Q1-> Minimum Numbers of Jumps

```
class Solution{
    static int minJumps(int[] arr) {
        int n = arr.length;

        if (n <= 1) {
            return 0; // No jumps needed for an empty array or single element
        }

        if (arr[0] == 0) {
            return -1; // Cannot make any jumps if the first element is zero
        }

        int jumps = 1; // Minimum number of jumps
        int maxReach = arr[0]; // Maximum index that can be reached overall
        int steps = arr[0]; // Remaining steps at the current position

        for (int i = 1; i < n; i++) {
            if (i == n - 1) {
                return jumps; // Reached the last element, return the number of jumps
            }

            steps--;

            maxReach = Math.max(maxReach, i + arr[i]);

            if (steps == 0) {
                jumps++;

                if (i >= maxReach) {
                    return -1; // Cannot progress further
                }
                steps = maxReach - i;
            }
        }

        return -1; // Cannot reach the last element
    }
}
```

Q2-> SubArray with given sum

```
class Solution
{
    //Function to find a continuous sub-array which adds up to a given number.
    static ArrayList<Integer> subarraySum(int[] arr, int n, int s)
    {
        // Your code here
        ArrayList<Integer> ans = new ArrayList<Integer>();
        int start = 0;
        int end = 0;
        int sum = 0;
        if(s==0){
            ans.add(-1);
            return ans;
        }

        while (end < n) {
            sum += arr[end];

            while (sum > s) {
                sum -= arr[start];
                start++;
            }
            if (sum == s) {
                ans.add(start + 1);
                ans.add(end + 1);
                return ans;
            }

            end++;
        }

        ans.add(-1);
        return ans;
    }
}
```

Q3-> Remove Loop in Linked List

```
class Solution
{
    public static void removeLoop(Node head) {
        if (head == null || head.next == null) {
            return;
        }

        Node slow = head;
        Node fast = head;

        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;

            if (slow == fast) {
                break;
            }
        }

        if (slow != fast) {
            return;
        }

        int loopLength = 1;
        fast = fast.next;
        while (fast != slow) {
            fast = fast.next;
            loopLength++;
        }

        fast = head;
        for (int i = 0; i < loopLength; i++) {
            fast = fast.next;
        }

        slow = head;
        while (slow != fast) {
            slow = slow.next;
            fast = fast.next;
        }

        while (fast.next != slow) {
            fast = fast.next;
        }

        fast.next = null;
    }
}
```

Q4-> Nth node form end of the Linked List

```
class Solution
{
  //Function to find the data of nth node from the end of a linked list.
  int getNthFromLast(Node head, int n)
  {
        // Your code here
        int count = 0;
        Node temp=head;
        while(temp!= null){
          temp=temp.next;
          count++;
        }
        if(n>count){
          return -1;
        }
        count=count-n+1;
        temp= head;
        for(int i=0;i<count;i++){
          if(i!=count-1){
            temp = temp.next;
          }
          else{
            break;
          }
        }
        return temp.data;
  }
}
```

Q5-> Implement stack using linked list

```
class MyStack
```

```
{
    // class StackNode {
    //     int data;
    //     StackNode next;
    //     StackNode(int a) {
    //         data = a;
    //         next = null;
    //     }
    // }
    StackNode top;

    //Function to push an integer into the stack.
    StackNode head = null;
    void push(int a)
    {
        StackNode newNode = new StackNode(a);
        if(head==null){
            head = newNode;
            return;
        }
        else{
            newNode.next = head;
            head = newNode;
        }
        // Add your code here

    }

    //Function to remove an item from top of the stack.
    int pop()
    {
        // Add your code here
        if(head==null){
            return -1;
        }
        else{
            
            int n = head.data;
            head = head.next;
            return n;
            
        }
    }
}
```

Q6-> Get minimum element from the stack

```
class GfG
{
    int minEle;
    Stack<Integer> s;

    // Constructor
    GfG()
             {
        s = new Stack<Integer>();
        
             }

    /*returns min element from stack*/
    int getMin()
    {
             // Your code here
                 minEle = Integer.MAX_VALUE;
                 if(s.empty()){
```

```
                return -1;
            }
            for(int i=0;i<s.size();i++){
                minEle = Math.min(minEle, s.get(i));
            }
            return minEle;
    }

    /*returns poped element from stack*/
    int pop()
    {
        // Your code here
            if(s.empty()){
                return -1;
            }
            int n = s.peek();
            s.pop();
            return n;


    }

    /*push element x into the stack*/
    void push(int x)
    {
        // Your code here


            s.push(x);
    }
}
```