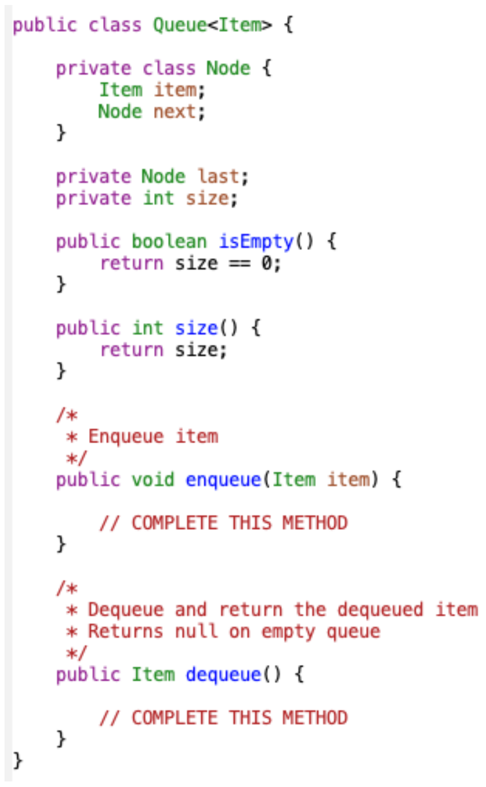
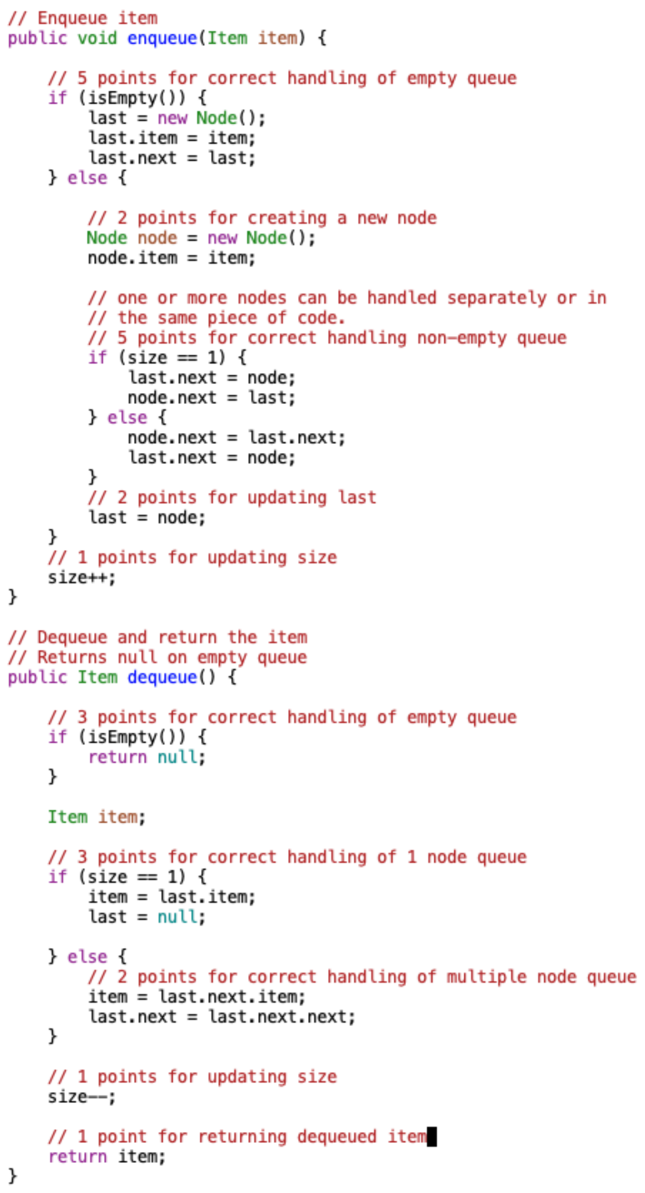
Problem 1: Queue and circular linked list (25 points)

a) Implement a Queue, where enqueue and dequeue operations take constant time, using a circular linked list with only one access pointer to the last element of the list.



```java
public class Queue<Item> {

    private class Node {
        Item item;
        Node next;
    }

    private Node last;
    private int size;

    public boolean isEmpty() {
        return size == 0;
    }

    public int size() {
        return size;
    }

    /*
     * Enqueue item
     */
    public void enqueue(Item item) {

        // COMPLETE THIS METHOD
    }

    /*
     * Dequeue and return the dequeued item
     * Returns null on empty queue
     */
    public Item dequeue() {

        // COMPLETE THIS METHOD
    }
}
```

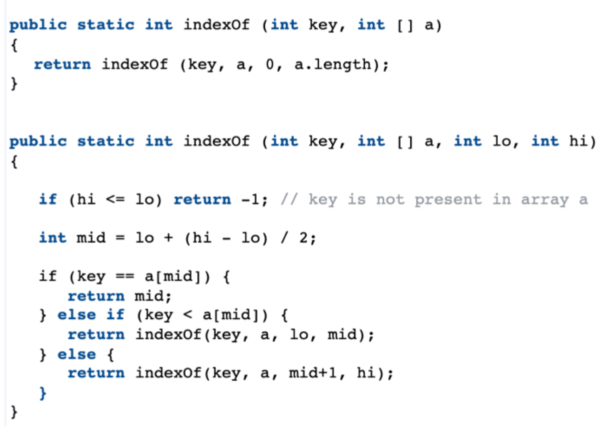


```java
// Enqueue item
public void enqueue(Item item) {

    // 5 points for correct handling of empty queue
    if (isEmpty()) {
        last = new Node();
        last.item = item;
        last.next = last;
    } else {

        // 2 points for creating a new node
        Node node = new Node();
        node.item = item;

        // one or more nodes can be handled separately or in
        // the same piece of code.
        // 5 points for correct handling non-empty queue
        if (size == 1) {
            last.next = node;
            node.next = last;
        } else {
            node.next = last.next;
            last.next = node;
        }
        // 2 points for updating last
        last = node;
    }
    // 1 points for updating size
    size++;
}

// Dequeue and return the item
// Returns null on empty queue
public Item dequeue() {

    // 3 points for correct handling of empty queue
    if (isEmpty()) {
        return null;
    }

    Item item;

    // 3 points for correct handling of 1 node queue
    if (size == 1) {
        item = last.item;
        last = null;

    } else {
        // 2 points for correct handling of multiple node queue
        item = last.next.item;
        last.next = last.next.next;
    }

    // 1 points for updating size
    size--;

    // 1 point for returning dequeued item
    return item;
}
```

Problem 2: Binary Search (25 points)

The binary search implementation `indexOf` code below searches for a key on `a[lo, hi)`. The notation `a[lo,hi)` means `a[lo], a[lo+1] ... a[hi-1]` (does not include `a[hi]`).

i) **(10 points)** Draw the decision tree for binary search, using `indexOf`, for an array of 7 elements. Include failure nodes, and mark comparisons on the nodes and branches.



```
public static int indexOf (int key, int [] a)
{
  return indexOf (key, a, 0, a.length);
}


public static int indexOf (int key, int [] a, int lo, int hi)
{

   if (hi <= lo) return -1; // key is not present in array a

   int mid = lo + (hi - lo) / 2;

   if (key == a[mid]) {
      return mid;
   } else if (key < a[mid]) {
      return indexOf(key, a, lo, mid);
   } else {
      return indexOf(key, a, mid+1, hi);
   }
}
```

// 5 points for correct index on tree
// 3 points for failure nodes
// 2 points for comparison marks

```
              3=

        </             \>

     1=                    5=

  </     \>         </      \>

  0 =      2=         4=       6=

</   \>  </   \>  </   \>  </   \>

 F    F   F    F   F    F   F    F
```

ii) **(5 points)** What is the maximum number of comparisons for a successful search? 5

iii) **(5 points)** What is the minimum number of comparisons for a successful search? 1

iv) **(5 points)** What is the maximum number of comparisons for an unsuccessful search? 6

Problem 3: Binary Search Tree (40 points)

a) **(8 points)** You are given the following numbers to insert into an empty BST:

5, 7, 8, 12, 15, 27

Select which insertion order would yield the tree with the least height?

A) 15, 5, 27, 8, 7, 12
B) 12, 7, 15, 27, 5, 8
C) 8, 27, 7, 5, 15, 12
D) 7, 5, 12, 8, 15, 27

Sol: B
This tree of B has a height of 2. All other options yield BSTs with height 3.

b) (12 points) A perfect balanced BST requires all interior nodes to contain two children and all leaf nodes to be on the same level. A perfect balanced BST of *h*=0 would have *n*=1 node, *h*=1 would have *n*=3 nodes, *h*=2 would have *n*=7 nodes, and so on. Where *h* is *tree height* and *n* is the *number of nodes in the tree*.

What is the worst-case Big O, in terms of *n*, for successfully searching for an element in a perfect balanced BST? Give the reasoning process for full credit.

Sol:

O(log n). If there is no reasoning, 0 pts.

Reasoning: Suppose the tree height is h, then 2^0 + 2^1 + 2^2 + … + 2^h = n, (5 pts)
then 2^(h+1) – 1 = n, then h = log2(n+1)-1, (2pts)
The worst case count of comparisons is h+1 which is log2(n+1)-1+1=log2(n+1). (5 pts)

c) (20 points) Implement a method `countInRange` to count, in a BST, all keys that are within a given range. Assume that keys in the BST are unique, i.e. there are no duplicates. You MUST use recursion, and you may not use any helper methods.

For example, suppose that the keys in a BST are 4, 7, 10, 14, 15, 17, 20, 30.
`countInRange(root, 8, 17)` returns 4 because keys 10, 14, 15, and 17 are greater than or equal to 8 and smaller than or equal to 17.

`countInRange(root, 21, 29)` returns 0 because there are no keys greater than or equal to 21 and smaller than or equal to 29.

```
public class BSTNode {
   int key;
   Value value;
```

```
      BSTNode left;
      BSTNode right;
      ...
   }
   // counts number of keys in BST that are in the range low to high,
   inclusive
   // i.e. all keys k such that low <= k <= high
   public static int countInRange(BSTNode root, int low, int high) {

       // COMPLETE THIS METHOD

   }
```

Sol:

```
public static int countInRange(BSTNode root, int low, int high){

   if (root == null) {  // 2 pts
      return 0;         // 1 pt
   }

   int count = 0;  // 1 pt

   if (low <= root.key && root.key <= high) {  // 4 pts
      count++;
   }
   if (low < root.key) {
      count += countInRange(root.left, low, high); // 5 pts
   }
   if (root.key < high) {
      count += countInRange(root.right, low, high); // 5 pts
   }
   return count;  // 2 pts
}
```

Problem 4: (40 points) 2-3 trees and red-black trees

a) **(15 points)** Draw the resulting 2-3 tree when you insert the keys N O I T U Q Y S A E in that order to an initially empty tree. You need to show the tree:
  (i) after 3 keys N O I are inserted, and
  (ii) after 7 keys N O I T U Q Y are inserted, and
  (iii) the final tree after all keys N O I T U Q Y S A E are inserted.

The tree after first 2 keys inserted is shown as the following 3-node.

N O

Your answers must be shown as a 2-3 tree with only 2-nodes and 3-nodes.

Answers

(i) 5-pts (all good), 3-pts if they wrote answer as I O N (3-node), 0 pts (otherwise)

```
   N
I     O
```

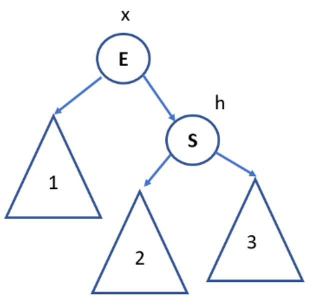
(ii) 5-pts (all good), 2-pts if they left any 3-nodes, 0 pts (otherwise)

```
    N T
I       O Q   U Y
```

(iii) 5-pts (all good), 3-pts if they left any 3-nodes, 0 pts (otherwise)

```
               Q
        E N             T
      A   I   O      S     U Y
```

b) **(9 points)** Suppose that you are given the following Binary Search Tree (BST). The keys E and S are in nodes with references x and h respectively.

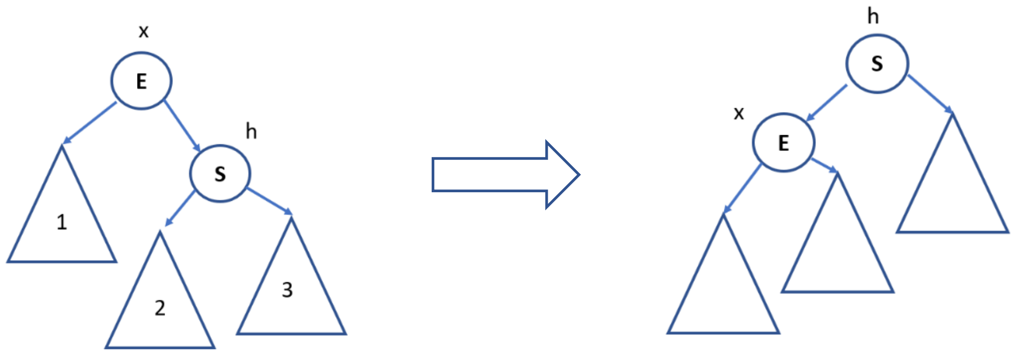


(i) **(3 points)** Write the range of keys that are possible in subtrees marked as 1, 2 and 3. The answers should be given in terms of keys E and S

Answer: (1 pts each) (1) < E (2) between E and S (3) more than

(ii) **(6 points)** Write at least 3 lines of code that transforms the left BST into the right BST shown below. You can assume that each node (Node class below) has left and right references (i.e. x.right refers to right subtree of x).

```
public class Node {
   int key;
   Value value;
   Node left;
   Node right;
   ...
}
```



Answer: Node tmp = h.left;

h.left = x;

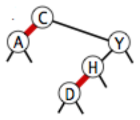
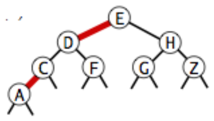
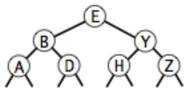
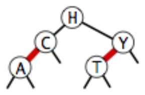
x.right = tmp;

2-points for each right line of code.

If there are more instructions that is ok. If the answers are given using keys E and S, that is E.left etc (deduct 0.5 pts from each answer)

c) **(16 points)** Define the height of a node as the maximum number of links from that node to any leaf node. The height of a tree is then the height of the root node. Define a **balanced** tree as a tree where the height difference between left and right subtrees for any node is at most 1. An **ordered tree** is one where an in-order traversal of the tree produces a sorted sequence. Label the following red-black trees as balanced and/or ordered.

2 points for each correct answer

Tree	Ordered? (yes/no)	Balanced? (yes/no)
	YES	NO
	NO	YES
	YES	YES
	YES	YES