Task 3

Algorithm

- Initialize a Queue object with a maximum size.
- Read in a line of input and parse it into an array of integers.
- For each integer in the array:
  - Insert the integer into the queue.
  - If the bottom element of the queue (peek) is equal to the size of the queue, and the queue is not empty:
  - Remove the bottom element of the queue and print it.
  - If the queue is not empty, print the remaining elements of the queue.

Explanation

On the first day, the disk of size 4 is given. But you cannot put the disk on the bottom of the tower as a disk of size 5 is still remaining.

On the second day, the disk of size 5 will be given so now disk of sizes 5 and 4 can be placed on the tower.

On the third and fourth day, disks cannot be placed on the tower as the disk of 3 needs to be given yet. Therefore, these lines are empty.

On the fifth day, all the disks of sizes 3, 2, and 1 can be placed on the top of the tower.

```java
import java.util.Scanner;

public class Queue{
    private int max;
    private int[] queue;
    private int noOfItems;

    public Queue(int size){
        this.max = size;    //  N denoting the total number of disks that are
given to you in the N subsequent days
        queue = new int[max]; //initializing the array to the maximum i-th in
Day (disk)
        noOfItems  =0; // number of items in Queue
    }
    // to insert element to based on the Priority and sort it
    public void insert(int value){
        int i ;
        // if no of items is 0 then the value will be inserted at 0 index
        if(noOfItems == 0){
            queue[0] = value;
            noOfItems++;
            return;
        }
        // else no of items is > 0 then value is compared and sorted
        for(i = noOfItems-1; i >= 0 ; i--){
            if(value < queue[i]){
                queue[i+1] = queue[i];
            }else{
                break;
            }
        }
        queue[i+1] = value; // insertion of the value after sorting
        noOfItems++;
    }
    // mostly used
    public int remove(){
        max--;
        return queue[--noOfItems];
    }
    // not used
    public boolean isfull(){
        return noOfItems == max;
    }
    // used for checking if there is items in the array
    public boolean isEmpty(){
        return noOfItems == 0;
    }
    public int peek(){
```

```
        return queue[noOfItems - 1];
    }
    public int size(){
        return max;
    }
    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        Queue tower = new Queue(Integer.parseInt(in.nextLine())); // max
        String stringDisk[] = in.nextLine().split(" ");// split the values and
save it in array

        for(int i = 0; i < stringDisk.length; i ++){
            tower.insert(Integer.parseInt(stringDisk[i])); // insert and sort
the values
            while (!tower.isEmpty() && tower.peek() == tower.size()) { // if
both are true poll or remove the values from the array

                System.out.print(tower.remove() + " ");
            }
            System.out.println();
        }
    }
}
```

Task 4

Algorithm:

- Create a new BinarySearchTree object called b.
- Initialize a Scanner object called in to read input from user.
- Read the first line of input and split it into two strings, N and C.
- Read the second line of input and assign it to the temp variable. Set the elements of temp to N after it has been split by spaces.
- Read the third line of input and assign it to the temp variable. Assign the elements of temp to C after it has been split by spaces.
- For each element in N and C, insert it into the BinarySearchTree with the following logic:
  - if the index of the element is 0, insert the element at the root with a value of 0.
  - Otherwise, insert the element at its corresponding index in N with the corresponding value in C.
- For each vertex in the BinarySearchTree, call the printAncestorsWithSameColor function with the root of the tree and the vertex as arguments.

```java
import java.util.Scanner;
public class BinarySearchTree{

    public static Node root;
    public int maxColor;
    public int maxVertex;



    public BinarySearchTree(){
        this.root = null;
    }

    // finding the data in the tree
    public Node find(int vertex){
        Node current = root;

        while(current !=null){
            if(current.vertex == vertex){
                return current;
            }else if(current.vertex > vertex ){
                current = current.left;
            }else{
                current = current.right;
            }
        }
        return null;
    }
    public int getColor(int vertex){
       return find(vertex).color;
    }

    public void insert(int id , int color ){
        maxVertex++;
        Node newNode = new Node(id , color ,maxVertex);
        if(root == null){
            root = newNode;

            return;
        }
        Node current = root;
        Node parent = null;
        while(true){
            parent = current;
            if(maxVertex< current.vertex){
                current = current.left;
                if(current == null){
```

```
                    parent.left = newNode;

                    return;
                }
            }else{
                current = current.right;
                if(current == null){
                    parent.right = newNode;

                    return;
                }
            }
        }
    }

    void printAncestorsWithSameColor(Node node,  Node target) {
        // Base case: node is null or target is not found in the tree
        if (node == null) {
            return;
        }

        // If target is found, print the current node and return

         if (node.color == target.color && node.vertex == target.vertex) {
            System.out.print(-1 + " ");
            return;
        }else if (node.color == target.color) {
            System.out.print(node.vertex+ " ");
            return;
        }
        // Recursive case: search for the target in the left and right
subtrees
        printAncestorsWithSameColor(node.left, target);
        printAncestorsWithSameColor(node.right, target);
    }



    class Node{
        int data , color , vertex;
        Node left;
        Node right;
        public Node (int data ,int color , int vertex){
            this.data = data;
            this.color = color;
            this.vertex =vertex;
            left = null;
            right = null;
```

```
        }

    }
    public static void main(String[] args) {
        BinarySearchTree b = new BinarySearchTree();
        Scanner in = new Scanner(System.in);
        // assigning values to N and C
        String temp = in.nextLine();
        String[] N,C;
        N = new String [Integer.parseInt(temp.substring(0,1))] ;
        C = new String[Integer.parseInt(temp.substring(2,temp.length()))];
        // ------------------------------------------------------------------
---------
        // Asigning values to N[i]
        temp = in.nextLine();
        N[0] = "0";
        N = temp.split(" ");
        //Asigning values to C[i]
        temp = in.nextLine();
        C = temp.split(" ");
      //  N[0] = temp.split( " "); // root
        System.out.println(Arrays.toString(N));
        System.out.println(Arrays.toString(C));
        for(int i = 0 ; i <= N.length; i ++){
            if(i == 0){
                b.insert(0, Integer.parseInt(C[i])); // for the root
            }else{
                b.insert(Integer.parseInt(N[i-1]), Integer.parseInt(C[i]));
            }
        }

       for(int i = 1; i <=b.maxVertex; i++){
        b.printAncestorsWithSameColor(b.root, b.find(i));
       }
    }
}
```