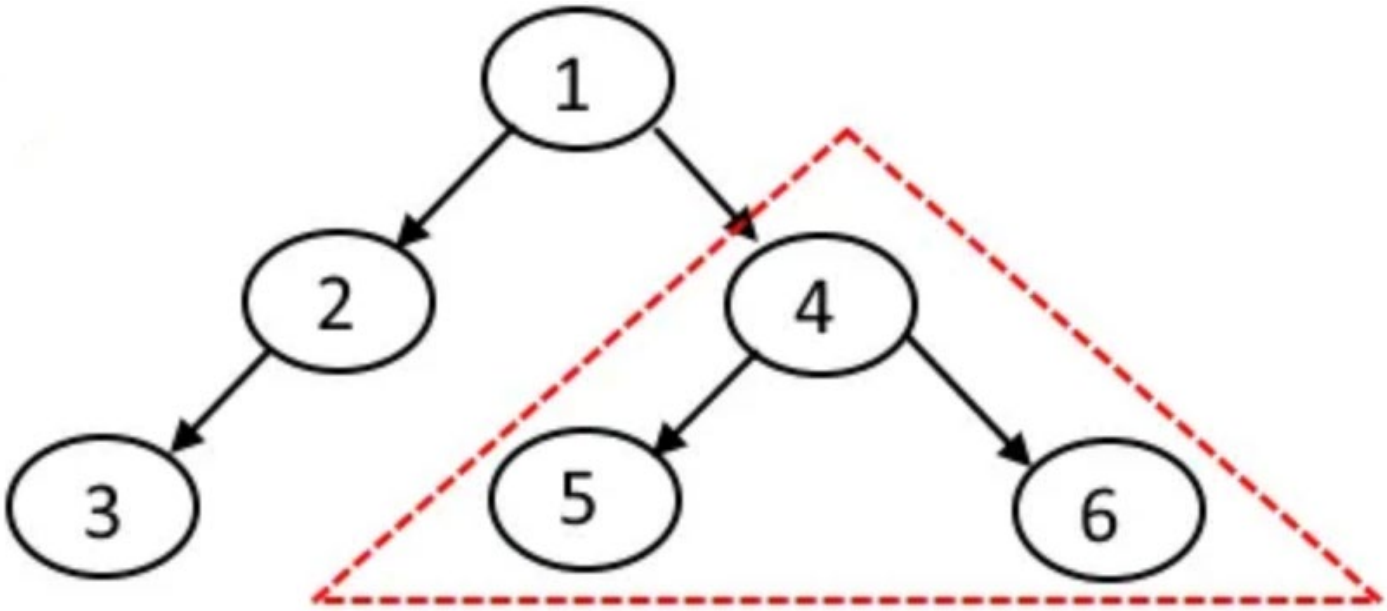
Problem: Index is not updated, infinite recursion.
Fix: Change recursive call `intArrrayToString(arr)` to `intArrrayToStringHelper(arr, index + 1)`.

Q6.2.

Problem: No base case, index out of bounds.
Fix: Add base case: `if(index == arr.length) return;`

Q7.

Here is a function that deletes a subtree whose root data is equal to the given key. For example, `deleteSubtree(4)` will delete the subtree in the red triangle.



Assume keys are unique.

```
public class Node {
        public Node left;
        public int data;
        public Node right;
```

```
}

public void deleteSubtree(int key){
    root = deleteSubtree(root, key);
}
private Node deleteSubtree(Node r, int key){
    if (r ==  null)
        return null;
    if (r.data == key){
        r = null;
    }
    deleteSubtree(r.left, key);
    deleteSubtree(r.right, key);
}
```

Q7.1.

Problem: Left/right subtrees not updated, no return statements.
Fix: Change last two lines of code to:

```
r.left = deleteSubtree(r.left, key);
r.right = deleteSubtree(r.right, key);
return r;
```

Q7.2.

Problem: Setting `r = null` when key is found does not delete the subtree.
Fix: Change `r = null` to `return null`.

Q8. Min Height

Complete the function `minHeight()` below that calculates the minimum height of a binary tree. The minimum height of a binary tree is the number of edges on the shortest path from the root to a leaf. You are allowed to add a helper method. Note: A tree with one node (the root) has a height of 0.

```
public class Tree {
    private class TreeNode {
        TreeNode left, right;
        int data;
    }
    private TreeNode root;

    public int minHeight() {
        return Math.max(0, minHeightHelper(root));
    }

    private int minHeight(Node n) {
        if (n == null) {
            return -1; // Only if root = null
        }
        if (n.left == null && n.right == null) {
            return 0;
        }
        if (n.left == null) {
            return 1 + minHeight(n.right);
```

```
        }
        if (n.right == null) {
            return 1 + minHeight(n.left);
        }
        return 1 + Math.min(minHeight(n.left), minHeight(n.right));
    }
}
```

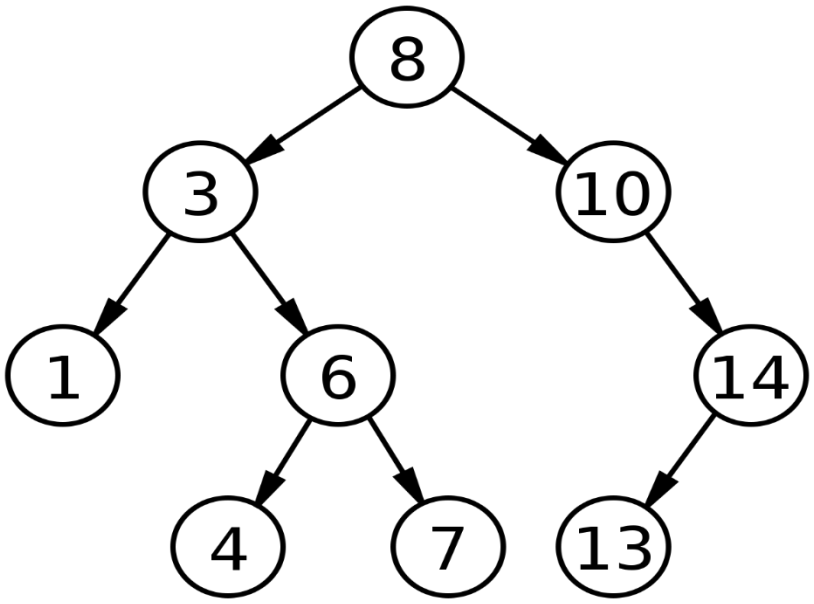
Q9. Singly Linked List

Given the head hA of a single linked list A and the head hB of the single linked lists B, write a function Node append(Node h1, Node h2) that appends list B to list A and returns a new list.

```
public class Node {
    Node next;
    int data
    public Node(int data){
      this.data = data;
    }
}

public Node append(Node hA, Node hB){
    if (hA == null) {
        return hB;
    }
    Node tA = hA;
    while (tA.next != null) {
        tA = tA.next;
    }
    tA.next = hB;
    return hA;
}
```

Q10. Binary Tree Traversal

Write the preorder, inorder, and postorder traversal of the binary tree shown below:



Preorder: 8, 3, 1, 6, 4, 7, 10, 14, 13

Inorder: 1, 3, 4, 6, 7, 8, 10, 13, 14

Postorder: 1, 4, 7, 6, 3, 13, 14, 10, 8

Q11. Recursion

```
int mystery(int a) {
  if (a == 0) {
    return 0;
  }else {
    return mystery(a-1) + 2*a-1;
  }
}
```

What is the output of `mystery(3)`? 9