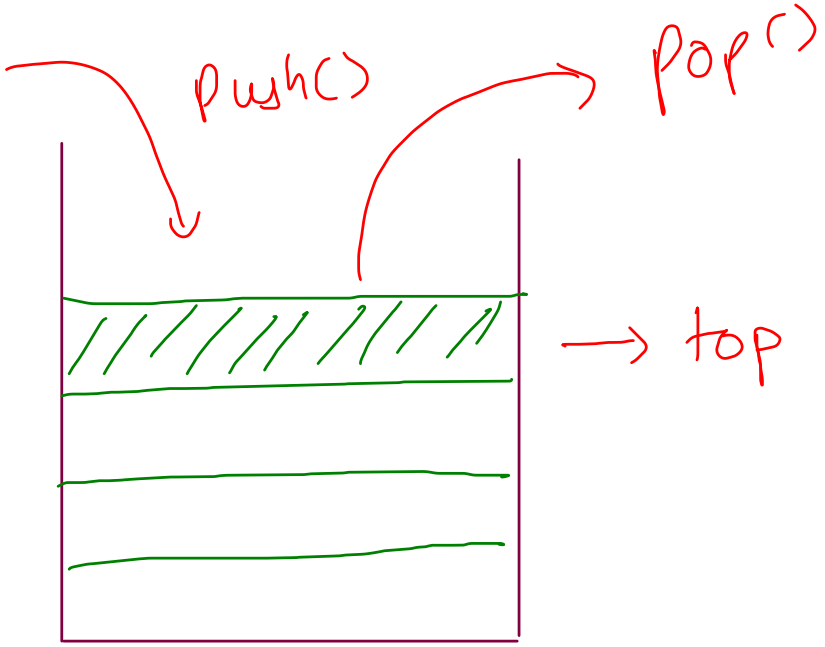
Stack

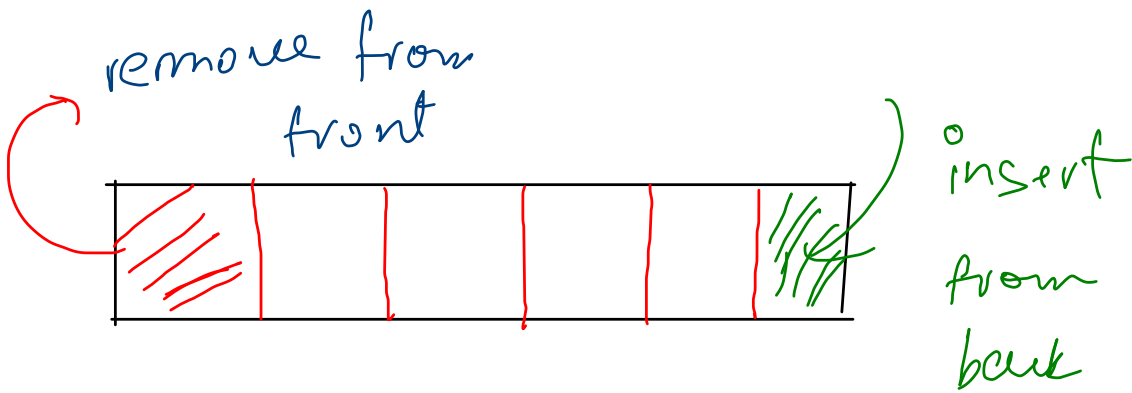
{ Lifo or filo }



pop and push both from the top

Queue

{ fifo or LILO }



Queue is an interface hence it cannot be instantiated

Hence, we use ArrayDeque for creating the object while the reference is of Queue only-

→ Stack is by default implemented using arrayList.

Normal Stack (Pepcoding)

```java
public static class CustomStack {
  int[] data;
  int top;

  public CustomStack(int cap) {
    data = new int[cap];
    top = -1;
  }

  int size() {
    // write ur code here
    return top + 1;
  }

  void display() {
    // write ur code here
    for(int i=top;i>=0;i--) {
        System.out.print(data[i] + " ");
    }
    System.out.println();
  }

  void push(int val) {
    // write ur code here
    if(top == data.length -1) {
        System.out.println("Stack overflow");
        return;
    }

    top++;
    data[top] = val;
  }
```

```java
  int pop() {
    // write ur code here
    if(top == -1) {
        System.out.println("Stack underflow");
        return -1;
    }

    int x = data[top];
    top--;
    return x;
  }

  int top() {
    // write ur code here
    if(top == -1) {
        System.out.println("Stack underflow");
        return -1;
    }

    return data[top];
  }
}
```

Dynamic Stack

```
// change the code of this function according
void push(int val) {
  if (tos == data.length - 1) {
    int n = data.length;
    int[] arr = new int[n * 2];
    for(int i=0;i<n;i++) {
        arr[i] = data[i];
    }

    data = new int[2*n];

    for(int i=0;i<n;i++) {
        data[i] = arr[i];
    }

    tos = n;
    data[tos] = val;
  } else {
    tos++;
    data[tos] = val;
  }
}
```

→ doubling

Here amortized time complexity of push is O(1)

↓ avg

Σ Push Time

Σ Push

↓

no of times push is performed

Stack using LL

```java
public static class LLToStackAdapter {
  LinkedList<Integer> list;

  public LLToStackAdapter() {
    list = new LinkedList<>();
  }


  int size() {
    // write your code here
    return list.size();
  }

  void push(int val) {
    // write your code here
    list.addFirst(val);
  }

  int pop() {
    // write your code here
    if(list.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }

    return list.removeFirst();
  }

  int top() {
    // write your code here
    if(list.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }

    return list.getFirst();
  }
}
```

push → addfirst
pop → remove first
} O(1)

Queue using LL

```
public static class LLToQueueAdapter {
  LinkedList<Integer> list;

  public LLToQueueAdapter() {
    list = new LinkedList<>();
  }

  int size() {
    // write your code here
    return list.size();
  }

  void add(int val) {
    // write your code here
    list.addLast(val);
  }

  int remove() {
    // write your code here
    if(list.size() == 0) {
        System.out.println("Queue underflow");
        return -1;
    }

    return list.removeFirst();
  }

  int peek() {
    // write your code here
    if(list.size() == 0) {
        System.out.println("Queue underflow");
        return -1;
    }

    return list.getFirst();
  }
}
```

if tail also available

addLast → queue add

removeFirst → remove

↓

removeLast will be O(N) always as we need prev pointer

Two Stacks in an Array -

```
class Stacks
{
    //Function to push an integer into the stack1.
    void push1(int x, TwoStack sq)
    {
        if(sq.top1 == sq.top2 - 1) return;
        sq.top1++;
        sq.arr[sq.top1] = x;
    }

    //Function to push an integer into the stack2.
    void push2(int x, TwoStack sq)
    {
        if(sq.top1 == sq.top2 - 1) return;
        sq.top2--;
        sq.arr[sq.top2] = x;
    }

    //Function to remove an element from top of the stack1.
    int pop1(TwoStack sq)
    {
        if(sq.top1 == -1) return -1;
        int x = sq.arr[sq.top1];
        sq.top1--;
        return x;
    }

    //Function to remove an element from top of the stack2.
    int pop2(TwoStack sq)
    {
        if(sq.top2 == sq.arr.length) return -1;
        int x = sq.arr[sq.top2];
        sq.top2++;
        return x;
    }
}
```

Normal Queue

```
public static class CustomQueue {
  int[] data;
  int front;
  int size;

  public CustomQueue(int cap) {
    data = new int[cap];
    front = 0;
    size = 0;
  }

  int size() {
    // write ur code here
    return size;
  }

  void display() {
    // write ur code here
    for(int i=0;i<size;i++)
    {
        int index=(front+i)%data.length;
        System.out.print(data[index] + " ");
    }
    System.out.println();
  }
```

```
  void add(int val) {
    // write ur code here
    if(size==data.length)
    {
        System.out.println("Queue overflow");
        return;
    }

    int rear=(front+size)%data.length;
    data[rear]=val;
    size++;
  }

  int remove() {
    // write ur code here
    if(size==0)
    {
        System.out.println("Queue underflow");
        return -1;
    }
    int val=data[front];
    front=(front+1) % data.length;
    size--;
    return val;
  }

  int peek() {
     // write ur code here
     if(size==0)
     {
         System.out.println("Queue underflow");
         return -1;
     }
     return data[front];
  }
}
```

Leetcode → Implement Circular Queue.

```java
class MyCircularQueue {

    int front;
    int rear;
    int size;
    int []arr;

    public MyCircularQueue(int k) {
        arr = new int[k];
        size = 0;
        front = rear = 0;
    }

    public boolean enQueue(int value) {
        if(size == arr.length) return false;
        arr[rear] = value;
        rear = (rear + 1)%arr.length;
        size++;
        return true;
    }

    public boolean deQueue() {
        if(size == 0) return false;
        front = (front + 1)%arr.length;
        size--;
        return true;
    }

    public int Front() {
        if(size == 0) return -1;
        return arr[front];
    }

    public int Rear() {
        if(size == 0) return -1;
        return arr[(rear - 1 + arr.length) % arr.length];
        //this is because rear vala element rear se ek index peeche hai
        //aur agar index 0 par rear hua to rear-1 karne se problem create ho
        //jaegi. Isliye humne arr.length ko add krva liya
    }

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

    public boolean isFull() {
        if(size == arr.length) return true;
        return false;
    }
}
```

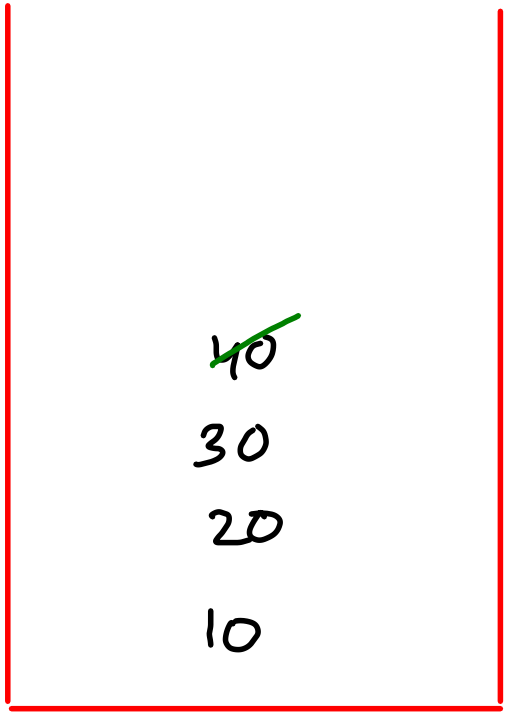
Dynamic Queue

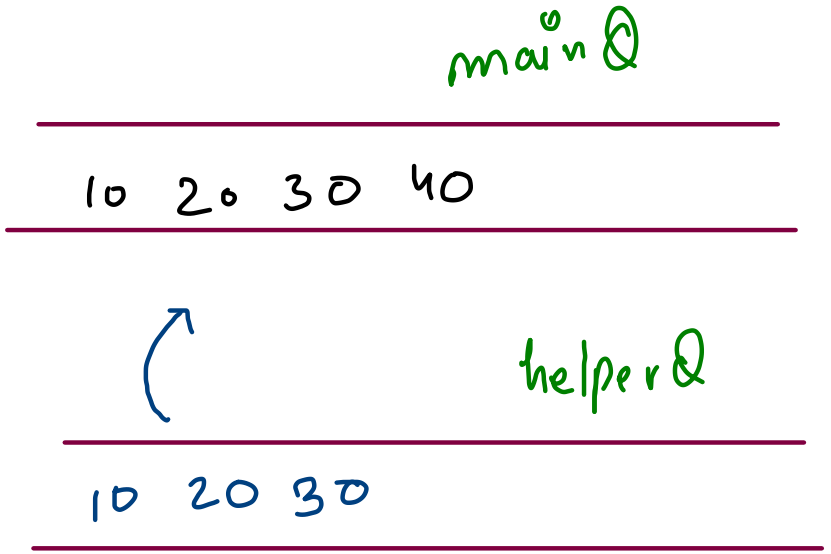
doubling

```java
// change this code
void add(int val) {
  // write ur code here
  if(size == data.length){
    int n = data.length;
    int[] temp = new int[2*n];

    for(int i = 0; i < size; i++){
        int idx = (front + i) % data.length;
        temp[idx] = data[idx];
    }

    data = new int[2*n];
    data = temp;

    int idx = (front + size) % data.length;
    data[idx] = val;
    size++;

  } else {
    int idx = (front + size) % data.length;
    data[idx] = val;
    size++;
  }
}
```

Stack using Queue → Push Efficient { Strictly a Queue Not a Deque }

push → q.add





pop → maing.remove & add helperQ except last ele.
mainQ = helperd.

top → same as pop just add the last ele too.

When we make push efficient, pop and peek() of Stack become O(N) both.

```
public static class QueueToStackAdapter {
  Queue<Integer> mainQ;
  Queue<Integer> helperQ;

  public QueueToStackAdapter() {
    mainQ = new ArrayDeque<>();
    helperQ = new ArrayDeque<>();
  }

  int size() {
    // write your code here
    return mainQ.size();
  }

  void push(int val) {
    // write your code here
    mainQ.add(val);
  }
```

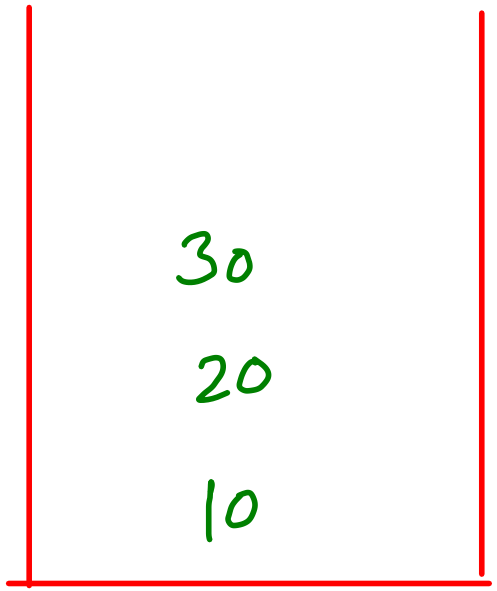
```
int pop() {
  // write your code here

  if(mainQ.size() == 0) {
      System.out.println("Queue underflow");
      return -1;
  }

  int res = -1;
  while(mainQ.size() > 0) {
      int ele = mainQ.remove();
      if(mainQ.size() == 0) {
          res = ele;
          break;
      }
      helperQ.add(ele);
  }

  mainQ = helperQ;
  helperQ = new ArrayDeque<>();
  return res;
}
```

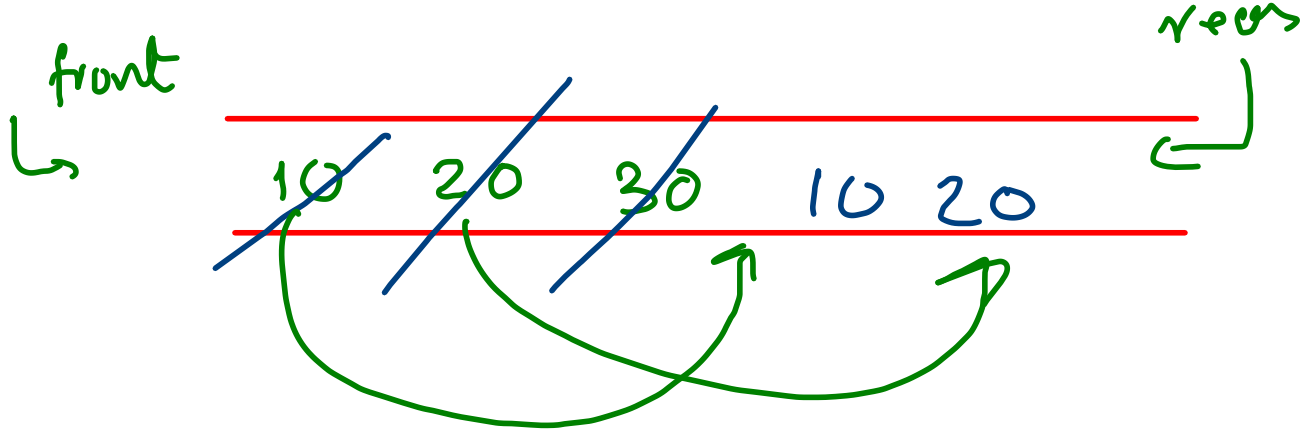
↳ Stack

```
int top() {
  // write your code here
   if(mainQ.size() == 0) {
      System.out.println("Queue underflow");
      return -1;
  }

  int res = -1;
  while(mainQ.size() > 0) {
      int ele = mainQ.remove();
      if(mainQ.size() == 0) {
          res = ele;
      }
      helperQ.add(ele);
  }

  mainQ = helperQ;
  helperQ = new ArrayDeque<>();
  return res;

}
```

↳ Stack

Push Efficient using 1 Queue





push → q.add

remove → remove & add for all ele.

```java
public static class QueueToStackAdapter {
  Queue<Integer> q;

  public QueueToStackAdapter() {
    q = new ArrayDeque<>();
  }

  int size() {
    // write your code here
    return q.size();
  }

  void push(int val) {
    // write your code here
    q.add(val);
  }

  int pop() {
    // write your code here
    if(q.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }
    int s = q.size();
    int res = 0;
    while(s > 0) {
        int ele = q.remove();
        s--;
        if(s == 0) {
          res = ele;
          break;
        }
        q.add(ele);
    }

    return res;
  }
```

```java
  int top() {
    // write your code here
    if(q.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }
    int s = q.size();
    int res = 0;
    while(s > 0) {
        int ele = q.remove();
        s--;
        if(s == 0) res = ele;
        q.add(ele);
    }

    return res;
  }
}
```

Stack using 1 queue

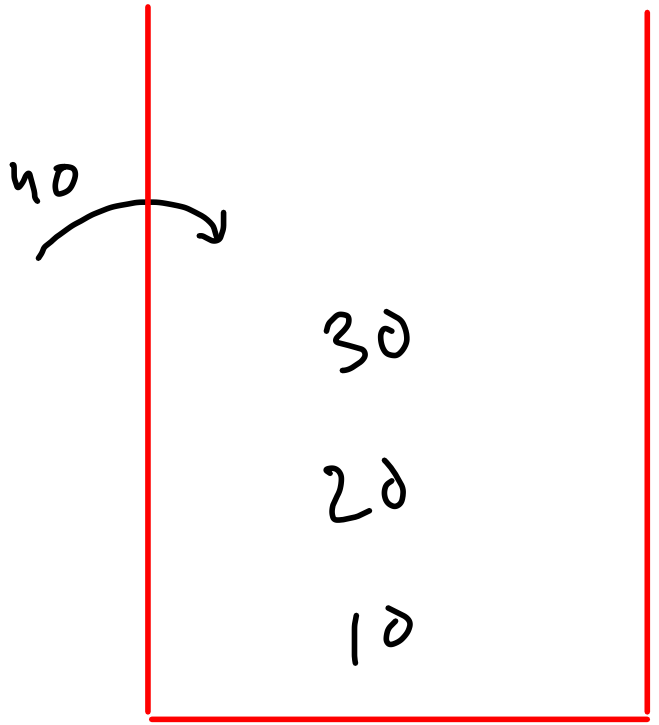
Push Efficient.

Stack using Queue → Pop Efficient.

push → 1 helperQ.add(ele)
2 mainQ → all pop & push in helperQ
3 mainQ = helperQ

mainQ

30	20	10



helperQ

40	30	20	10

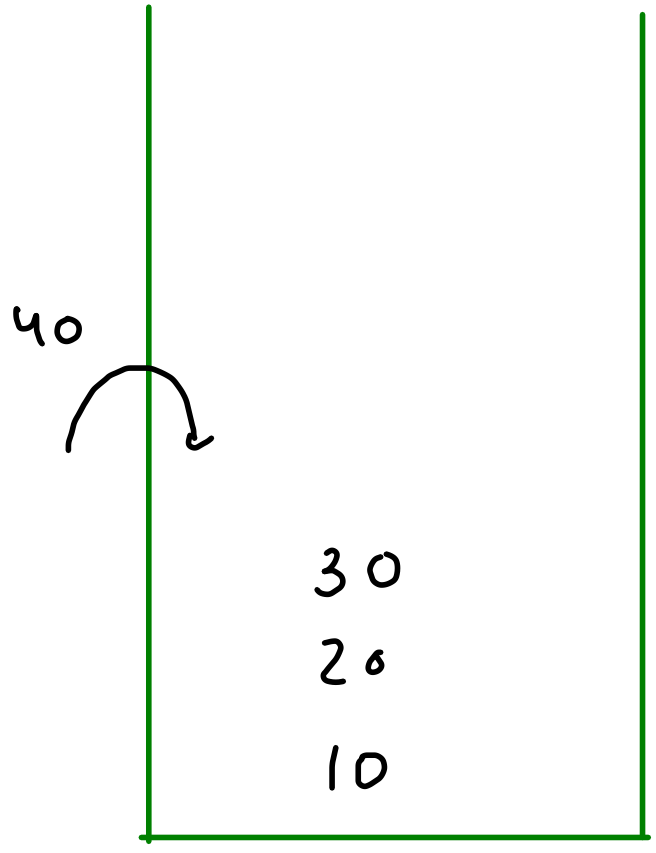
pop → mainQ.remove()

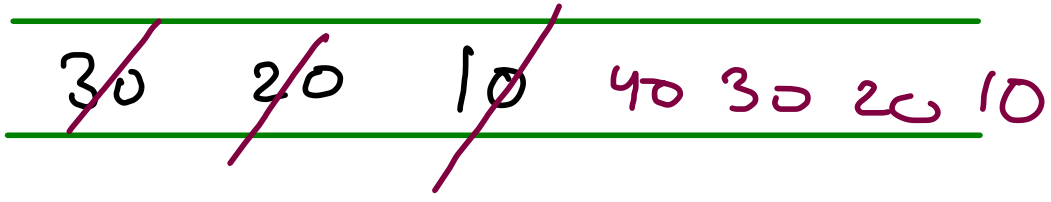
peek → mainQ.peek()

```java
public static class QueueToStackAdapter {
  Queue<Integer> mainQ;
  Queue<Integer> helperQ;

  public QueueToStackAdapter() {
    mainQ = new ArrayDeque<>();
    helperQ = new ArrayDeque<>();
  }

  int size() {
    // write your code here
    return mainQ.size();
  }

  void push(int val) {
    // write your code here
    helperQ.add(val);

    while(mainQ.size() > 0) {
        helperQ.add(mainQ.remove());
    }

    mainQ = helperQ;
    helperQ = new ArrayDeque<>();
  }

  int pop() {
    // write your code here
    if(mainQ.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }

    return mainQ.remove();
  }
```

```java
  int top() {
    // write your code here
    if(mainQ.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }

    return mainQ.peek();
  }
}
```

Stack Using 1 Queue → pop Efficient





When removing the first ele, add curr ele first.

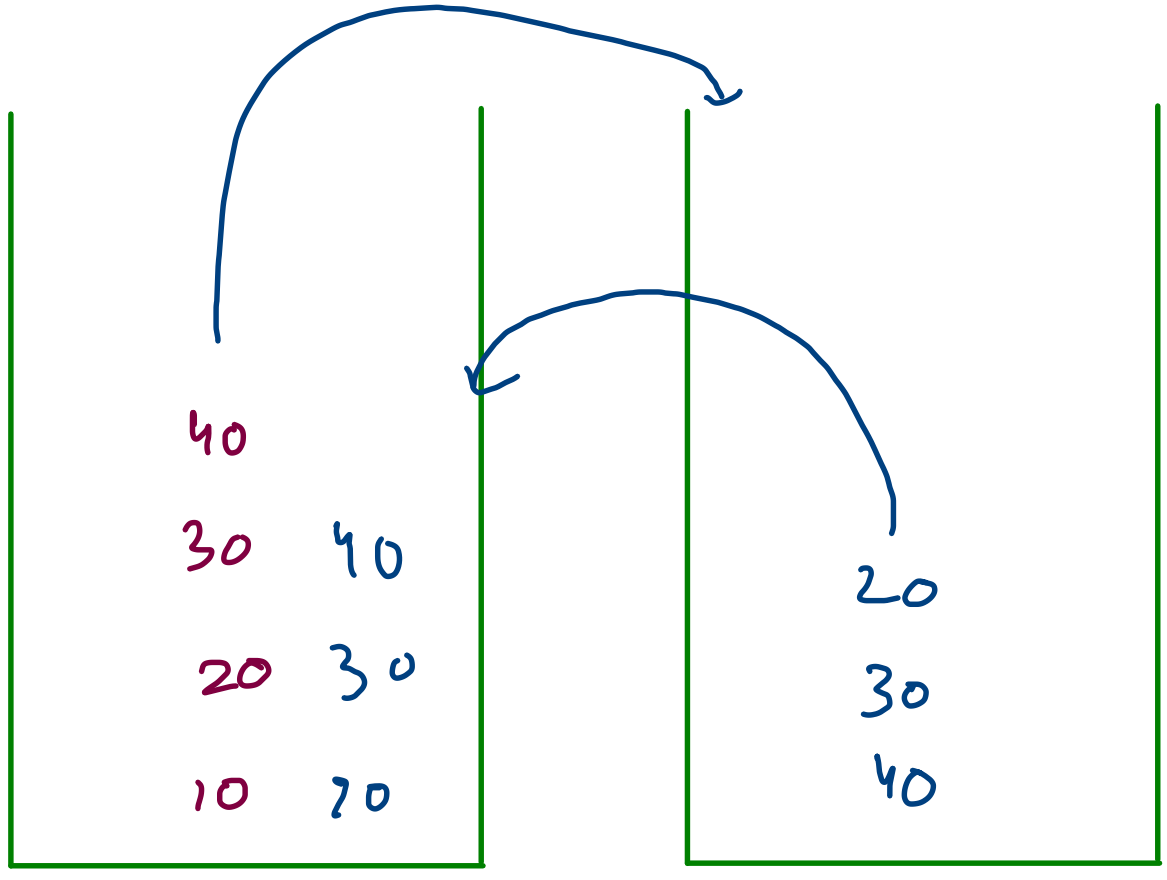
```java
public static class QueueToStackAdapter {
  Queue<Integer> q;

  public QueueToStackAdapter() {
    q = new ArrayDeque<>();
  }

  int size() {
    // write your code here
    return q.size();
  }

  void push(int val) {
    // write your code here
    int s = q.size();

    if(q.size() == 0) {
        q.add(val);
        return;
    }

    for(int i=1;i<=s;i++) {
        int ele = q.remove();
        if(i == 1) {
            q.add(val);
        }
        q.add(ele);
    }
  }
```

```java
  int pop() {
    // write your code here
    if(q.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }

    return q.remove();
  }

  int top() {
    // write your code here
    if(q.size() == 0) {
        System.out.println("Stack underflow");
        return -1;
    }

    return q.peek();
  }
}
```

Stack using 1 queue

Pop efficient

Queue using Stack – Add Efficient



~~10~~ 20 30 40

add → ms.push()

remove → pop ele from ms & add it to hs

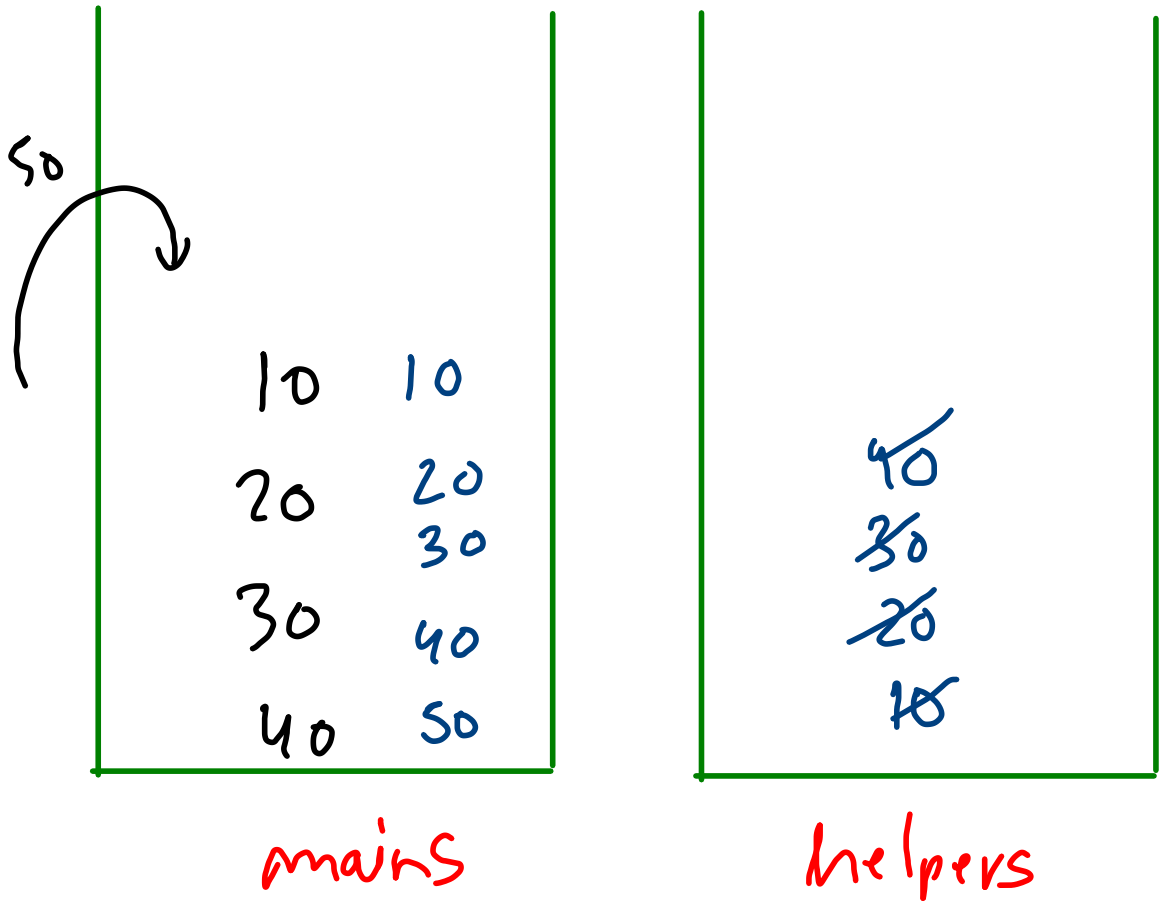
for every ele except last remaining pop eles from hs to ms.

```
public class Main {

  public static class StackToQueueAdapter {
    Stack<Integer> mainS;
    Stack<Integer> helperS;

    public StackToQueueAdapter() {
      mainS = new Stack<>();
      helperS = new Stack<>();
    }

    int size() {
      // write your code here
      return mainS.size();
    }

    void add(int val) {
      // write your code here
      mainS.push(val);
    }
```

```
    int remove() {
      // write your code here
      if(mainS.size() == 0) {
          System.out.println("Queue underflow");
          return -1;
      }

      int res = 0;
      int s = mainS.size();
      while(s > 0) {
          int ele = mainS.pop();
          s--;
          if(s == 0) {
              res = ele;
              break;
          }
          helperS.push(ele);
      }

      while(helperS.size() > 0) {
          mainS.push(helperS.pop());
      }

      return res;
    }
```

```
    int peek() {
      // write your code here
      if(mainS.size() == 0) {
          System.out.println("Queue underflow");
          return -1;
      }

      int res = 0;
      int s = mainS.size();
      while(s > 0) {
          int ele = mainS.pop();
          s--;
          if(s == 0) {
              res = ele;
          }
          helperS.push(ele);
      }

      while(helperS.size() > 0) {
          mainS.push(helperS.pop());
      }

      return res;
    }
  }
```

Queue using Stack → Remove Efficient



10 20 30 40 50

remove → pop

```java
public static class StackToQueueAdapter {
  Stack<Integer> mainS;
  Stack<Integer> helperS;

  public StackToQueueAdapter() {
    mainS = new Stack<>();
    helperS = new Stack<>();
  }

  int size() {
    // write your code here
    return mainS.size();
  }

  void add(int val) {
    // write your code here
    int s = mainS.size();
    while(s-- > 0) {
        int ele = mainS.pop();
        helperS.push(ele);
    }
    mainS.push(val);

    while(helperS.size() > 0) {
        mainS.push(helperS.pop());
    }
  }

  int remove() {
    // write your code here
    if(mainS.size() == 0) {
        System.out.println("Queue underflow");
        return -1;
    }
    return mainS.pop();
  }
```

```java
  int peek() {
    // write your code here
    if(mainS.size() == 0) {
        System.out.println("Queue underflow");
        return -1;
    }
    return mainS.peek();
  }
}
```

Balanced Brackets

```
[(a + b) + {(c + d) * (e / f)}] -> true
[(a + b) + {(c + d) * (e / f)]} -> false
[(a + b) + {(c + d) * (e / f)} -> false
([(a + b) + {(c + d) * (e / f)}] -> false
```

opening bracket → push

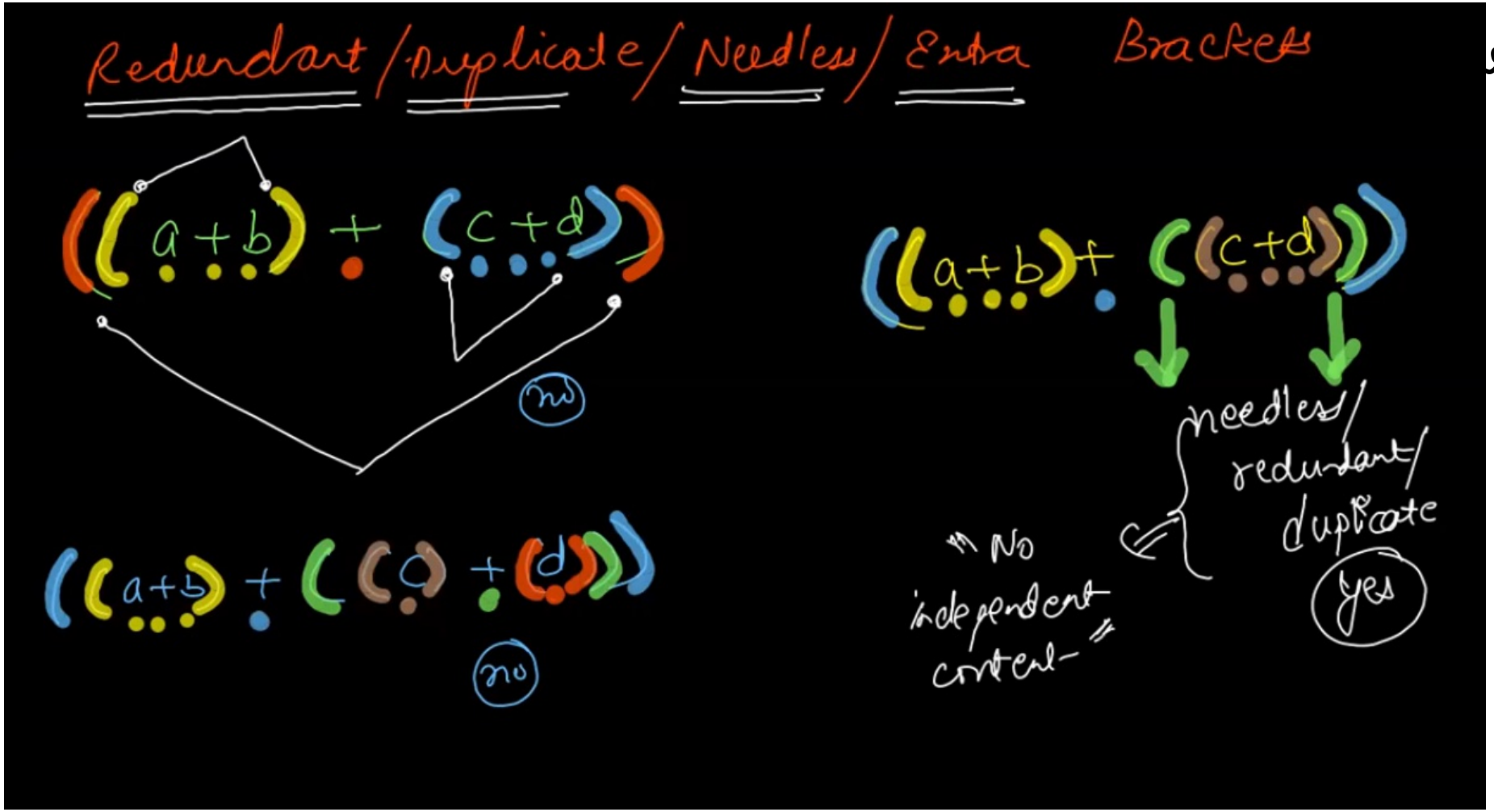
closing → if st.empty → return false

else check if corresponding opening bracket is at the top if yes then pop it, if no → return false.

```java
public static boolean isBalanced(String str) {
    Stack<Character> st = new Stack<>();

    for(int i=0;i<str.length();i++) {
        char ch = str.charAt(i);

        if(ch == '(' || ch == '[' || ch == '{') {
            st.push(ch);
        } else if(ch == ')' || ch == ']' || ch == '}') {
            if(st.size() == 0) return false;

            if(ch == ')') {
                char top = st.pop();
                if(top != '(') return false;
            } else if(ch == ']') {
                char top = st.pop();
                if(top != '[') return false;
            } else {
                char top = st.pop();
                if(top != '{') return false;
            }
        }
    }

    if(st.size() > 0) return false;
    return true;
}
```

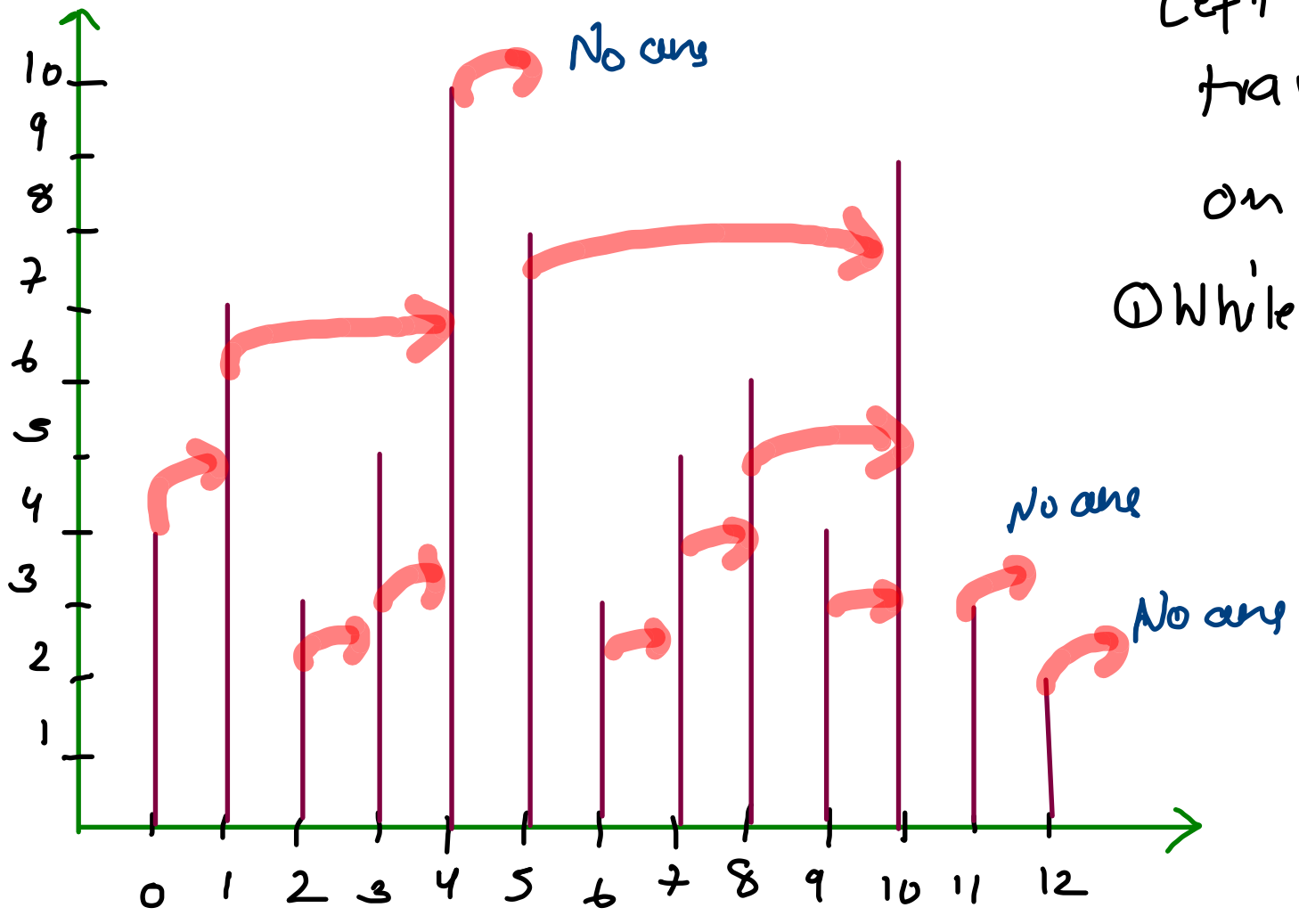
Duplicate brackets / Redundant / Needless

{ Brackets will be balanced for



```java
public static boolean solve(String str) {
    Stack<Character> st = new Stack<>();
    for(int i=0;i<str.length();i++) {
        char ch = str.charAt(i);
        if(ch == ')') {
            if(st.peek() == '(') {
                return true;
            }

            while(st.peek() != '(') {
                st.pop();
            }

            st.pop();
        } else st.push(ch);
    }

    return false;
}
```

NGE to the Right

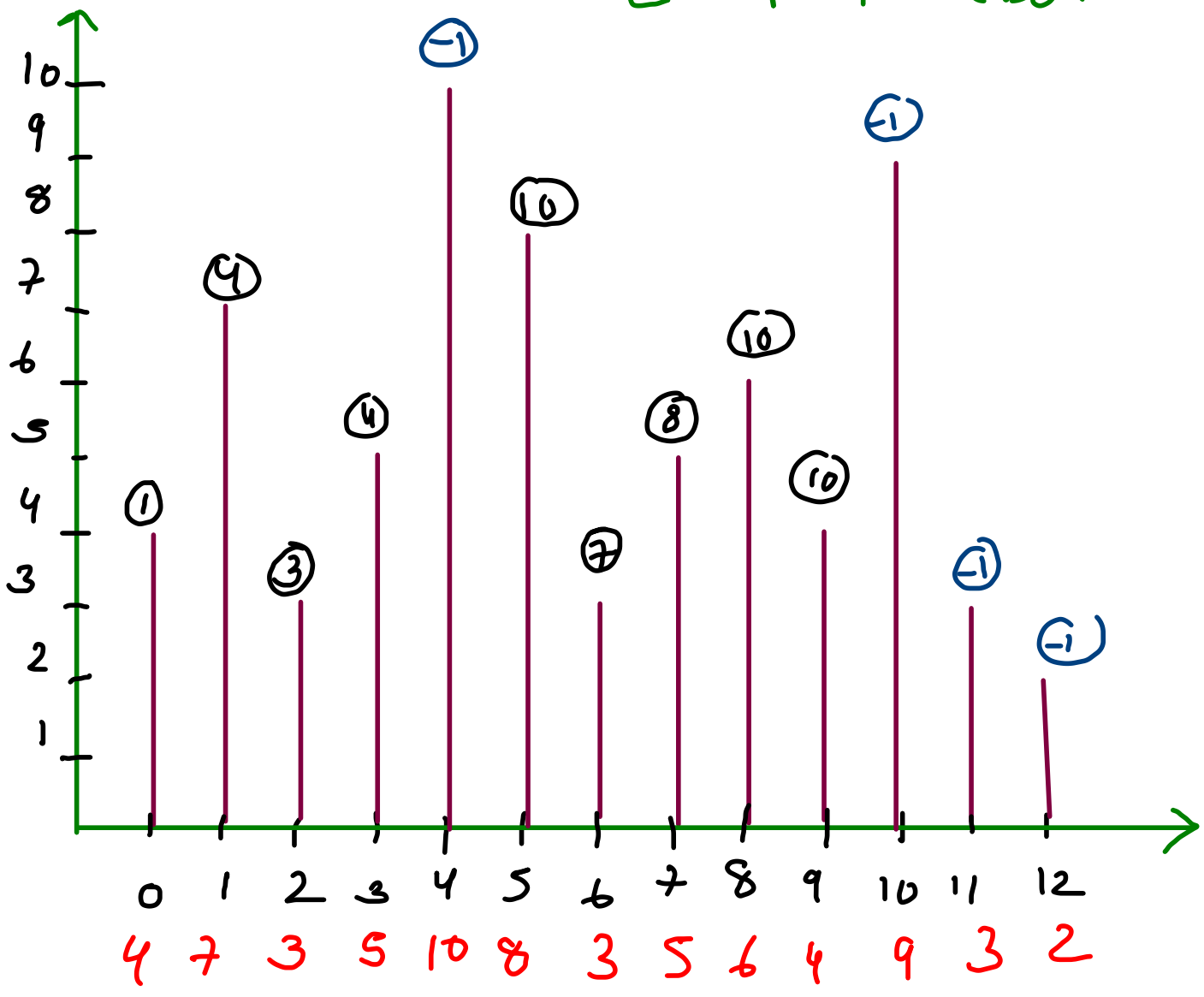


Left to right traversal algo on next page.

① While (st.peek() < curr ele) pop

② push curr ele.

L to R traversal



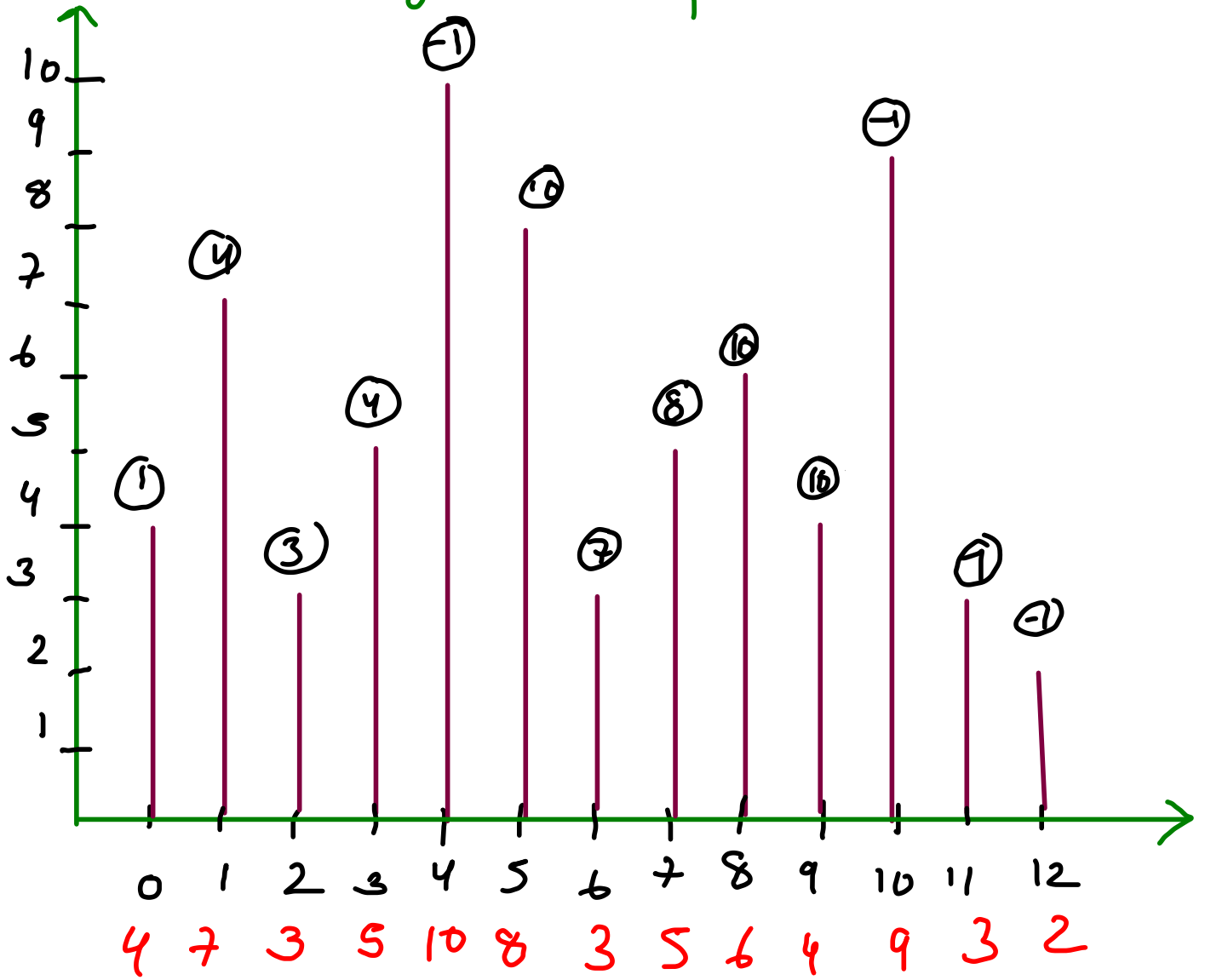
~~(12,2)~~
~~(11,3)~~
~~(10,9)~~
~~(9,14)~~
~~(8,16)~~
~~(7,15)~~
~~(6,13)~~
~~(5,8)~~
~~(4,10)~~
~~(3,5)~~
~~(2,13)~~
~~(1,7)~~
~~(0,14)~~

```java
public static int[] solve(int[] arr){
  int[] res = new int[arr.length];
  Arrays.fill(res,-1);
  //we are storing indices in the stack
  Stack<Integer> st = new Stack<>();
  st.push(0);

  for(int i=1;i<arr.length;i++) {
     if(arr[i] < arr[st.peek()]) {
         st.push(i);
     } else {
         while(st.size() > 0 && arr[st.peek()] < arr[i]) {
             int idx = st.pop();
             res[idx] = arr[i];
         }
         st.push(i);
     }
  }

  return res;
}
```

TC: O(N)

SC: O(N)

Right to Left Traversal

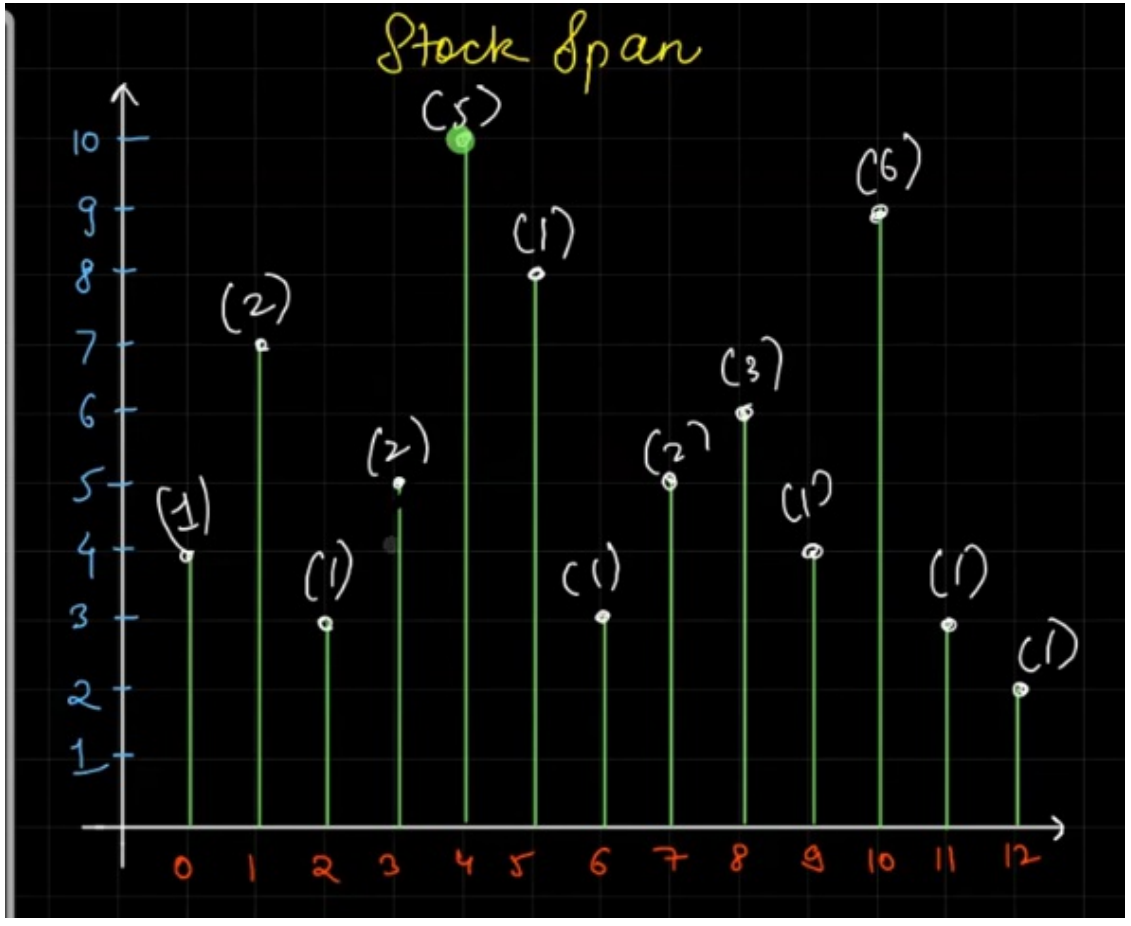


(0,4)
(1,7)
~~(2,3)~~
~~(3,5)~~
(4,10)
~~(5,8)~~
~~(6,3)~~
~~(7,5)~~
~~(8,6)~~
~~(9,4)~~
~~(10,9)~~
~~(11,3)~~
~~(12,2)~~

```java
public static int[] solve(int[] arr){
  int[] res = new int[arr.length];
  Stack<Integer> st = new Stack<>();
  res[res.length-1] = -1;
  st.push(res.length-1);

  for(int i=res.length-2;i>=0;i--) {
      if(arr[st.peek()] > arr[i]) {
          res[i] = arr[st.peek()];
      } else {
          while(st.size() > 0 && arr[st.peek()] < arr[i]) {
              st.pop();
          }
          if(st.size() == 0) res[i] = -1;
          else res[i] = arr[st.peek()];
      }
      st.push(i);
  }

  return res;
}
```

Right to Left traversal.

Stock Span (NGE to left → variation)



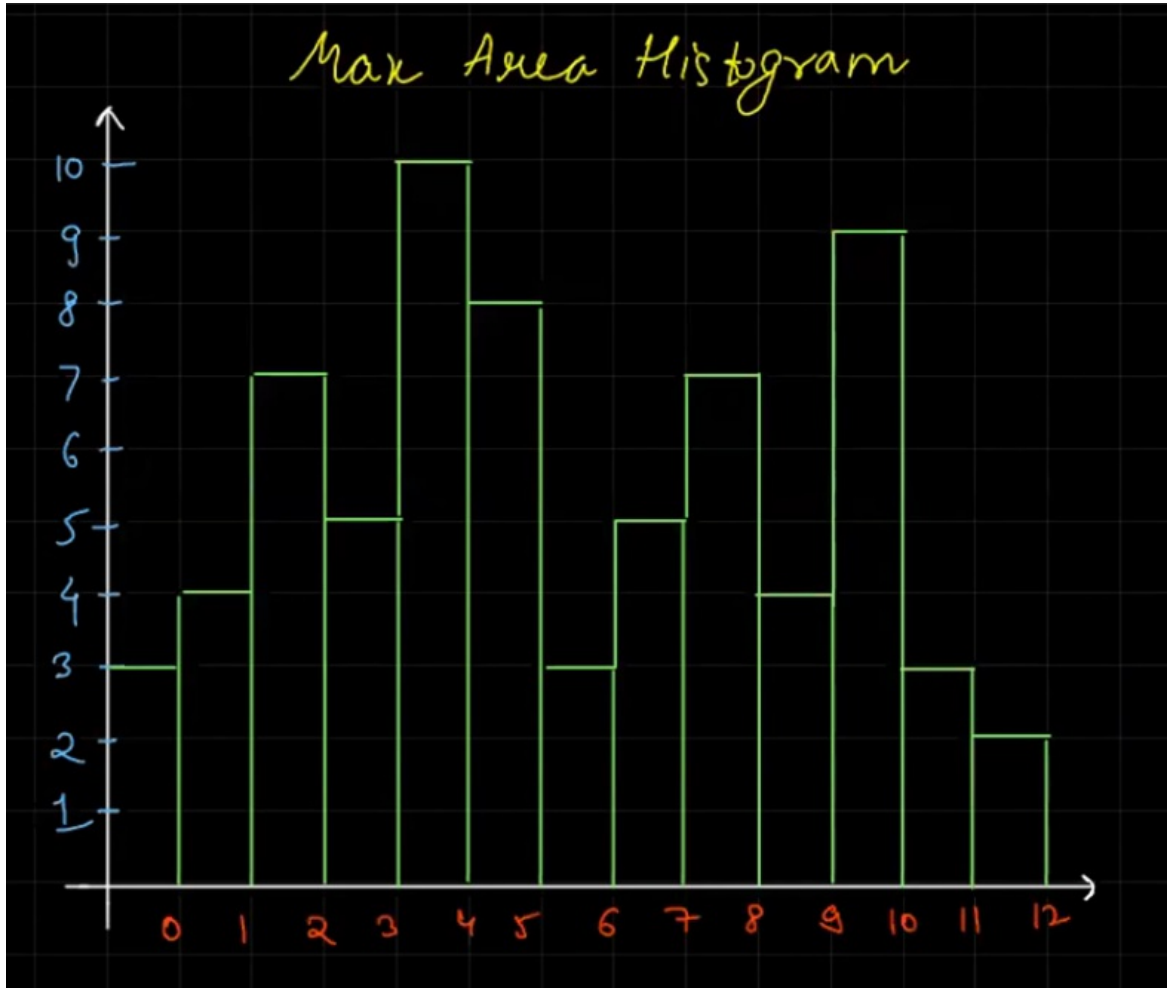
Nge to the left humne kitni door hai Index wise, that is Stock span.

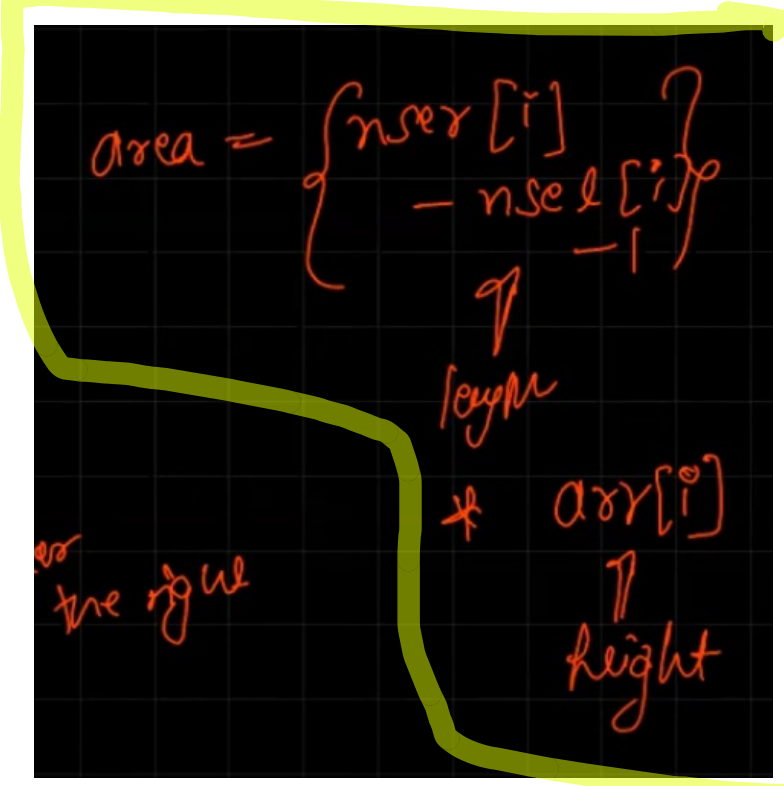
O(N) TC O(N) SC

```java
public static int[] solve(int[] arr){
   int[] res = new int[arr.length];
  Stack<Integer> st = new Stack<>();

  for(int i=0;i<arr.length;i++) {
      while(st.size() > 0 && arr[st.peek()] < arr[i]) {
          st.pop();
      }

      if(st.size() == 0) {
          res[i] = i + 1;
      } else {
          res[i] = i - st.peek();
      }

      st.push(i);
  }

  return res;
}
```

Stock Span / NGE to left Variation

Largest Area Histogram



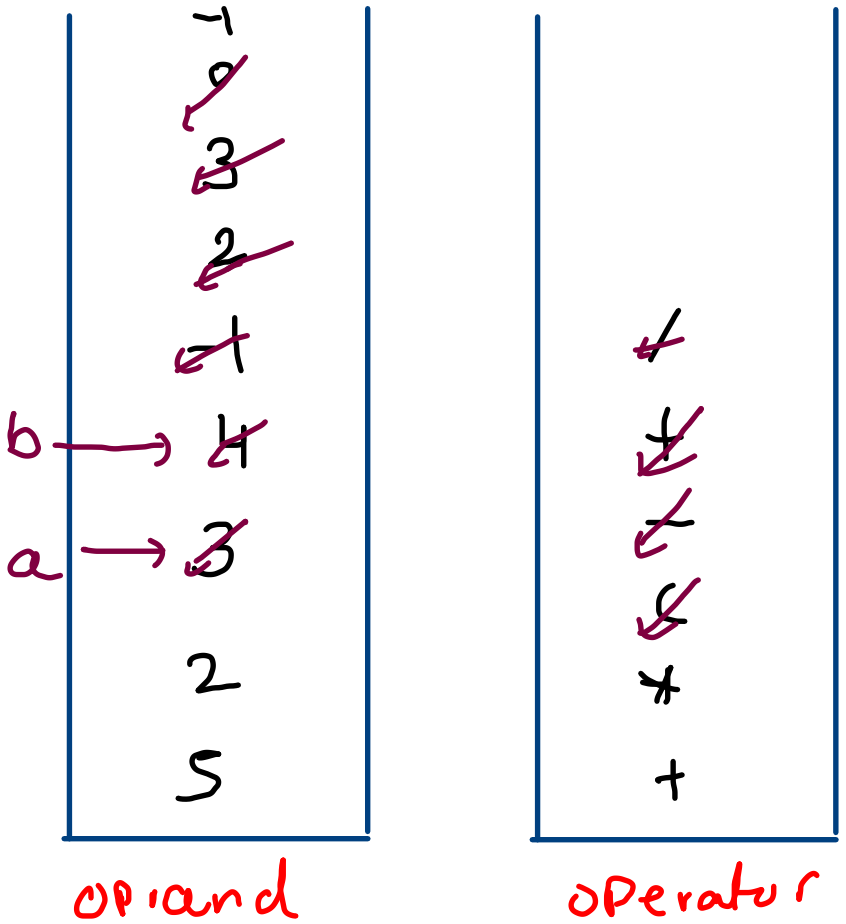


Next Smaller Element Variation.

```
class Solution {
    public int largestRectangleArea(int[] heights) {
        int n = heights.length;
        Stack<Integer> st=new Stack<>();
        int[] rb=new int[n];  //nse on the right
        int[] lb=new int[n]; //nse on the left

        st.push(heights.length-1);
        rb[heights.length-1]=heights.length;

        for(int i=heights.length-2;i>=0;i--)
        {
            while(st.size()>0 && heights[st.peek()]>=heights[i])
            {
                st.pop();
            }
            if(st.size()==0)
            {
                rb[i]=heights.length;
            }
            else
            {
                rb[i]=st.peek();
            }
            st.push(i);
        }
        st=new Stack<>();
        st.push(0);
        lb[0]=-1;
        for(int i=1;i<heights.length;i++)
        {
            while(st.size()>0 && heights[st.peek()]>=heights[i])
            {
                st.pop();
            }
            if(st.size()==0)
            {
                lb[i]=-1;
            }
            else
            {
                lb[i]=st.peek();
            }
            st.push(i);
        }

        int maxArea=0;
        for(int i=0;i<heights.length;i++)
        {
            int width=rb[i]-lb[i]-1;
            int area=heights[i]*width;
            if(area>maxArea)
            {
                maxArea=area;
            }
        }
        return maxArea;
    }
}
```

Infix Evaluation

5 + 2 * (3 - 4 + 2 / 3) * 4 + 2



(3 - 4) = -1 ↓ push

2 / 3 = 0

Priority

0. Brackets
1. Division & Multiplication
2. Addition & Subtraction

Left to Right Associativity

(+, - *, /)

Power (^) is Right to left associative.


```
if (ch >= '0' && ch <= '9')
    operand.push(ch)
else if (ch == '(')
    operator.push(ch)
else if (ch == ')')
{
    pop until '(' on top, perform, res push operand
    pop '(' also
}
else {
    while (stk.size > 0 && stk.peek() != '('
           && prec(stk.peek()) >= pred(ch))
        perf & push
    push yourself;
}
```

3

0.7 x

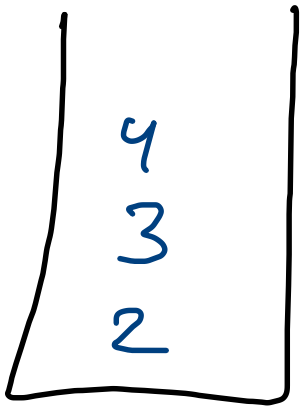
```java
public static int calculate(int a,char op, int b) {
    if(op == '+') {
        return a + b;
    } else if(op == '-') {
        return a - b;
    } else if(op == '*') {
        return a * b;
    } else {
        return a/b;
    }
}

public static int precedence(char op) {
    if(op == '*' || op =='/') return 2;
    else if(op == '+' || op == '-') return 1;
    return 0;
}
```

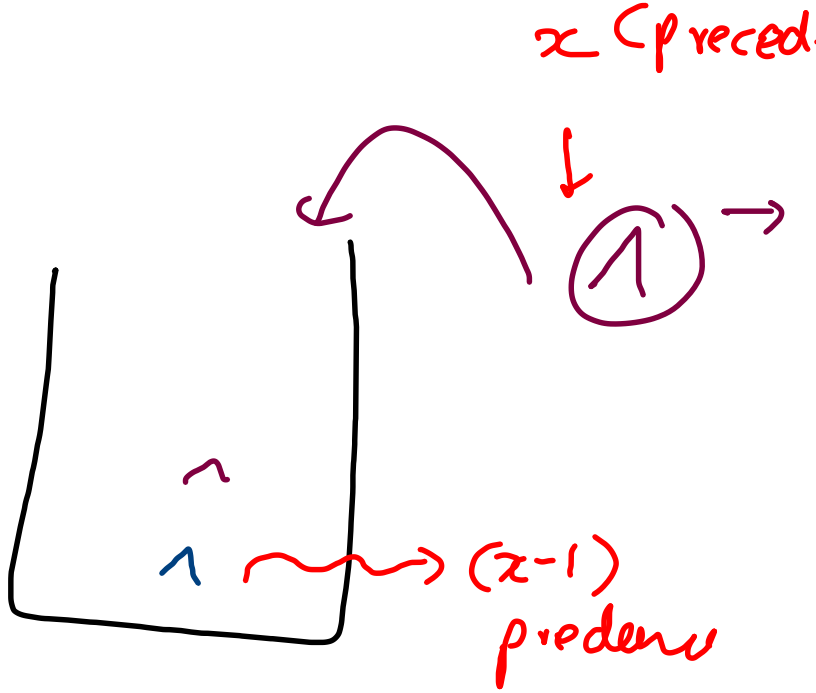
TC = O(N)

```java
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String exp = br.readLine();

    // code
    Stack<Character> operator = new Stack<>();
    Stack<Integer> operand = new Stack<>();

    for(int i=0;i<exp.length();i++) {
        char ch = exp.charAt(i);
        if(Character.isDigit(ch) == true) {
            operand.push(ch-'0');
        } else if(ch == '(') {
            operator.push(ch);
        } else if(ch == ')') {
            while(operator.peek() != '(') {
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(calculate(a,op,b));
            }
            operator.pop();
        } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            while(operator.size() > 0 && operator.peek() != '(' && precedence(operator.peek()) >= precedence(ch)) {
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(calculate(a,op,b));
            }
            operator.push(ch);
        }
    }

    while(operator.size() > 0) {
        int b = operand.pop();
        int a = operand.pop();
        char op = operator.pop();
        operand.push(calculate(a,op,b));
    }

    System.out.println(operand.pop());
}
```

Agar same que me right to left associativity bol dia to hame dekhna padega ki operator stack me jo hai aur jo stack k bahar hai unki precedence alag ho. Ex

2 ^ 3 ^ 4





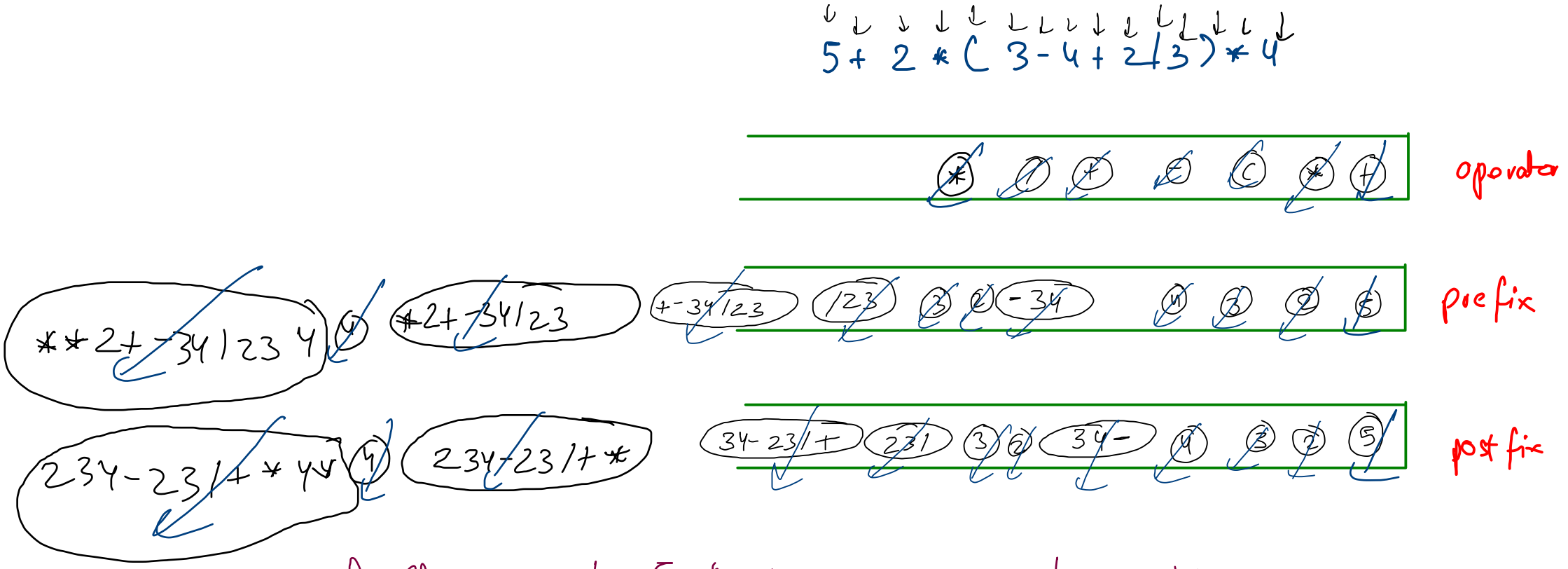
Infix Conversion

- Prefix
- Postfix

(Mostly compilers convert infix to postfix)

Prefix : operator operand 1 operand 2 (+ab)

Postfix : Operand 1 operand 2 operator (ab+)



Prefix: + 5 * * 2 + - 3 4 / 2 3 4

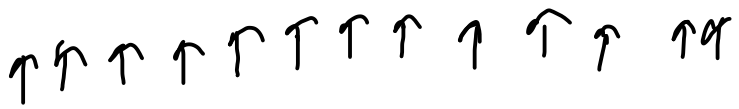
Postfix: 5 2 3 4 - 2 3 1 + * 4 * +

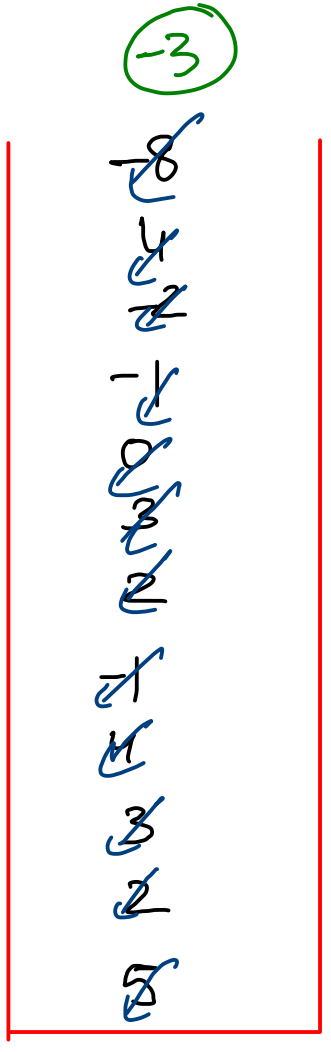
```java
import java.io.*;
import java.util.*;

public class Main{

public static int precedence(char ch) {
    if(ch == '*' || ch == '/') return 2;
    if(ch == '+' || ch == '-') return 1;
    return 0;
}

public static void performOp(Stack<String> prefix,Stack<String> postfix,char op) {
    String prefixB = prefix.pop();
    String prefixA = prefix.pop();
    String prefixRes = "" + op + prefixA + prefixB;
    prefix.push(prefixRes);

    String postfixB = postfix.pop();
    String postfixA = postfix.pop();
    String postfixRes = "" + postfixA + postfixB + op;
    postfix.push(postfixRes);

}
```

```java
public static void main(String[] args) throws Exception {
    BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
    String exp = br.readLine();

    // code
    Stack<Character> operator = new Stack<>();
    Stack<String> prefix = new Stack<>();
    Stack<String> postfix = new Stack<>();

    for(int i=0;i<exp.length();i++) {
        char ch = exp.charAt(i);

        if(ch >= 'a' && ch <= 'z') {
            prefix.push(ch + "");
            postfix.push(ch + "");
        } else if(ch == '(') {
            operator.push(ch);
        } else if(ch == ')') {
            while(operator.peek() != '(') {
                char op = operator.pop();
                performOp(prefix,postfix,op);
            }
            operator.pop();
        } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            while(operator.size() > 0 && operator.peek() != '(' && precedence(operator.peek()) >= precedence(ch)) {
                char op = operator.pop();
                performOp(prefix,postfix,op);
            }
            operator.push(ch);
        }
    }

    while(operator.size() > 0) {
        char op = operator.pop();
        performOp(prefix,postfix,op);
    }

    System.out.println(postfix.peek());
    System.out.println(prefix.peek());
 }
}
```

Postfix Evaluation and Conversion.

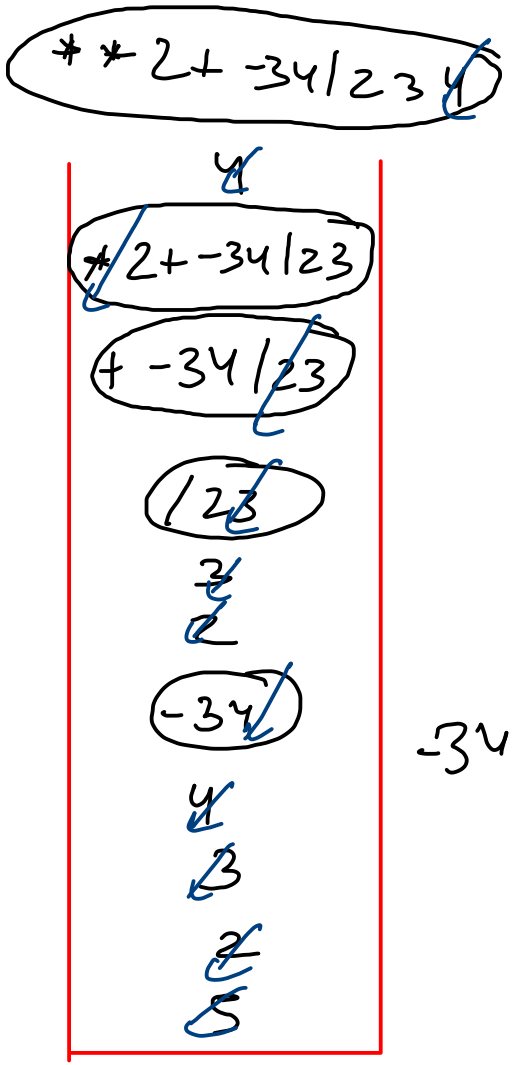
5 2 3 4 - 2 3 / + * 4 * +





evaluation

+ 5 * * 2 + - 3 4 / 2 3 4 ⇒ prefix



prefix

```java
// code
Stack<Integer> value = new Stack<>();
Stack<String> infix = new Stack<>();
Stack<String> prefix = new Stack<>();

for(int i=0;i<exp.length();i++) {
    char ch = exp.charAt(i);

    if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
        int v2 = value.pop();
        int v1 = value.pop();
        int val = operation(v1,ch,v2);
        value.push(val);

        String inv2 = infix.pop();
        String inv1 = infix.pop();
        String inv = "(" + inv1 + ch + inv2 + ")";
        infix.push(inv);

        String prev2 = prefix.pop();
        String prev1 = prefix.pop();
        String prev = "" + ch + prev1 + prev2;
        prefix.push(prev);
    } else {
        value.push(ch - '0');
        infix.push("" + ch);
        prefix.push("" + ch);
    }
}

System.out.println(value.pop());
System.out.println(infix.pop());
System.out.println(prefix.pop());
```

```java
public static int operation(int val1, char op, int val2) {

    if(op == '+') {
        return val1 + val2;
    } else if(op == '-') {
        return val1 - val2;
    } else if(op == '*') {
        return val1 * val2;
    }

    return val1/val2;
}
```

Basic Calculator. 1, 2 & 3 → Lintcode

Leetcode

Ask the interviewer whether the string can or cannot the following ( ), Spaces, More than 1 digit nos, unary operators like (-5) etc. further are there any operators apart from arithmetic operators that have associativity R to L.

" − 3 + (−5 − (+5 + -2))" → Input String

" −3+(−5−(+5+-2))" → Input string

Since the res was empty initially, & we had '−' sign, we add a 0 before it

res = 0−3+(

" −3+(−5−(+5+-2))"

Since '−' is just after the opening bracket, add 0 before it.

res = 0−3+(0−5−(

" −3+(−5−(+5+-2))"

Since '+' is just after the opening bracket, we can ignore it

res = 0−3+(0−5−(5+

" −3 +(−5 −(+5 +−2))"

Since '−' is just after '+' change the + symbol in resultant to '−'.

Res = 0−3 + (0−5 −(5−2))

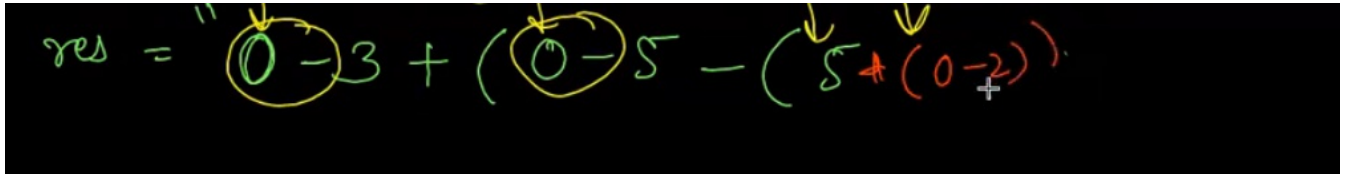
→ Evaluating this string is easy & we have it in infix evaluation. Just have to make some changes.

"-3+(-5-(+5+-2))"

Before this '-' we have '+'. We can also have *.

So, the safer thing to do when there is an operator

before - is to take it into brackets

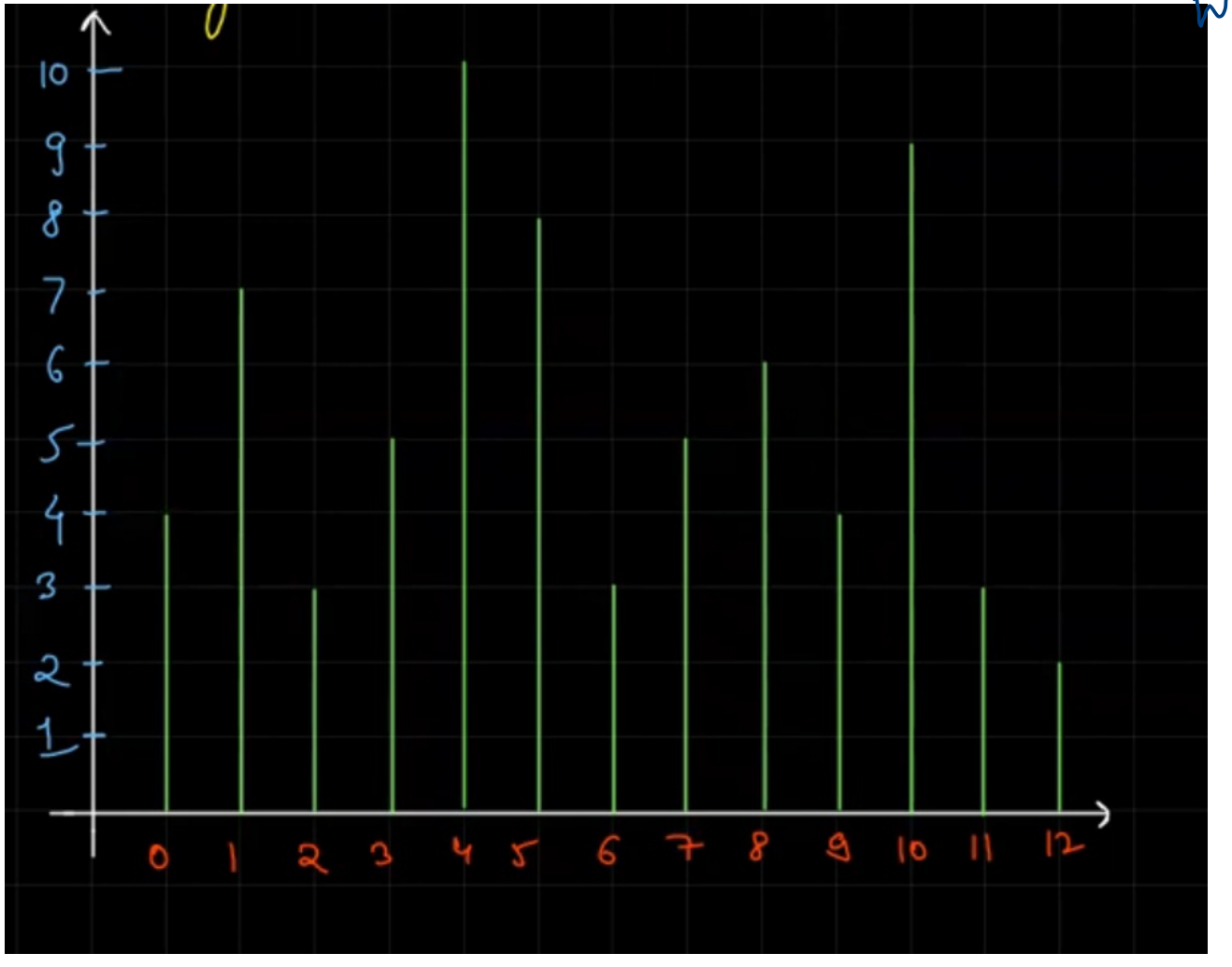


```java
class Solution {
    int performOp(int a, char op, int b) {
        int out = 0;

        switch(op)
        {
            case '+': out = a + b; break;
            case '-': out = a - b; break;
            case '*': out = a * b; break;
            case '/': out = a / b; break;
        }
        return out;
    }

    int precedence(char op)
    {
        if(op == '+' || op == '-') return 1;
        else if (op == '*' || op == '/') return 2;
        else return 0;
    }
```

```java
    public String refine(String s){
        StringBuilder str = new StringBuilder("");

        for(int i=0; i<s.length(); i++){
            char ch = s.charAt(i);

            if(ch == ' ') continue;
            if(ch == '+'){
                if(str.length() == 0 || str.charAt(str.length() - 1) == '('){
                    // unary +
                    str.append("0+");
                } else if(str.charAt(str.length() - 1) == '+' || str.charAt(str.length() - 1) == '-')
                    continue;
                else str.append('+');
                }

            else if(ch == '-'){
                if(str.length() == 0 || str.charAt(str.length() - 1) == '('){
                    // unary +
                    str.append("0-");
                } else if(str.charAt(str.length() - 1) == '+')
                    str.setCharAt(str.length() - 1, '-');
                else if(str.charAt(str.length() - 1) == '-')
                    str.setCharAt(str.length() - 1, '+');
                else str.append('-');
                }

            else str.append(ch);
        }

        return str.toString();
    }
}
```

```java
public int calculate(String exp) {
    exp = refine(exp);

    Stack<Character> operator = new Stack<>();
    Stack<Integer> operand = new Stack<>();

    for(int i=0; i<exp.length(); i++){
        char ch = exp.charAt(i);

        if(ch >= '0' && ch <= '9'){
            int num = 0;
            while(i < exp.length() && exp.charAt(i) >= '0' && exp.charAt(i) <= '9'){
                num = num * 10 + (exp.charAt(i) - '0');
                i++;
            }

            // operand -> push in operand wali stack
            operand.push(num);
            i--;

        } else if(ch == '('){
            operator.push(ch);

        } else if(ch == ')'){
            while(operator.peek() != '('){
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(performOp(a, op, b));
            }

            operator.pop(); // pop opening braces as well
        } else if(ch == '+' || ch == '-' || ch == '*' || ch == '/') {
            // operator -> + or - or * or /
            while(operator.size() > 0 && operator.peek() != '('
                && precedence(operator.peek()) >= precedence(ch)){
                int b = operand.pop();
                int a = operand.pop();
                char op = operator.pop();
                operand.push(performOp(a, op, b));
            }

            operator.push(ch);
        }
    }

    while(operator.size() > 0){
        int b = operand.pop();
        int a = operand.pop();
        char op = operator.pop();
        operand.push(performOp(a, op, b));
    }

    return operand.peek();
  }
}
```

Sliding Window Maximum.



window = 4

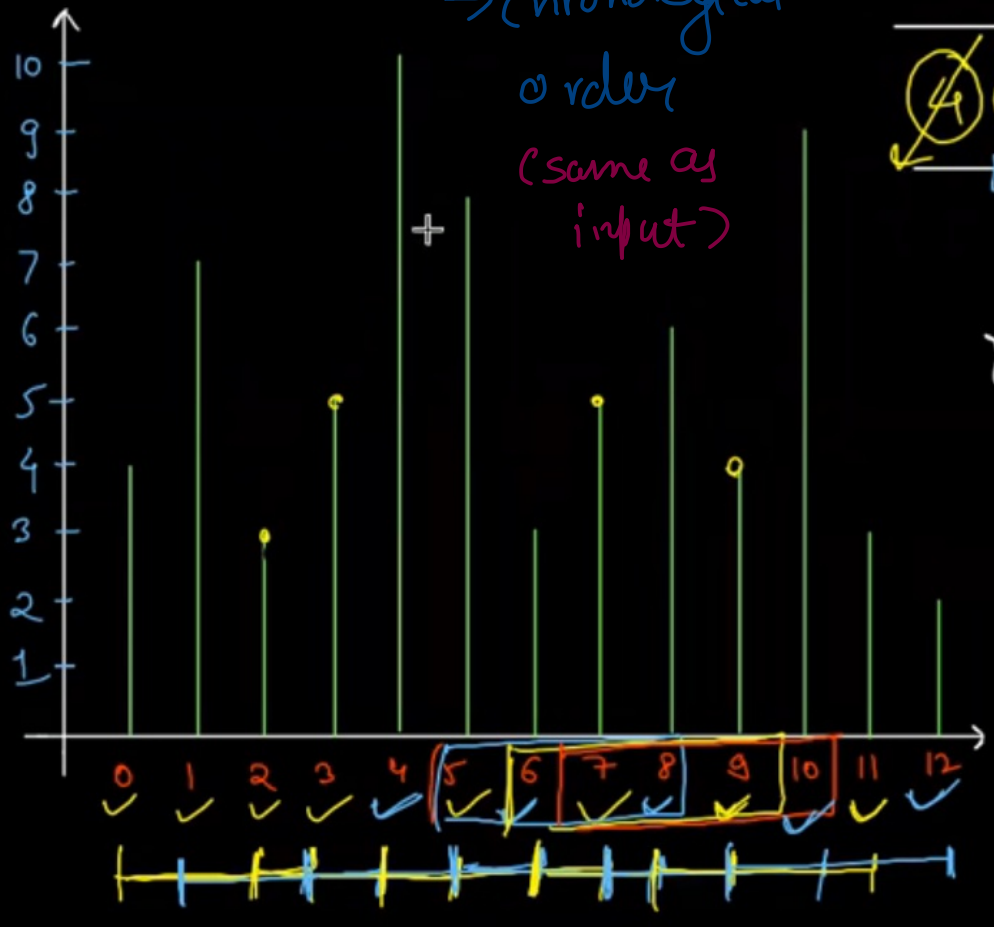
Bruteforce: O(N*k) TC

① Using DEqueue (Deque)

② Using Stack (NGE)

Deque is an ADT (Abstract Data Type). LL is itself an implementation, Deque is not.

Deque is in → decreasing order
→ chronological order (same as input)



{7, 10, 10, 10, 10, 8, 6, 9, 9, 9}

getFirst → maximum of current window

① removeFirst → which got excluded from the window

① removeLast → Smaller than ourself (Can't be max in any window)

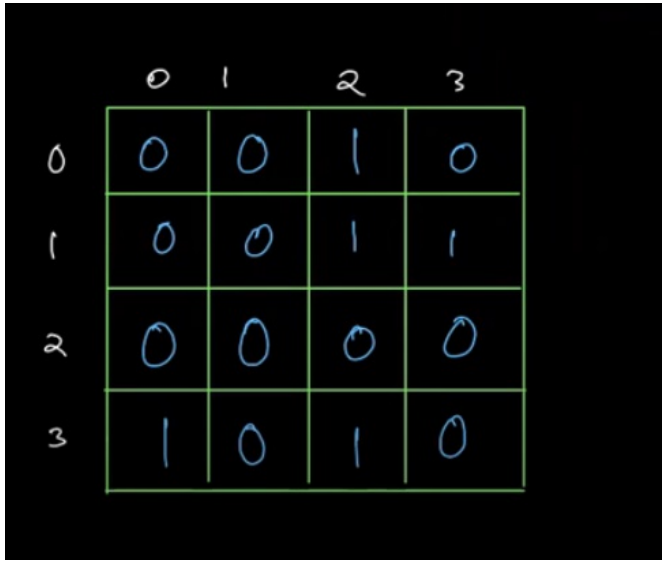
② addLast → We may the max in upcoming windows

```java
class Solution {
    public int[] maxSlidingWindow(int[] nums, int k) {
        int[] res = new int[nums.length - k + 1];
        Deque<Integer> q = new ArrayDeque<>();
        int idx = 0;
        for(int i=0;i<nums.length;i++) {
            //remove first
            if(q.size() > 0 && q.getFirst() <= i-k) {
                q.removeFirst();
            }

            //removeLast those elements that are smaller than us
            while(q.size() > 0 && nums[q.getLast()] < nums[i]) {
                q.removeLast();
            }

            //addLast -> we may be the answer of current or upcoming windows
            q.addLast(i);

            if(i >= k-1) {
                res[idx++] = nums[q.getFirst()];
            }
        }

        return res;
    }
}
```

O(N) Time

O(K) Space

Celebrity Problem.



	0	1	2	3
0	0	0	1	0
1	0	0	1	1
2	0	0	0	0
3	1	0	1	0

mat [i] [j]

- 1 → ith person knows jth person
- 0 → ith person does not know jth person

Celebrity → Who knows nobody
→ everybody knows him/her.

No celebrity ✓ Possible

More than 1 celebrity ✗ Not possible.

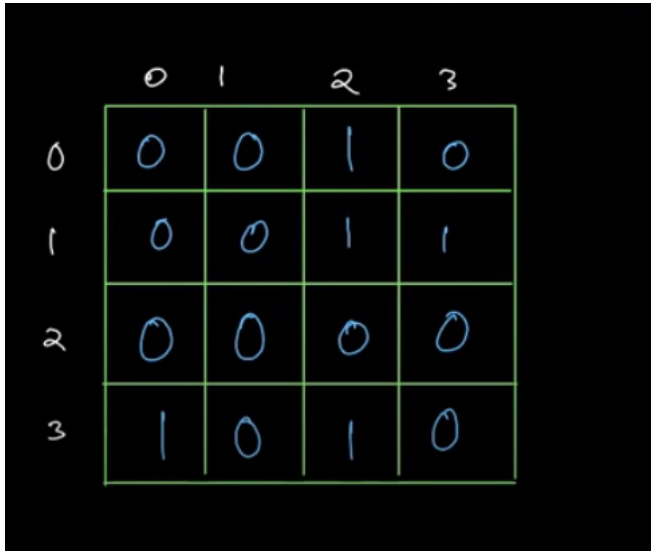
Uski Row me all 0s

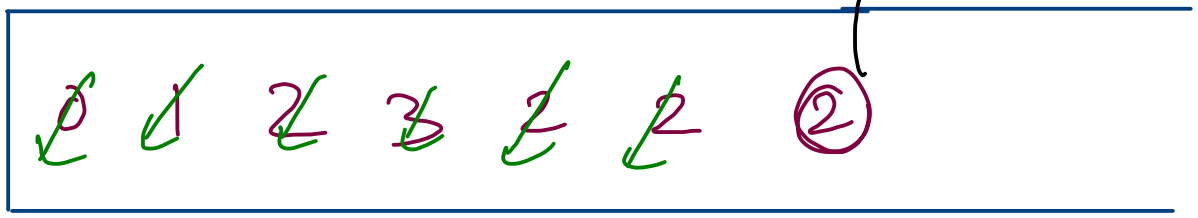
Uske col me all 1s except itself.

	0	1	2	3
0	0	0	0	0
1	0	0	1	1
2	0	0	0	0
3	1	0	1	0

We do not have any celebrity. Hence 0 celebrity is possible.

Let us say x and y both are celebrities. So, x does not know y & y does not know x. However since x is a celebrity, y should know x & vice-versa. Hence, there is a contradiction. So, we cannot have more than 1 celebrity.





Last remaining ek may or may not be a celeb. So, we have to check for it.

Possible Celebrities.

This is an elimination technique.

$T(n) = T(n-1) + O(1)$

$O(N)$

if mat [2] [3] = 1

↓

2 can't be a celebrity.

if mat [2] [3] = 0

↓

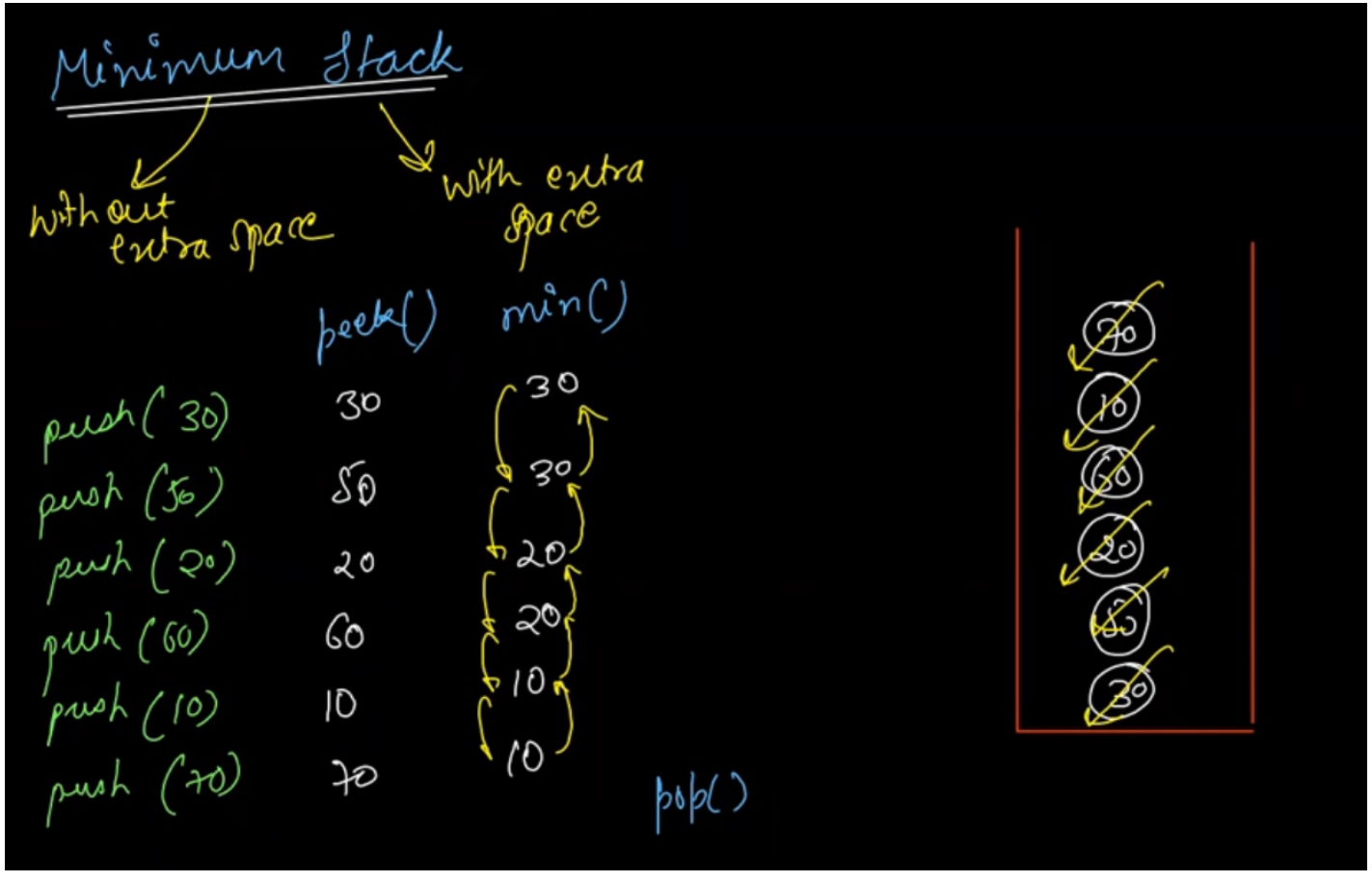
3 can't be a celeb.

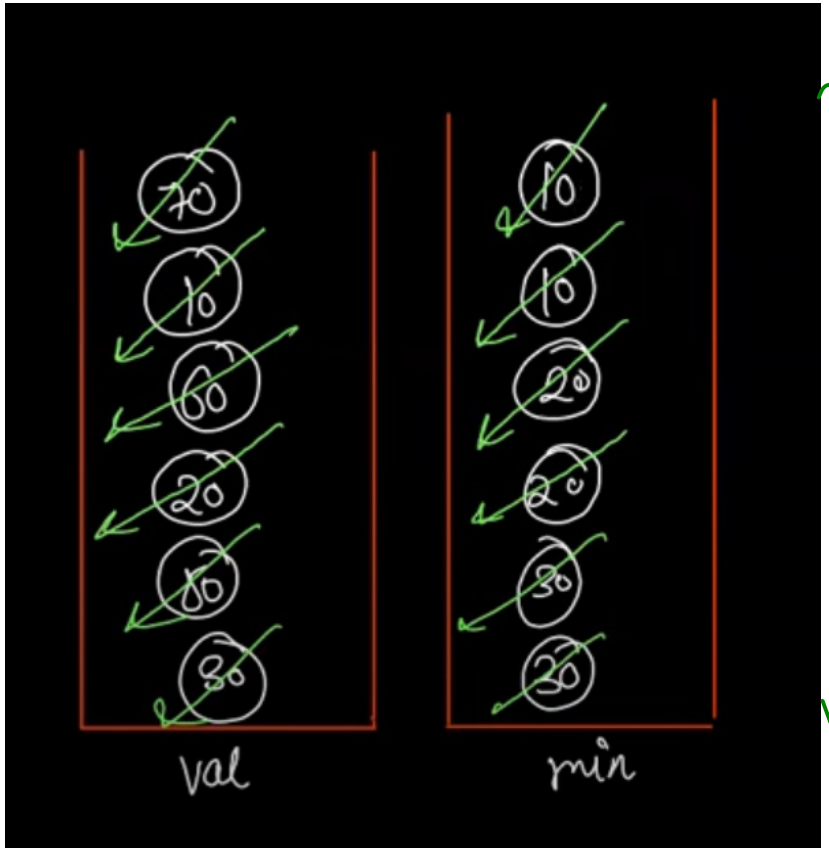
```java
class Solution
{
    //Function to find if there is a celebrity in the party or not.
    int celebrity(int M[][], int n)
    {
        // code here
        Stack<Integer> stk = new Stack<>();

        for(int i=0;i<M.length;i++) {
            stk.push(i);
        }

        while(stk.size() > 1) {
            int y = stk.pop();
            int x = stk.pop();
            if(M[x][y] == 1) {
                stk.push(y); //y may or may not be a celeb
            } else {
                stk.push(x); //x may or may not be a celeb
            }
        }

        int x = stk.pop();

        for(int j=0;j<M.length;j++) {
            if(j == x) continue;
            if(M[x][j] == 1) {
                return -1;
            }
        }

        for(int i=0;i<M.length;i++) {
            if(i == x) continue;
            if(M[i][x] == 0) {
                return -1;
            }
        }

        return x;
    }
}
```

Celebrity Problem Code.

Minimum Stack

- Without extra Space
- With extra Space





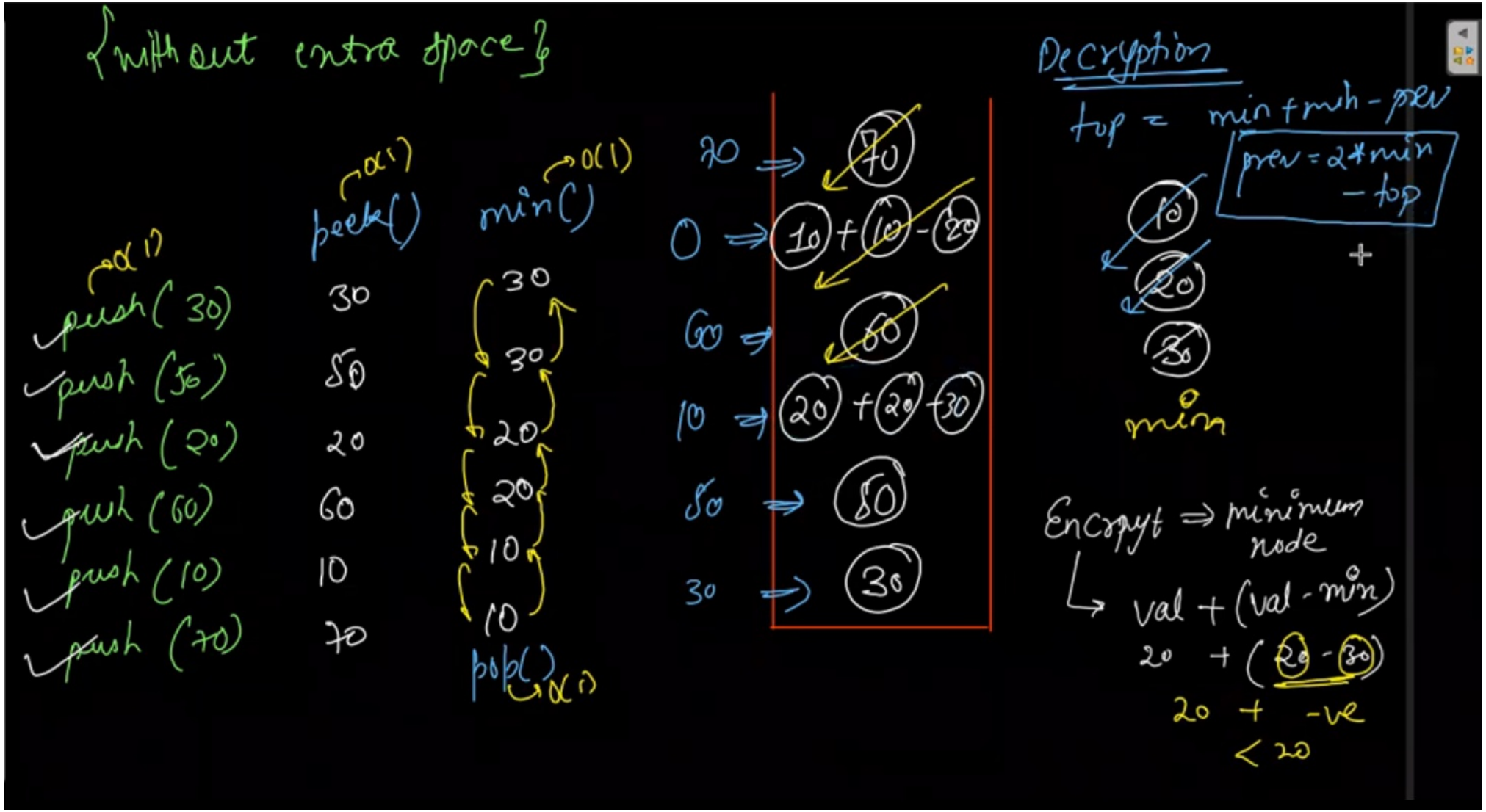
With O(N)

extra

Space.

```java
class MinStack {

    Stack<Integer> dataS;
    Stack<Integer> minS;
    int min;

    public MinStack() {
       dataS = new Stack<>();
       minS = new Stack<>();
       min = Integer.MAX_VALUE;
    }

    public void push(int val) {
        dataS.push(val);
        min = (minS.size() == 0) ? Integer.MAX_VALUE : minS.peek();
        if(min < val)
            minS.push(min);
        else
            minS.push(val);
    }

    public void pop() {
        if(dataS.size() == 0) return;
        dataS.pop();
        minS.pop();

        if(minS.size() == 0) min = Integer.MAX_VALUE;
    }

    public int top() {
        if(dataS.size() == 0) return -1;
        return dataS.peek();
    }

    public int getMin() {
        if(dataS.size() == 0) return -1;
        return minS.peek();
    }
}
```

Without extra space.



```java
class MinStack {
    Stack<Long> stk = new Stack<>();
    long min = Long.MAX_VALUE;

    public void push(int val) {
        if(stk.size() == 0) {
            stk.push(1l * val);
            min = (long)val;
        } else {
            //Encrypt the value if it is less than min
            if(val < min) {
                stk.push(2l * val - min);
                min = (long)val;
            } else {
                stk.push(1l * val);
            }
        }
    }

    public void pop() {
        if(stk.size() == 0) return;

        //check if top is encrypted
        if(stk.peek() < min) {
            //decrypt the prev min
            min = 2l * min - stk.pop();
        } else {
            stk.pop();
        }
    }

    public int top() {
        if(stk.size() == 0) return - 1;

        //check if top is encrypted
        if(stk.peek() < min) {
            //decrypt the prev min
            return (int)min;
        }

        long top = stk.peek();
        return (int)top;
    }

    public int getMin() {
        if(stk.size() == 0) return -1;

        return (int)min;
    }
}
```