Practice Questions

Solution 1:-

```
class Solution
{
   //Function to find the vertical order traversal of Binary Tree.
   static ArrayList <Integer> verticalOrder(Node root)
   {
      // add your code here
      Queue<Pair> q=new ArrayDeque<>();
      TreeMap<Integer,ArrayList<Integer>> hm=new TreeMap<>();
      ArrayList<Integer> ans=new ArrayList<>();

      q.add(new Pair(root,0));

      while(!q.isEmpty())
      {
         Node first=q.peek().first;
         int second=q.peek().second;
         q.poll();

         if(hm.containsKey(second))
         {
           hm.get(second).add(first.data);
         }else{
            ArrayList<Integer> a=new ArrayList<>();
            a.add(first.data);
            hm.put(second,a);
         }
```

```
            if(first.left!=null)
            {
                q.add(new Pair(first.left,second-1));
            }
            if(first.right!=null)
            {
                q.add(new Pair(first.right,second+1));
            }
        }

        for(Map.Entry<Integer,ArrayList<Integer>> e: hm.entrySet()){
            ArrayList<Integer> a=e.getValue();
            ans.addAll(a);
        }

        return ans;
    }
}
```

Solution 2:-

```
class Solution
{
    //Function to find the length of a loop in the linked list.
     public static Node startNodeCycle(Node head)
    {
        Node fastPtr = head;
        Node slowPtr = head;
        while(fastPtr != null && fastPtr.next != null)
```

```
    {
        fastPtr = fastPtr.next.next;
        slowPtr = slowPtr.next;
        if(slowPtr==fastPtr)
            return getStartNode(slowPtr,head);
    }
    return null;
}
public static Node getStartNode(Node slowPtr,Node head)
{
    Node temp =head;
    while(temp!=slowPtr)
    {
        slowPtr=slowPtr.next;
        temp=temp.next;
    }
    return temp;
}
static int countNodesinLoop(Node head)
{
    //Add your code here.
    Node slowPtr = startNodeCycle(head);
    if(slowPtr==null) return 0;
    Node temp = slowPtr.next;
    int count =0;
    while(temp!=slowPtr)
    {
```

```
            count ++;
            temp=temp.next;
        }
        return count+1;
    }
}
```

Solution 3:-

```
class Solution
{
public static long[] productExceptSelf(int nums[], int n)
{
        // code here
        long res[]=new long[nums.length];



        for(int i=0;i<nums.length;i++)
        {
            long product=1;
            check(nums,res,product,i);
        }

        return res;
}
public static void check(int nums[],long res[],long product,int index)
{
    for(int i=0;i<nums.length;i++)
    {
        if(i==index)
```

```
        {
            continue;
        }
        else
        {
            product=(long) product*nums[i];
        }
    }
    res[index]=product;
}
}
```

Solution 4:-

```
class Solution{
    //Function to partition the array around the range such
    //that array is divided into three parts.
    public void threeWayPartition(int array[], int a, int b)
    {
        // code here
        int smaller=0;
        int between=0;
        int larger=array.length-1;
        while(between<=larger){
            if(array[between]<a){
                swap(array,smaller,between);
                smaller++;
                between++;
            }
            else if(array[between]>=a && array[between]<=b) between++;
            else{
                swap(array,larger,between);
```

```
            larger--;
          }
        }
    }

    public void swap(int array[], int a, int b){
        int temp=array[a];
        array[a] = array[b];
        array[b] = temp;
    }
}
```