Stack And Queue {level 1}

- Stack Basics
  - → Normal Stack
  - → Dynamic Stack
- Queue Basics
  - → Normal Queue
  - → Dynamic Queue
- Two Stacks in Array
- Stack using Queue
  - → Push efficient
  - → Pop Efficient
- Queue using Stacks
  - → Push efficient
  - → Pop efficient
- Duplicate Brackets
- Balanced Brackets

Lecture (3)

- Next Greater to Right

HW
- NGE to Left
- NSE to Right
- NSE to Left

- Stock Span
- Largest Histogram
- Sliding window Maximum

Lecture (4)

- Infix Evaluation
- Infix Conversions
- Prefix Evaluation And Conversions
- Postfix Evaluation And Conversions

Lecture (5)

- Celebrity Problem
- Minimum Stack
  - → with Extra Space
  - → w/o Extra Space
- Smallest No following Pattern

Stack & Queue

{ level ② }

{ Thursday & Weekends }
{ optional for FJP-2 }

h2 - lecture ①

→ K Stacks in Array
→ K Queue in Array
→ Circular Deque
→ Maximum Stack
→ Middle Queue

h2 - lecture ②

→ Stack - Increment Op
→ Max Frequency Stack
→ Basic Calculator - I, II, III
→ Expression Tree - Construct & Evaluate

L2 - lecture (3)

- → Reverse Stack
- → Reverse Queue
- → Sort Stack
- → Sort Queue
- → Interleave Queue

L2 - lecture (4)

- → Trapping Rain Water - I & II
- → Maximum Area Container
- → Remove K Digits
- → Remove Duplicate letters
- → Lexico Smallest Subset

L2 - lecture (5)

→ Remove Outermost Parantheses

→ Longest valid Parantheses

→ Minimum Additions

→ minimum Removals - I & II

→ Score of Parantheses

→ Reverse Substrings

L2 - lecture (6)

→ Validate Stack Sequence

→ Asteroid Collision

→ Exclusive Time of Fn

→ No of Recent Calls

→ Valid Word After Substitution

Lecture (1)

Monday 8:30 PM to 12 AM

- Stack Basics
  - → Normal Stack
  - → Dynamic Stack
- Queue Basics
  - → Normal Queue
  - → Dynamic Queue
- Two Stacks in Array
- Queue using LL
- Stack using LL

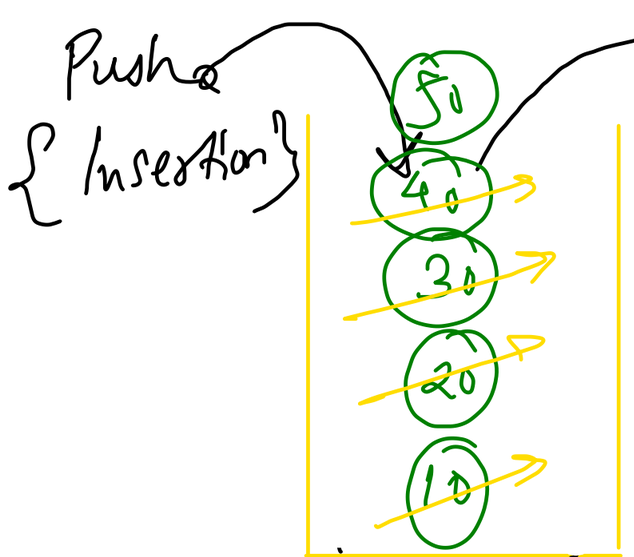
Data Structures

- → Array (1D, 2D, String)
- → ArrayList
- → Linked List

Stack

{ LIFO or FILO }

eg { Recursion call stack / Function callStack }



Stack fixed = (4)

top → peek

O(1) → push (10) First In

O(1) → push (20)

O(1) → push (30)

O(1) → push (40) Last In

O(1) → push (50) → Stack overflow

O(1) → pop() → 40 First out

O(1) → pop() → 30

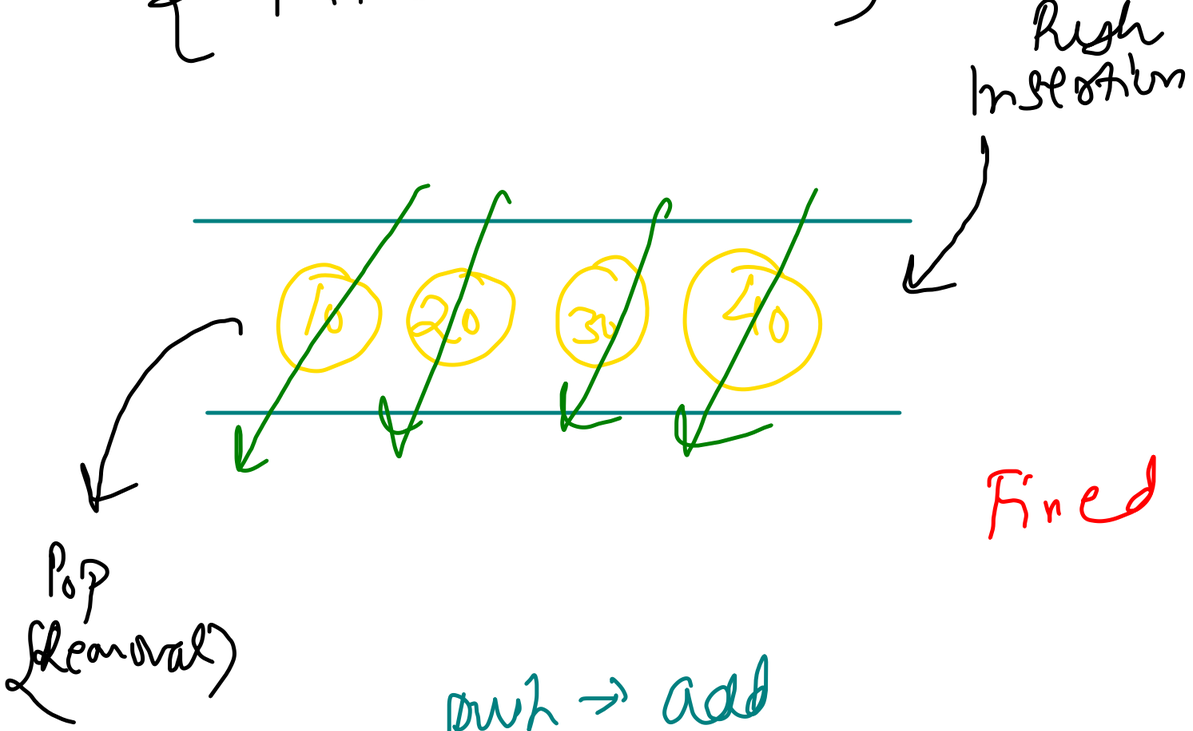
O(1) → pop() → 20

O(1) → pop() → 10 Last Out

O(1) → (top() / pop()) ? → Stack Underflow

Queue

{ FIFO or LILO }



Fixed size = 4

push → add
pop → remove
front → peek

O(1) push (10) first in

O(1) push (20)

O(1) push (30)

O(1) push (40) last in

push(50) → Queue Overflow

O(1) pop(); → 10 first out

O(1) pop(); → 20

O(1) pop(); → 30

O(1) pop(); → 40 last out

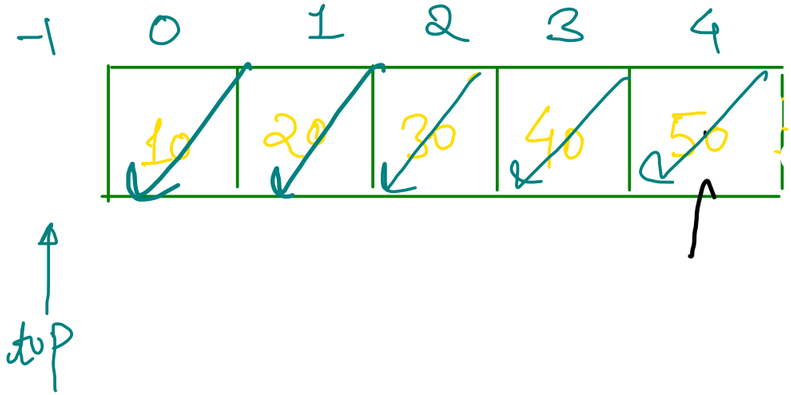
pop(); → Queue underflow (-1)

Stack

(1) Array → capacity = 5

(2) ArrayList

(3) Linked List



push(10) top++ arr[top] = val

size() → 1 (top+1)

push(20)

size() → 2

peek() → 20 return arr[top]

push(30)

peek() → 30

push(40)

push(50)

push(60) → Stack over flow top = cap - 1

60 ← pop() → int val = arr[top]
top--;
return val;

pop() → 40

pop() → 30

pop() → 20

pop() → 10

pop() → Stack underflow (top == -1)

size() → 0

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

```java
public CustomStack(int cap) {
  // Constructor -> O(1)
  arr = new int[cap];
  top = -1;
}

int size() {
    // O(1)
    return top + 1;
}

void display() {
    // O(N)
    for(int i=top; i>=0; i--){
        System.out.print(arr[i] + " ");
    }
    System.out.println();
}
```

```java
void push(int val) {
    // O(1)
    if(top == arr.length - 1){
        // Stack Overflow
        System.out.println("Stack overflow");
        return;
    }

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

size() == arr.length

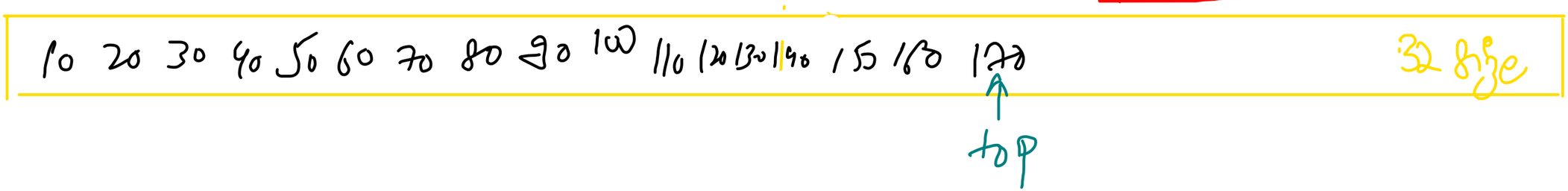
```java
int pop() {
    // O(1)
    if(top == -1){
        // Stack Underflow
        System.out.println("Stack underflow");
        return -1;
    }

    int val = arr[top];
    top--;
    return val;
}

int top() {
    // O(1)
    if(top == -1){ }

    int val = arr[top];
    return val;
}
```

size() == 0

Dynamic Stack

Doubling

Doubling



O(1) ← push (10)

O(1) ← push (20)

O(1) ← push (30)

O(1) ← push (40)

4 → 4 O(1) ← push (50)

O(1) ← push (60)

O(1) ← push (70)

O(1) ← push (80)

8 → O(8) ← push (90)

push (100) → O(1)

push (110) → O(1)

push (120) !

push (130) !

push (140) !

push (150) !

push (160) O(1)

push (170) → 16 → O(16)

33th, 65th, 129th

(Push in ArrayList)

(Avg) Amortized Time Complexity

$$= \frac{\Sigma \text{ Push Time}}{\text{No of times Push is offered}} = \frac{(1+1+1+1+\cdots\cdots 1)(+4+8+16+32)}{n}$$

n terms

$+2^2, 2^3, 2^4, 2^5, 2^6, 2^7, 2^8, 2^9, 2^{10}, 2^{11} + 2^{12}$

$$= O(1)$$

$$\frac{n+k}{n} \approx 1 \qquad \boxed{k/n \approx 1}$$

Memory Extra Allocation → least possible

$$2^x = n$$

$$x = \log_2 n$$

$$ar^{x-1} = n \qquad a = 2 \qquad r = 2$$

$$\frac{a(r^x - 1)}{(r-1)} = \frac{2(2^{\log_2 n} - 1)}{2-1}$$

$$= \frac{2(n-1)}{1} = (n)$$

$$\frac{n+k}{n} = \frac{n+n}{n} = 2$$

$O(1)$

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

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

Stack - using linked list {FILO, LIFO}

push → addFirst {O(1)}

pop → removeFirst {O(1)}

Ll{head & tail}

addFirst → O(1)

addlast → O(1)

removeFirst → O(1)

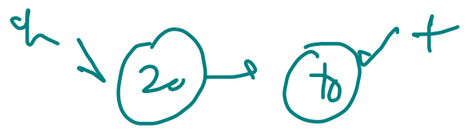
removelast → O(n)

push (10)

push (20)

push(30)

(30) pop()



(20) pop()

(10) pop()

h = t = null

pop() → Stack underflow

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

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


  int size() {
      return list.size();
  }

  void push(int val) {
      list.addFirst(val);
  }

  int pop() {
      if(size() == 0){
          System.out.println("Stack underflow");
          return -1;
      }
      return list.removeFirst();
  }

  int top() {
      if(size() == 0){
          System.out.println("Stack underflow");
          return -1;
      }
      return list.getFirst();
  }
}
```

Core

↳ DBMS → {Gate Smashers}

↳ CN → {Gate Smashers}

↳ OS → {Gate Smashers}

OOPS ↳ C++ (My notes)

Java (Nados, Abdul Bari)

Queue using LL

LL{head & tail}

addFirst → O(1)

addLast → O(1)

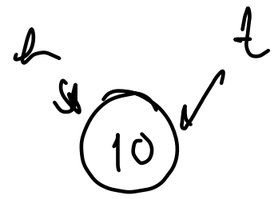
removeFirst → O(1)

removeLast → O(n)

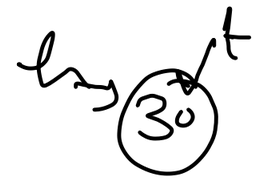
push → addLast

pop → removeFirst

push(10)



(20) pop



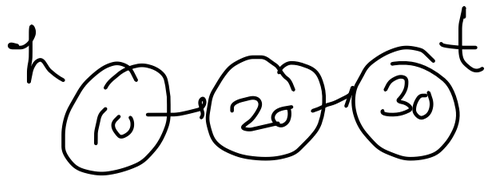
(30) pop

t = t = null

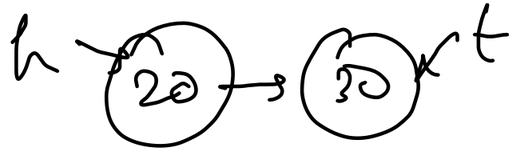
push(20)

pop → queue underflow

push(30)

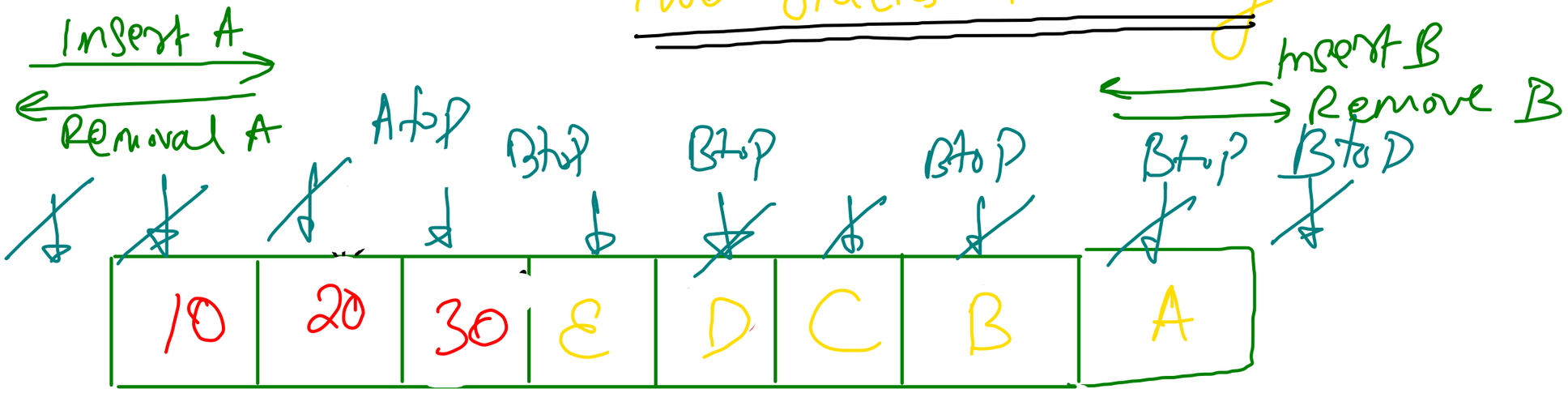


(30) pop



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

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

  int size() {
      return list.size();
  }

  void add(int val) {
      list.addLast(val);
  }

  int remove() {
      if(size() == 0) {
          System.out.println("Queue underflow");
          return -1;
      }
      return list.removeFirst();
  }

  int peek() {
      if(size() == 0) {
          System.out.println("Queue underflow");
          return -1;
      }
      return list.getFirst();
  }
```

Two Stacks in Array



1st stack

push (10)
push (20)
push (30)
push (40)
pop()

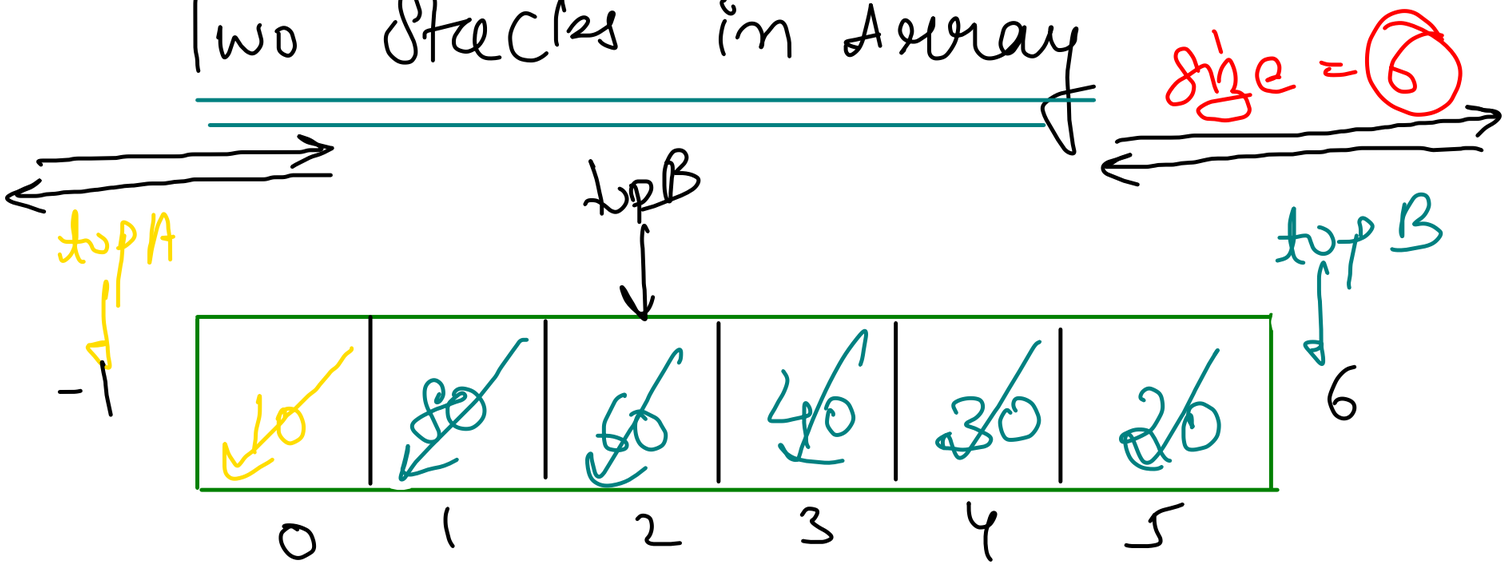
2nd stack

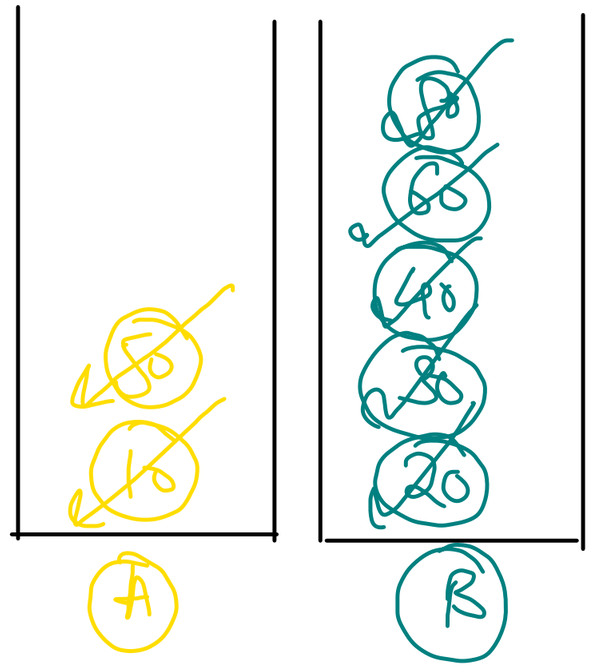
push (A)
push (B)
push (C)
push (D)
push (E)
pop()

Stack & Queue → Basics

- Stack Basics
  - → Normal Stack
  - → Dynamic Stack
  - → using 1th
- Queue Basics
  - → Normal Queue
  - → Dynamic Queue
  - → using 1th
- Two Stacks in Array
- Stack using Queue
  - → Push efficient
  - → Pop efficient
- Queue using Stacks
  - → Push efficient
  - → Pop efficient
- Duplicate Brackets
- Balanced Brackets

Two Stacks in Array





A

push(10)
push(50)
push(70) → stack overflow
pop()

pop()
pop() → stack overflow

B

push(20)
push(30)
push(40)
push(60)
push(80)

pop()
pop()
pop()
pop()
pop()
pop() → stack underflow

```java
int[] data;
int top1;
int top2;

public TwoStack(int cap) {
    data = new int[cap];
    top1 = -1;
    top2 = cap;
}
```

```java
int pop1() {
    if(top1 == -1){
        // Stack 1 Underflow
        System.out.println("Stack underflow");
        return -1;
    }

    int val = data[top1];
    top1--;
    return val;
}

int pop2() {
    if(top2 == data.length){
        // stack 2 underflow
        System.out.println("Stack underflow");
        return -1;
    }

    int val = data[top2];
    top2++;
    return val;
}
```

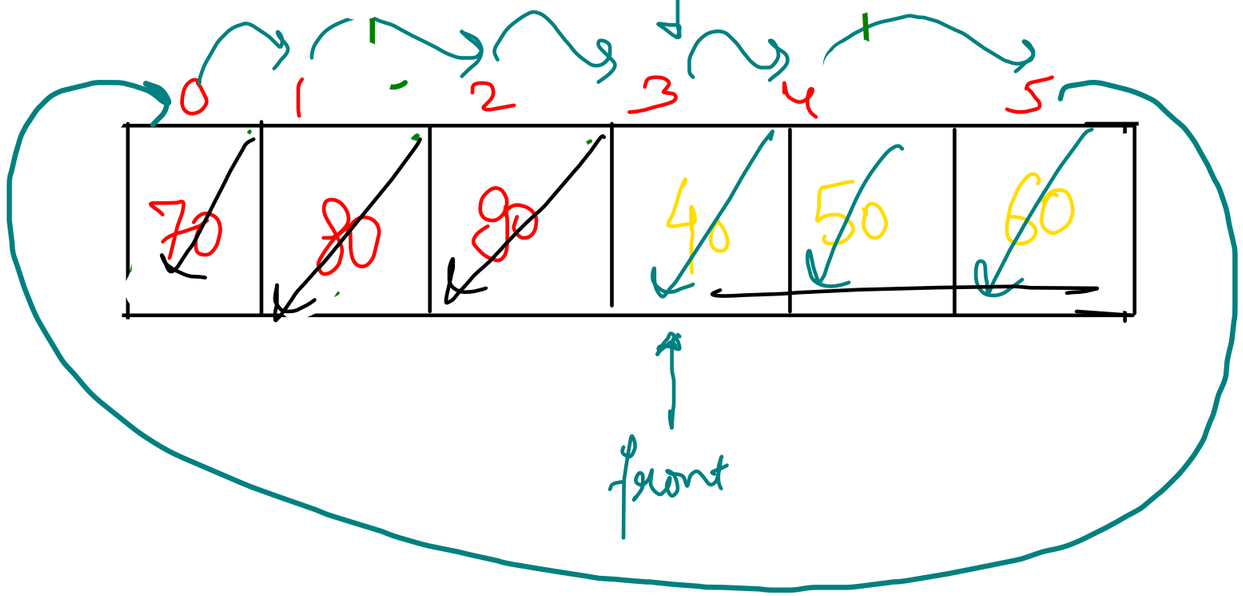
```java
void push1(int val) {
    if(size1() + size2() == data.length){
        //Stack overflow
        System.out.println("Stack overflow");
        return;
    }

    top1++;
    data[top1] = val;
}

void push2(int val) {
    if(size1() + size2() == data.length){
        //Stack overflow
        System.out.println("Stack overflow");
        return;
    }

    top2--;
    data[top2] = val;
}
```

```java
int top1() {
    if(top1 == -1) {
        System.out.println("Stack underflow");
        return -1;
    }
    return data[top1];
}

int top2() {
    if(top2 == data.length) {
        System.out.println("Stack underflow");
        return -1;
    }
    return data[top2];
}
```

```java
int size1() {
    return (top1 + 1);
}

int size2() {
    return (data.length - top2);
}
```

Normal Queue {Circular}

$$f/r = (f/r+1)\%n$$



size = ~~0 1 2 3 4 5~~ 0 1 2 3 4 5 6 5 4 3 2 1 0

cap = 6

push(10)
push(20)
push(30)
push(40)
push(50)
push(60)
push(70) → queue Overflow

pop() → 10
pop() → 20
pop() → 30
push(70)
push(80)
push(90)
push(100)
↓
pop() → 40
pop() → 50
pop() → 60
pop() → 70
pop() → 80
pop() → 90
pop() → 100

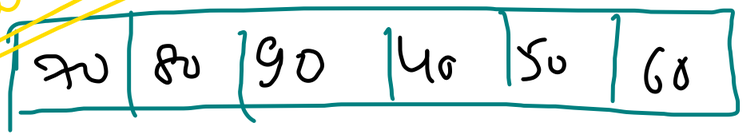
```java
int[] data;
int front;
int rear;
int size;

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

int size() {
    return size;
}
```

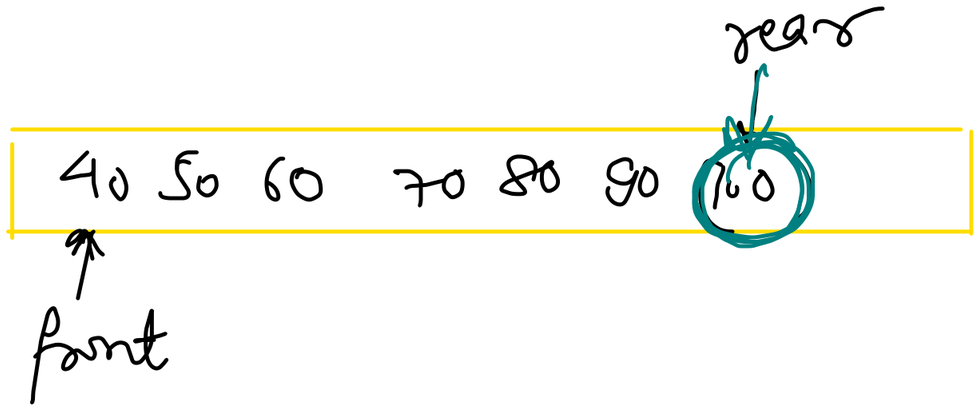
```java
void display() {
    int counter = size();
    int idx = front;
    while(counter-- > 0){
        System.out.print(data[idx] + " ");
        idx = (idx + 1) % data.length;
    }
    System.out.println();
}
```

```java
void add(int val) {
    if(size() == data.length){
        System.out.println("Queue overflow");
        return;
    }

    data[rear] = val;
    size++;
    rear = (rear + 1) % data.length;
}
```

```java
int remove() {
    if(size() == 0){
        System.out.println("Queue underflow");
        return -1;
    }

    int val = data[front];
    size--;
    front = (front + 1) % data.length;
    return val;
}

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

    int val = data[front];
    return val;
}
```

Dynamic Queue

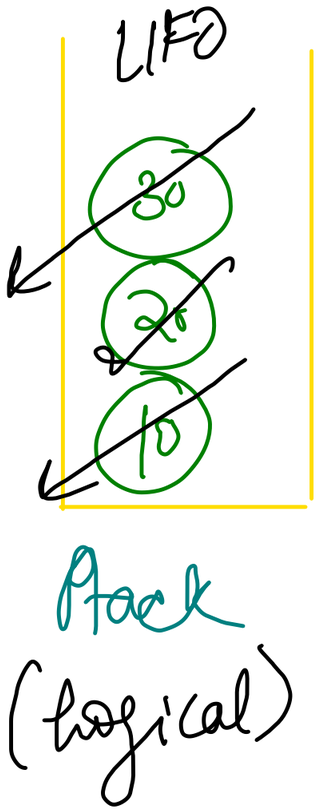




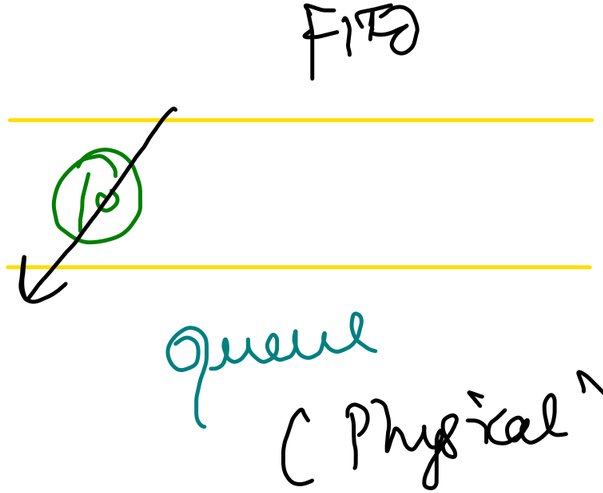
Add → Doubling

```
void add(int val) {
    if(size() == data.length){
        int[] temp = new int[2 * size()];

        int idx = front;
        for(int j=0; j<size(); j++){
            temp[j] = data[idx];
            idx = (idx + 1) % data.length;
        }

        front = 0;
        rear = size();
        data = temp;
    }

    data[rear] = val;
    size++;
    rear = (rear + 1) % data.length;
}
```

Stack using Queue



Stack
(logical)



Queue
(Physical)

(1) Stk.push(10) → O(1)

(2) Stk.peek() → 10
{1pop, 1push}

(3) Stk.push(20)

(4) Stk.peek() → 20
{2 pop, 2 push}

(5) Stk.push(30) → O(N)

(6) Stk.peek() → 30
{3 pop, 3 push}

(7) Stk.pop() → 30 → O(N)
{3 pop, 2 push}

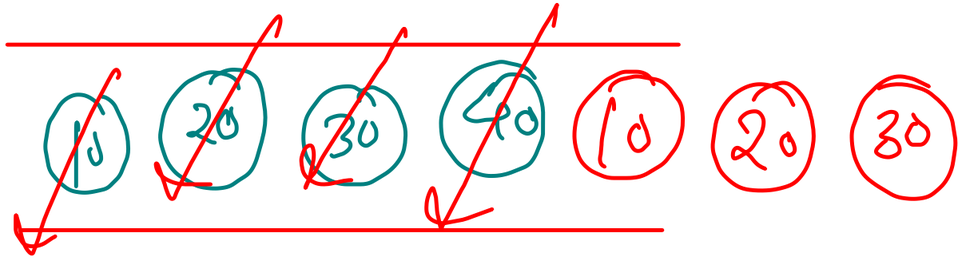
(8) Stk.pop() → 20
{2 pop, 1 push}

(9) Stk.pop() → 10
{1 pop, 0 pop}

```
Queue<Integer> q;

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

int size() {
    return q.size();
}

void push(int val) {
    q.add(val);
}
```

Queue to Stack Adapter
{ Push Efficient }

```
int pop() {
    if(size() == 0){
        System.out.println("Stack underflow");
        return -1;
    }

    int val = 0;
    for(int i=0; i<size(); i++){
        val = q.remove();
        if(i < size())
            q.add(val);
    }
    return val;
}
```



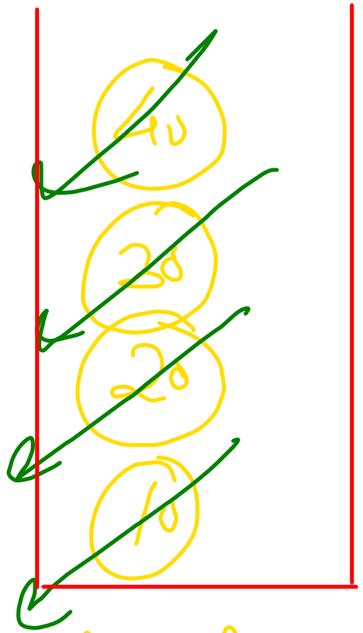
val = ∅ 10 20 30 40

i = ∅ 1 2 3

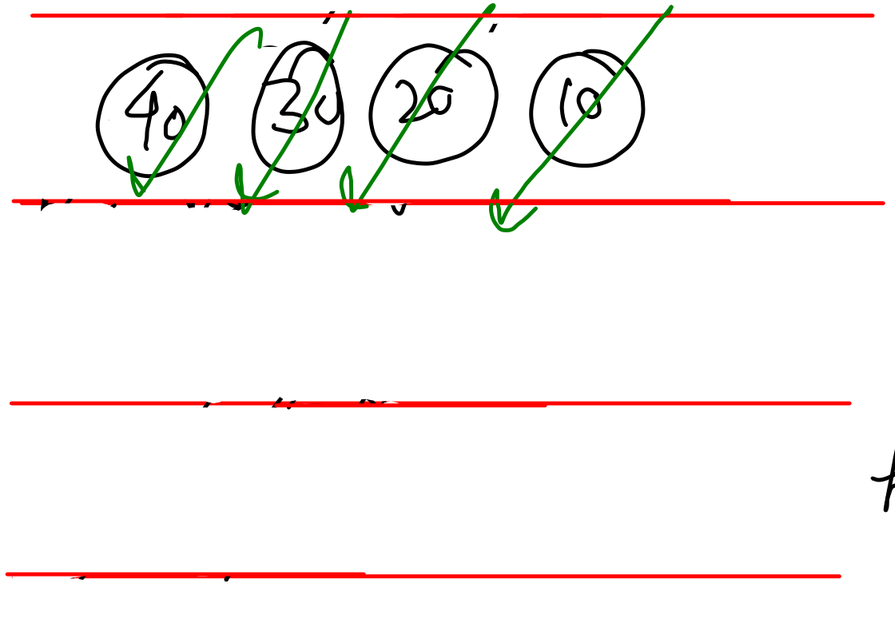
Weekend Dev

```
int top() {
    if(size() == 0){
        System.out.println("Stack underflow");
        return -1;
    }

    int val = 0;
    for(int i=0; i<size(); i++){
        val = q.remove();
        q.add(val);
    }
    return val;
}
```

Queue to Stack Adapter { Pop- Efficient }



Logical (Stack)



Main Q

Helper Q

Queue { Actual

① q.push(10) → O(N)
{ 0 pop, 1 push }

② q.push(20)
{ 1 pop, 2 push }

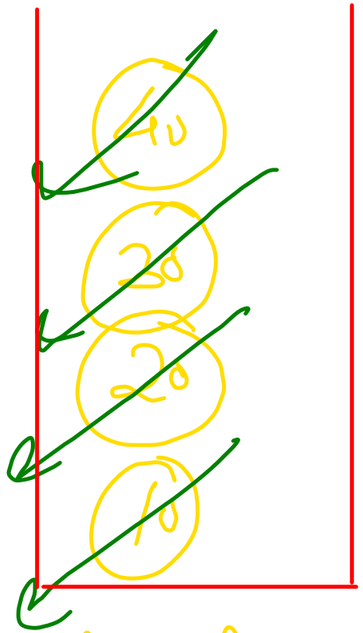
③ q.push(30)
{ 2 pop, 3 push }

④ q.push(40)
{ 3 pop, 4 push }

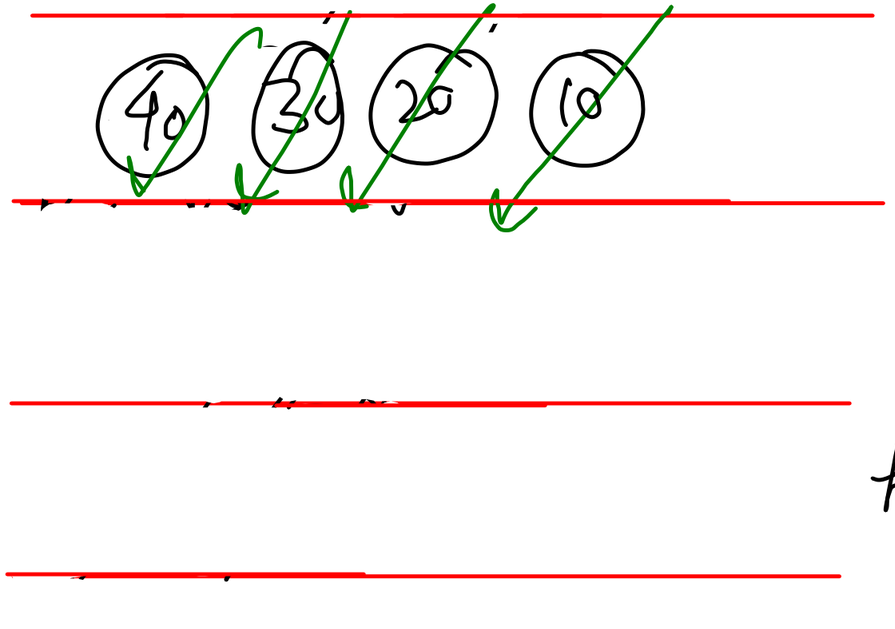
q.pop() × 4
↳ O(1)

OCJP ch-2 7,8,9

Queue to Stack Adapter { Pop- Efficient }



Logical (Stack)



mainQ

helperQ

Queue Actual

① q.push(10) → O(N)
{0pop, 1push}

② q.push(20)
{1pop, 2push}

③ q.push(30)
{2pop, 3push}

④ q.push(40)
{3pop, 4push}

q.pop() × 4
↳ O(1)

OCJP ch-2 7,8,9

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

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

    int size() {
        return mainQ.size();
    }

    void push(int val) {
        while(size() > 0){
            helperQ.add(mainQ.remove());
        }
        mainQ.add(val);
        while(helperQ.size() > 0){
            mainQ.add(helperQ.remove());
        }
    }
```

O(1)

O(N)

```java
int pop() {
    if(size() == 0){
        System.out.println("Stack underflow");
        return -1;
    }
    return mainQ.remove();
}

int top() {
    if(size() == 0){
        System.out.println("Stack underflow");
        return -1;
    }
    return mainQ.peek();
}
```

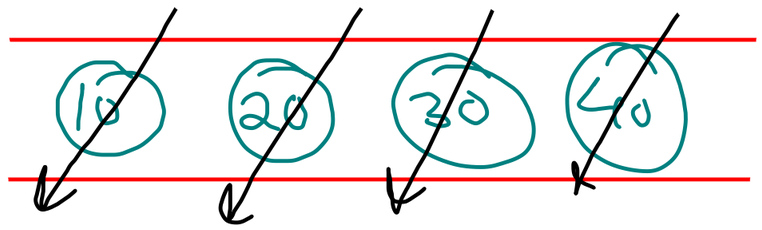
O(1)

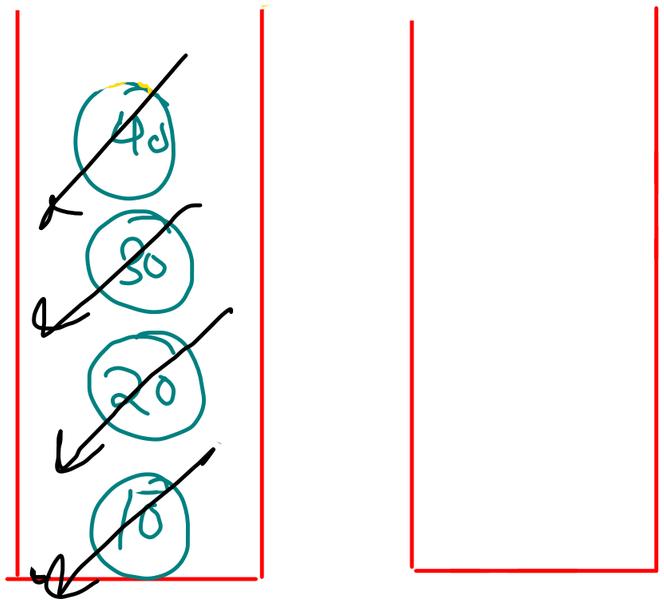
O(1)

Stack & Queue → Basics

- Stack Basics
  - → Normal Stack
  - → Dynamic Stack
  - → using th
- Queue Basics
  - → Normal Queue
  - → Dynamic Queue
  - → using th
- Two Stacks in Array
- Stack using Queue
  - → Push efficient
  - → Pop efficient
- Queue using Stacks
  - → Push efficient
  - → Pop efficient
- Duplicate Brackets
- Balanced Brackets

Queue using Stack





Actual {Stack}

helperStack

Push Efficient

Logical (Queue)

push (10) → O(1)

push (20) → O(1)

push (30) → O(1)

push (40) → O(1)

peek() → 10
↳ helper { N pop, N push }
O(N)

pop() → 10
↳ { N pop, (N-1) push }

pop() → 20
{ N pop, N-1 push }
O(N)

pop() → 30
{ N pop, N-1 push }

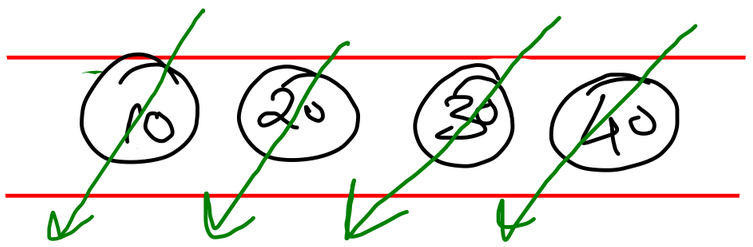
pop() → 40

pop() → Queue underflow

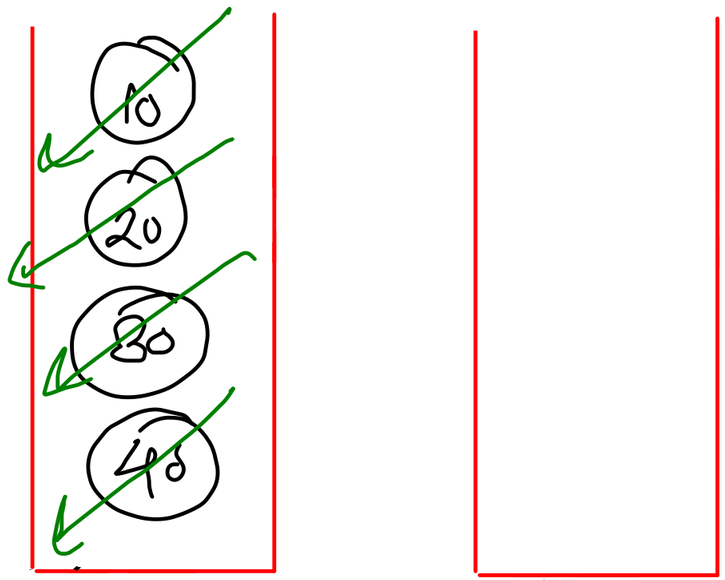
```java
public static class StackToQueueAdapter {
    Stack<Integer> mainS;
    Stack<Integer> helperS;

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

    int size() {
        return mainS.size();
    }

    void add(int val) {
        mainS.push(val);
    }
```

```java
int remove() {
    if(mainS.size() == 0){
        System.out.println("Queue underflow");
        return -1;
    }

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

    int val = helperS.pop();
    while(helperS.size() > 0)
        mainS.push(helperS.pop());

    return val;
}

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

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

    int val = helperS.peek();
    while(helperS.size() > 0)
        mainS.push(helperS.pop());

    return val;
}
```

Queue using Stack { Pop Efficient }



Logical (Queue)



(Actual)

(1) push (10)

(2) push (20)
{ insert At Bottom }
{ 1 pop, 2 push }

(3) push (30)
{ 2 pop, 3 push }

(4) push (40) { O(N) }
{ 3 pop, 4 push }

top()
pop() (5) → Stack underflow
(4) → O(1)
(1) (2) (3) 40
10 20 30

```java
Stack<Integer> mainS;
Stack<Integer> helperS;

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

int size() {
    return mainS.size();
}

void add(int val) {
    while(mainS.size() > 0){
        helperS.push(mainS.pop());
    }
    mainS.push(val);
    while(helperS.size() > 0){
        mainS.push(helperS.pop());
    }
}
```

O(1)

O(N)

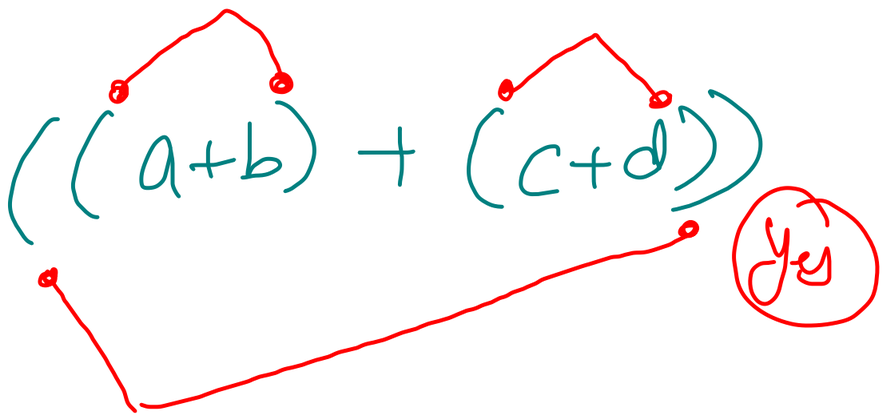
```java
int remove() {
    if(size() == 0){
        System.out.println("Queue underflow");
        return -1;
    }
    return mainS.pop();
}

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

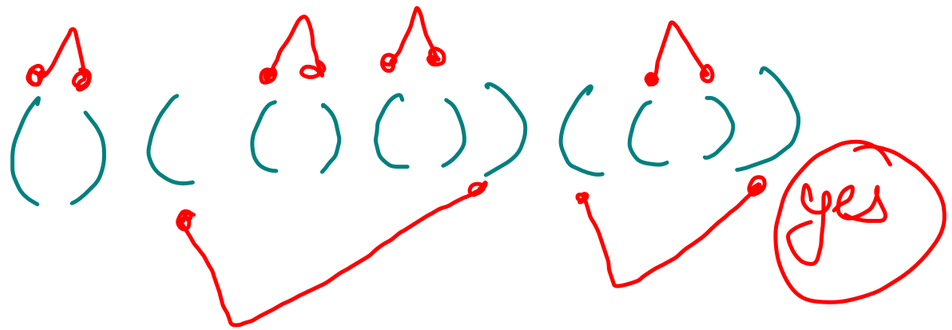
O(1)

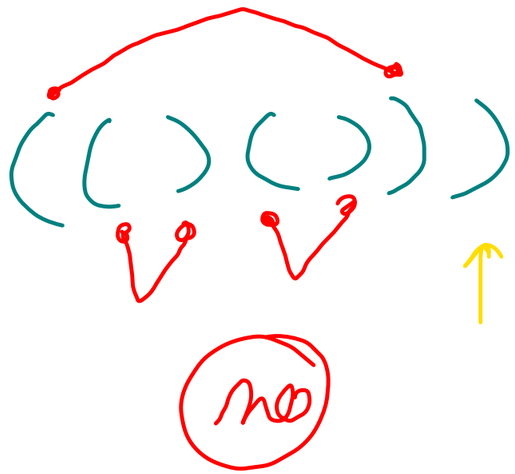
O(1)

break till 10:00 PM

Balanced Brackets



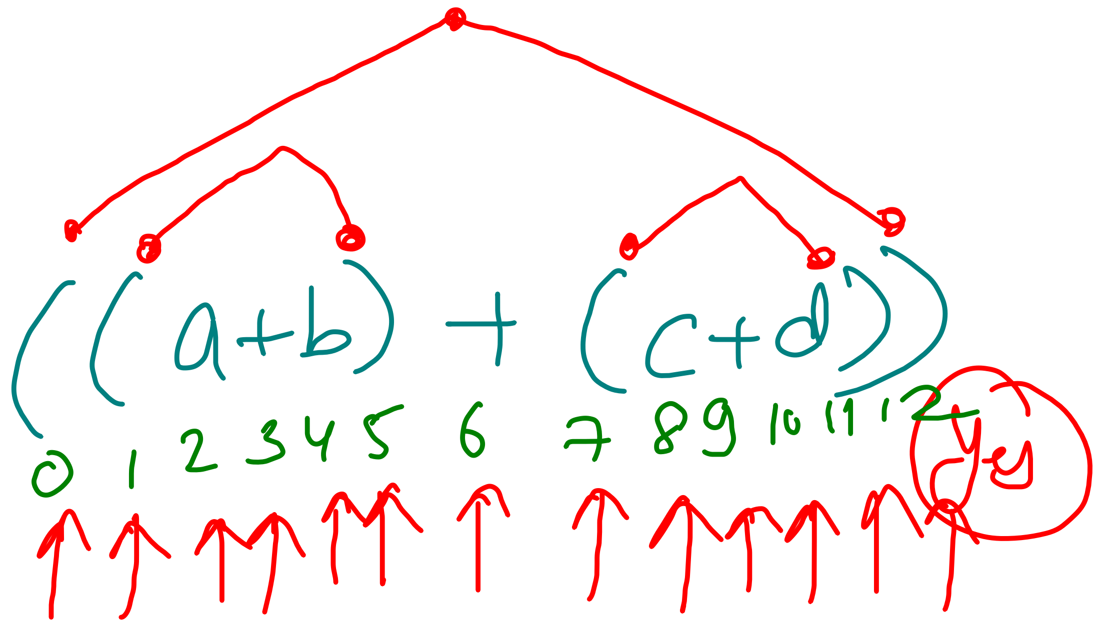


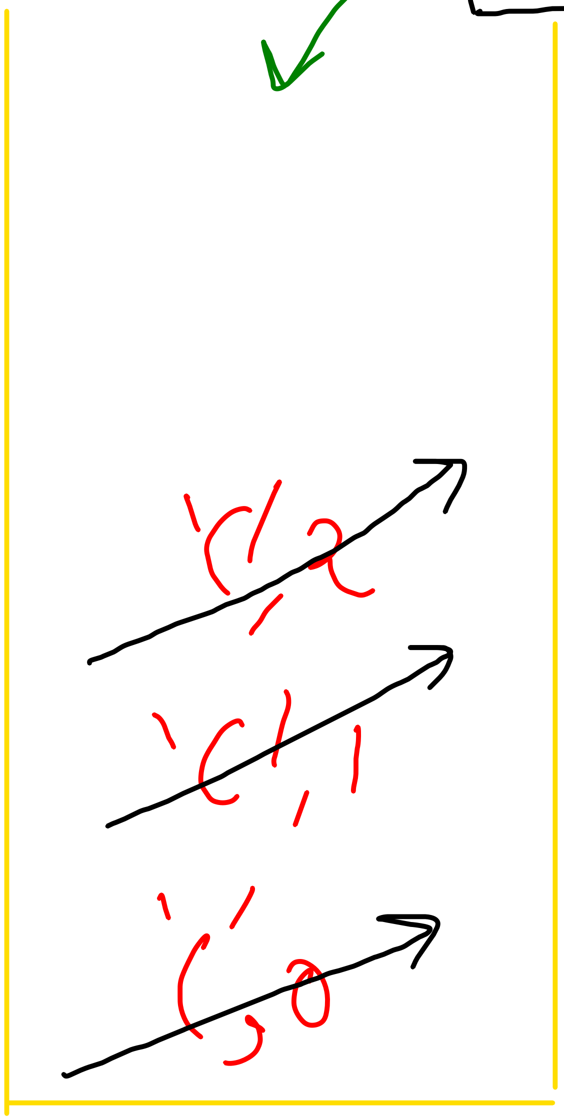


unbalanced char index

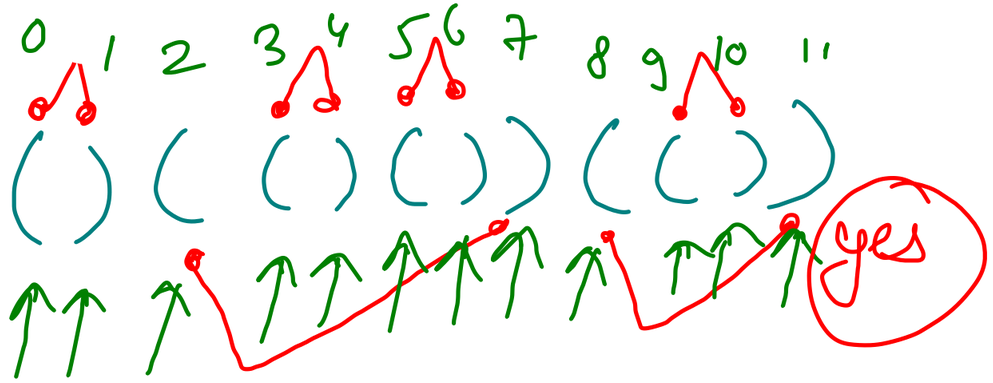
Stack<Integer>

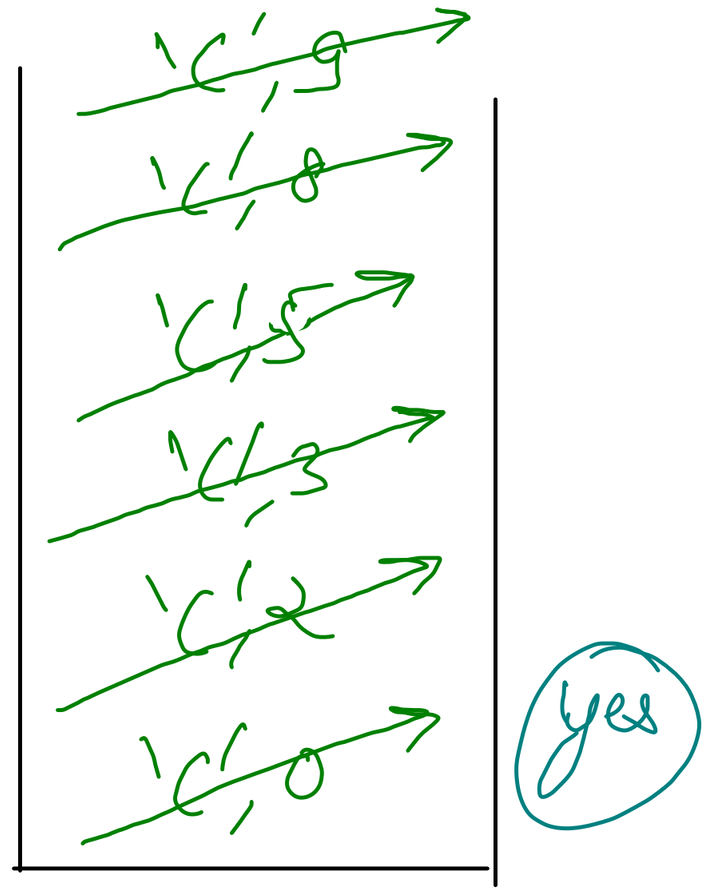
yes





1. opening braces ⇒ push
2. closing braces → stk.top() == '(' ⇒ stk.pop()
3. neither opening nor closing (a-z, 0-9, +, -, /, *)

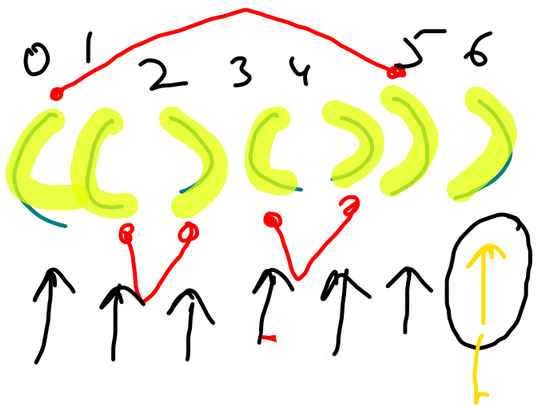




① opening braces ⇒ push

② closing braces → stk.top() == '('
⇒ stk.pop()

③ neither opening nor closing
(a-z, 0-9, +, -, /, *)

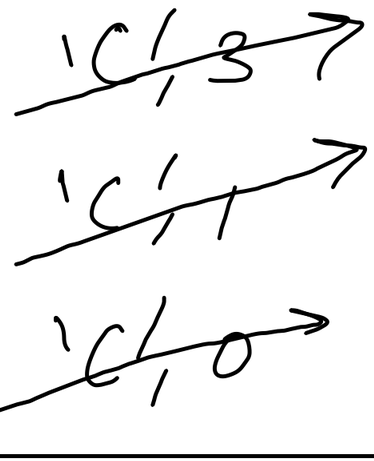


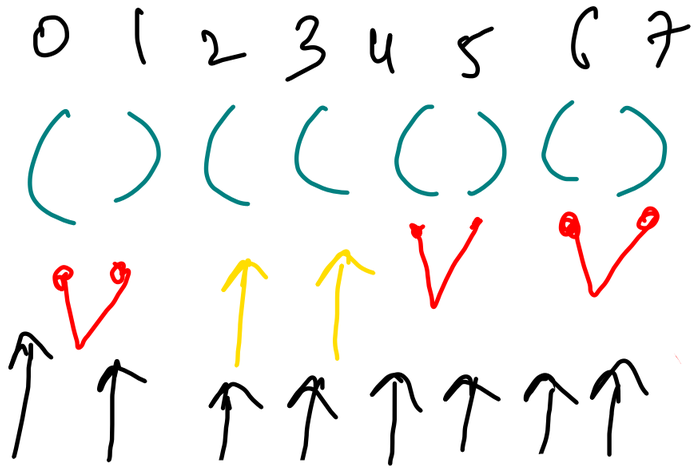
no

① opening braces ⇒ push

② closing braces → stk.top() == '('
⇒ stk.pop()
→ stk.size() == 0 ⇒ no ⇒ unbalanced

③ neither opening nor closing
(a-z, 0-9, +, -, /, *)



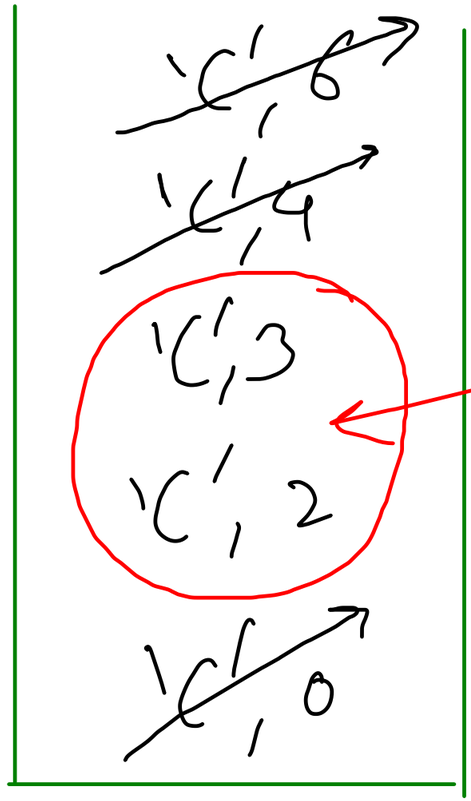


no

1. opening braces ⇒ push
2. closing braces → stk.top() == someType ⇒ stk.pop()
   → diffType, size == 0 ⇒ no
3. neither opening nor closing
   (a-z, 0-9, +, -, /, *)
   do nothing

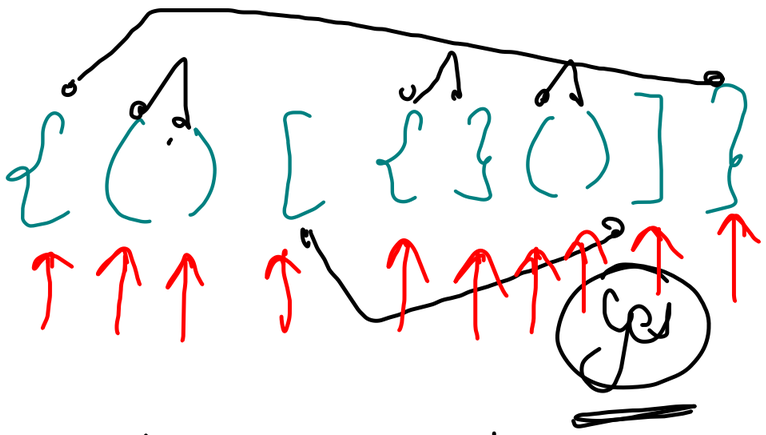
After traversal

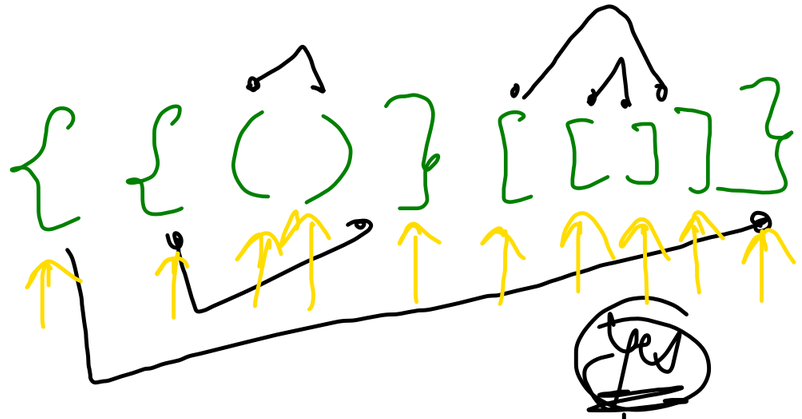
if stk.size == 0
return true
return false; no '(' unbalanced

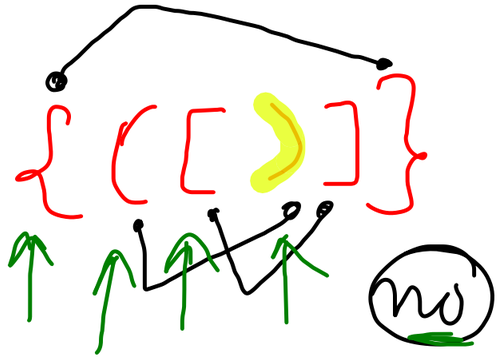


```java
Stack<Character> stk = new Stack<>();

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

    if(ch == ')'){
        if(stk.isEmpty() || stk.peek() != '('){
            return false;
        }
        stk.pop();
    } else if(ch == '}'){
        if(stk.isEmpty() || stk.peek() != '{'){
            return false;
        }
        stk.pop();
    } else if(ch == ']'){
        if(stk.isEmpty() || stk.peek() != '['){
            return false;
        }
        stk.pop();
    } else {
        stk.push(ch);
    }
}

return (stk.size() == 0) ? true : false;
```

O(N)

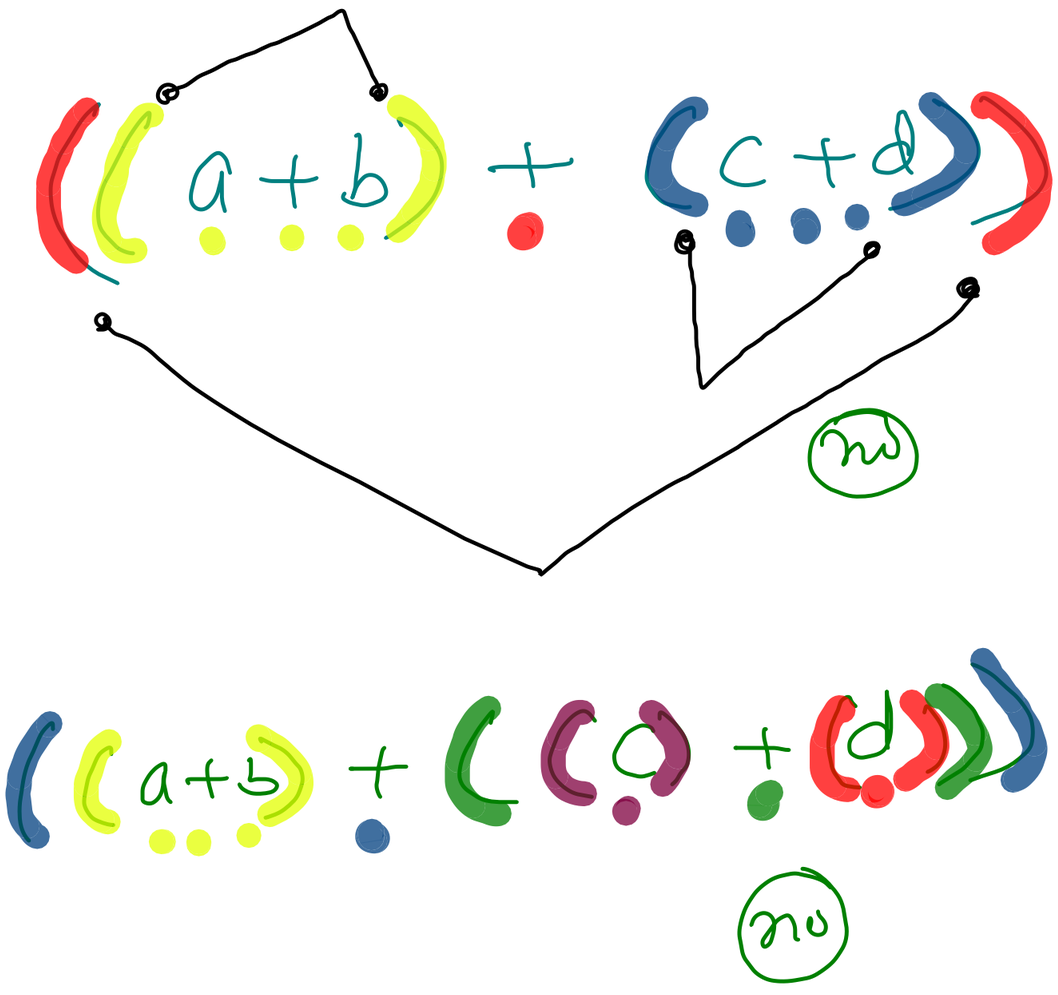


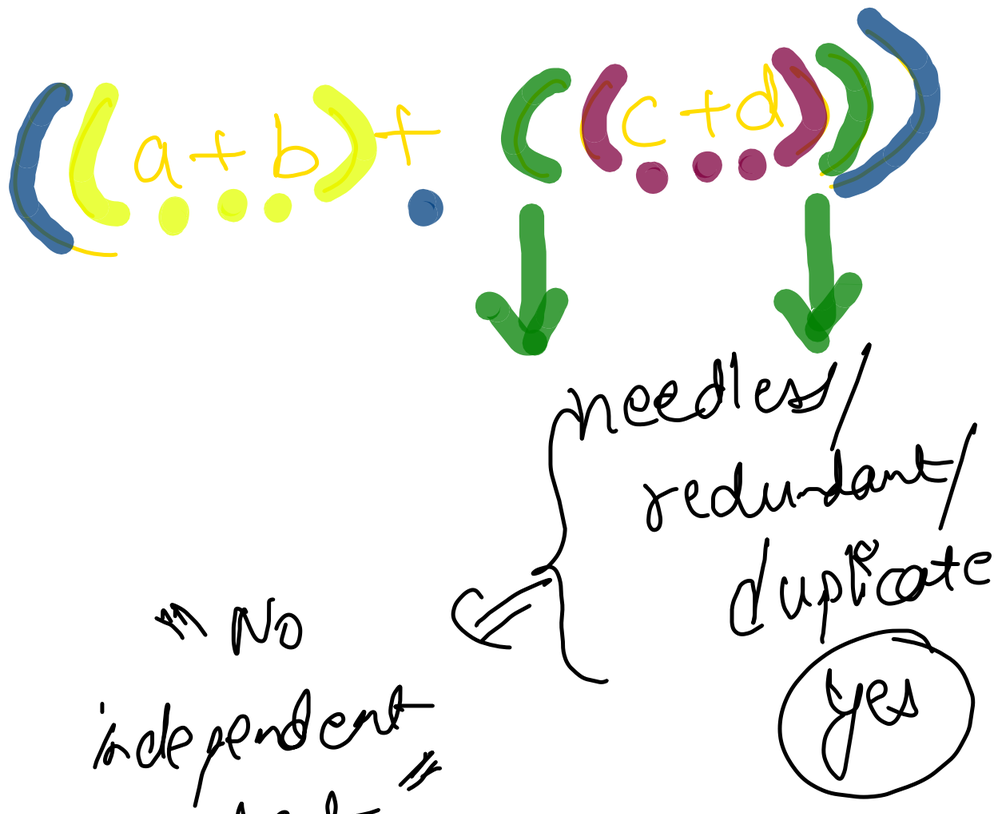


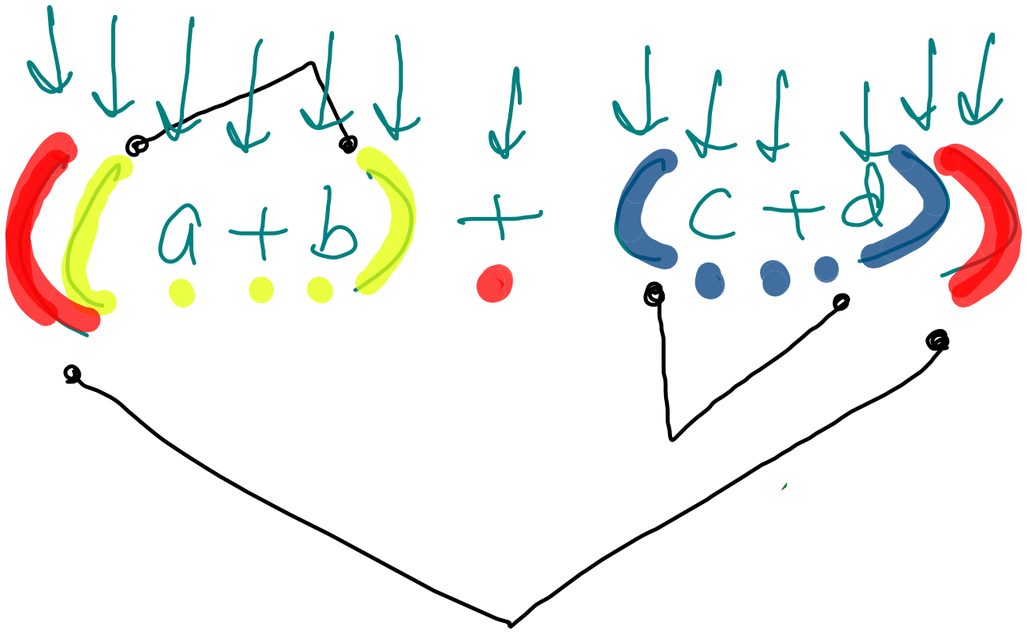


$$(a+[b+c)+d]$$

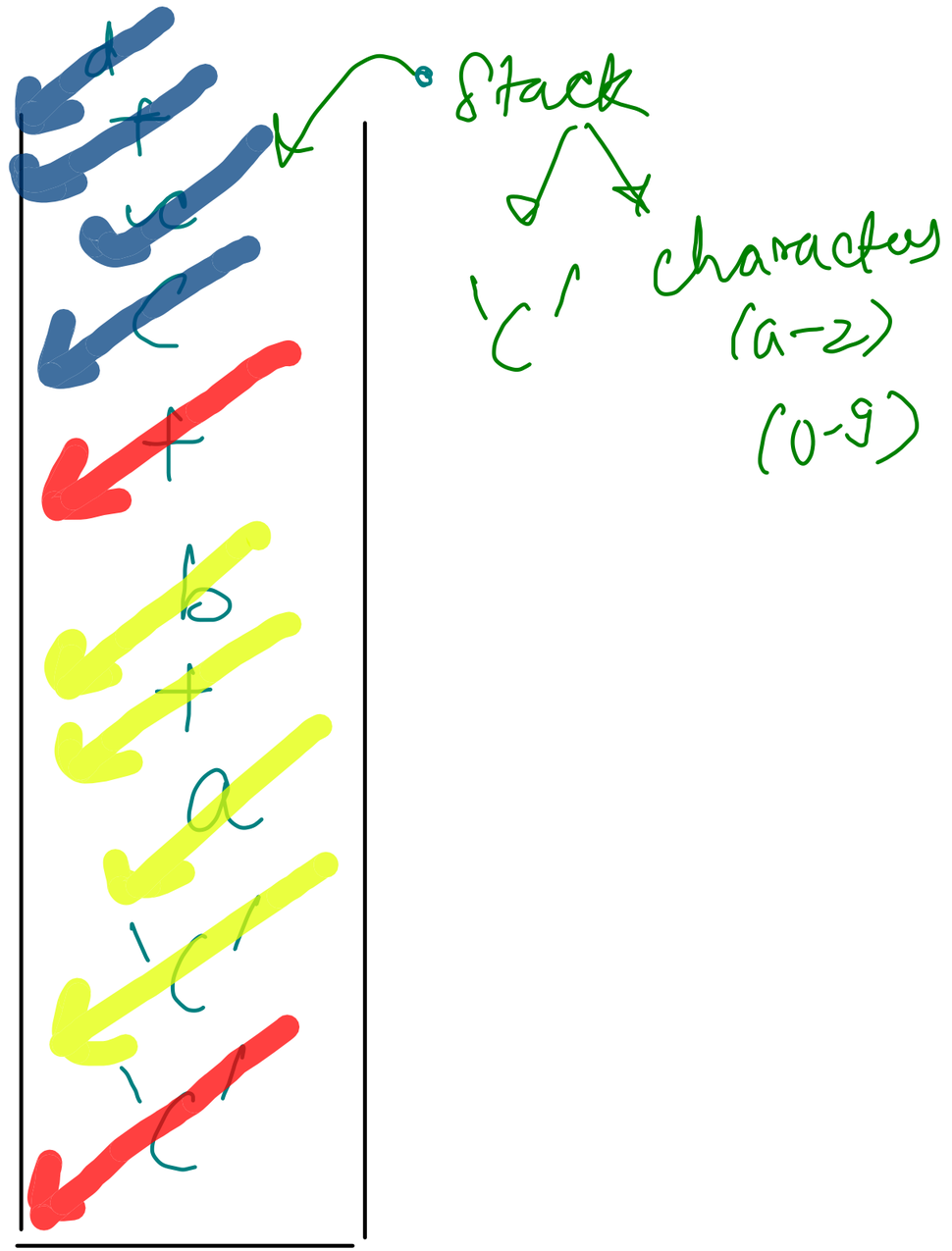
Redundant / Duplicate / Needless / Extra Brackets







no redundant braces

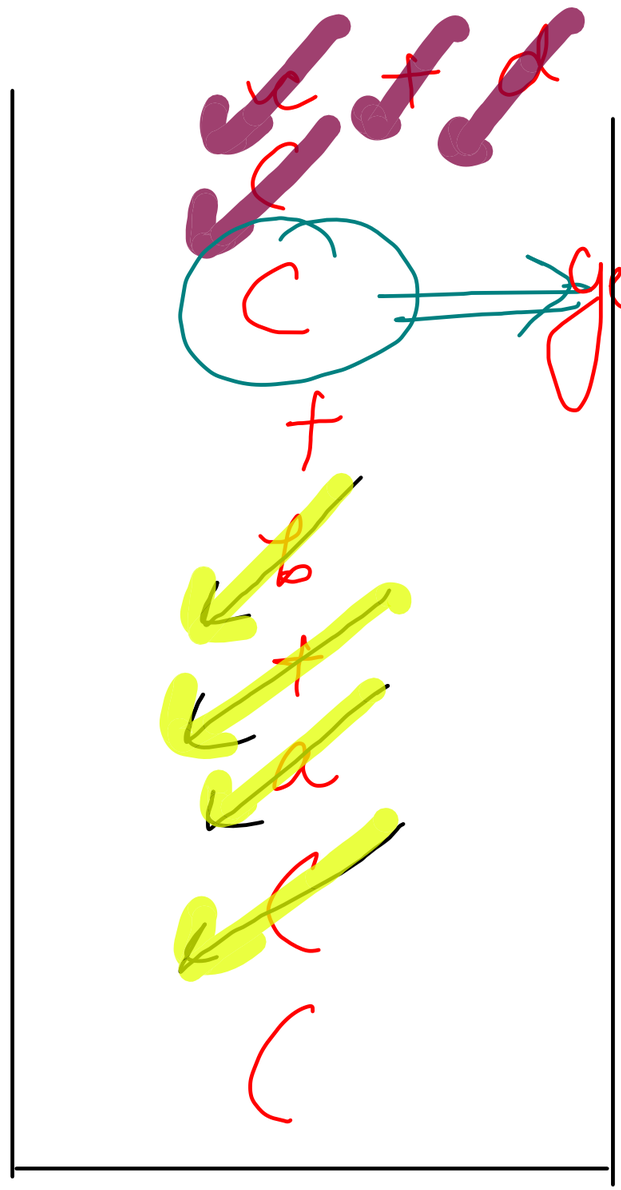


((a+b)+(c+d))

needles/ redundant/ duplicate

yes

⇒ No independent content ⇐



O(N) Time, O(N) auxiliary space (Stack)

```java
Stack<Character> stk = new Stack<>();
for(int i=0; i<str.length(); i++){
    char ch = str.charAt(i);

    if(ch == ')'){
        if(stk.peek() == '('){
            // No Indepedent braces
            System.out.println("true");
            return;
        }

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

        stk.pop(); // '(' is balanced, hence pop

    } else stk.push(ch);
}

System.out.println("false");
```

lecture (4) { Wednesday }

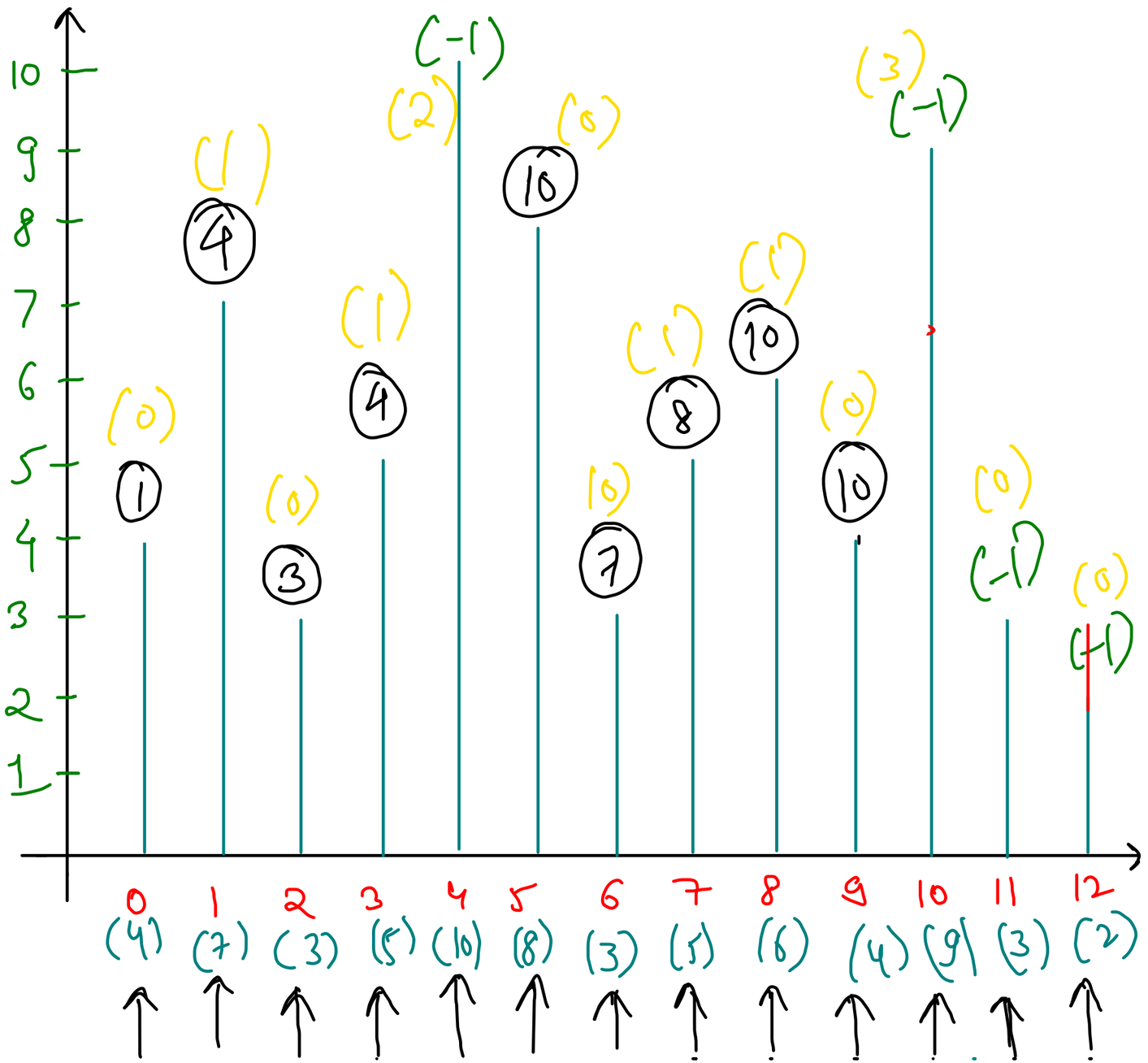
- Next Greater to Right

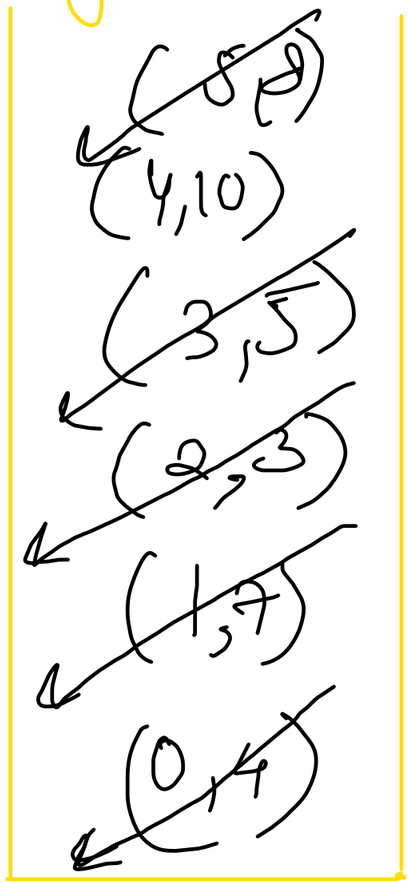
HW
- NGE to Left
- NSE to Right
- NSE to Left

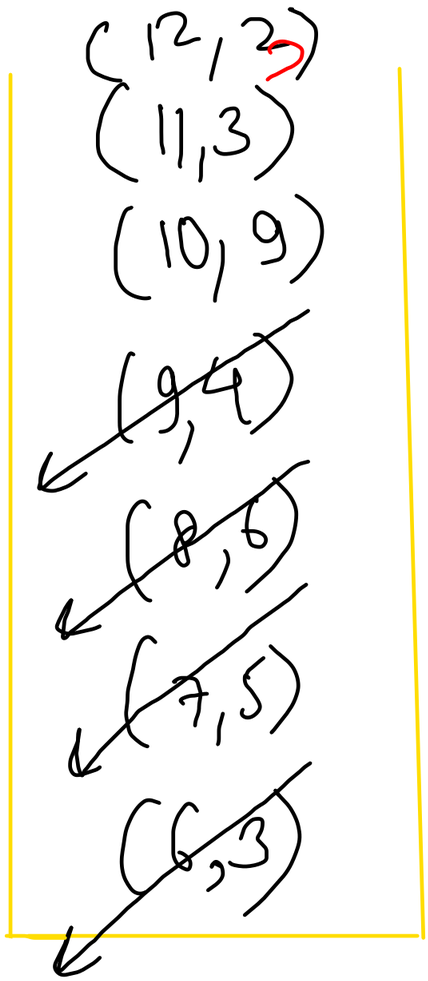
- Stock Span
- largest Histogram
- Sliding window Maximum

Interview Specific Problems

Next Greater Element to the right

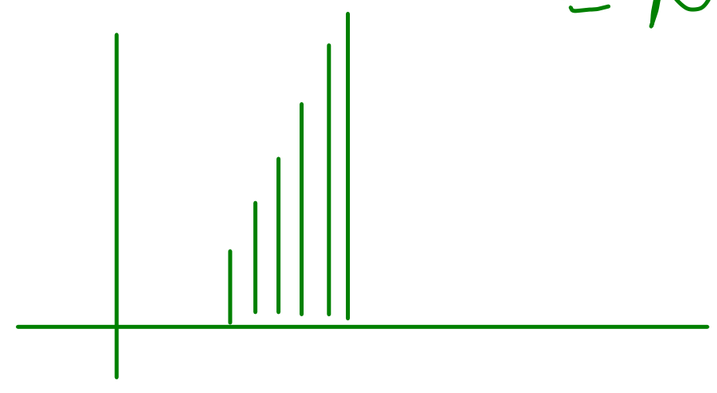






→ for those elements whose nge doesn't exist

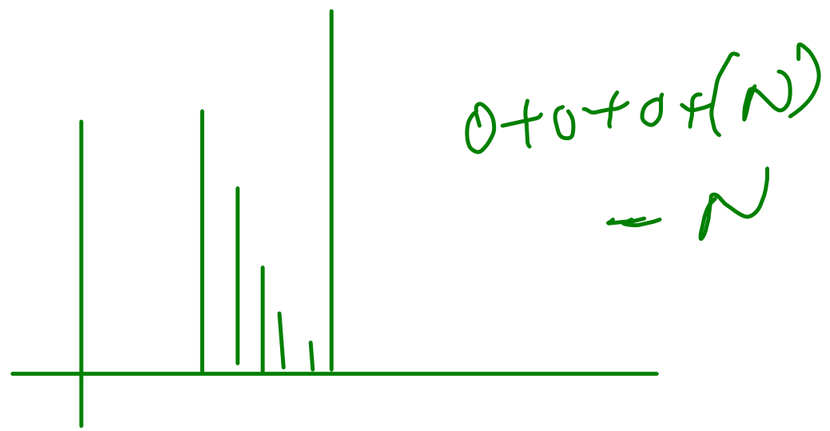
1+1+1+1+1+1+1
= N



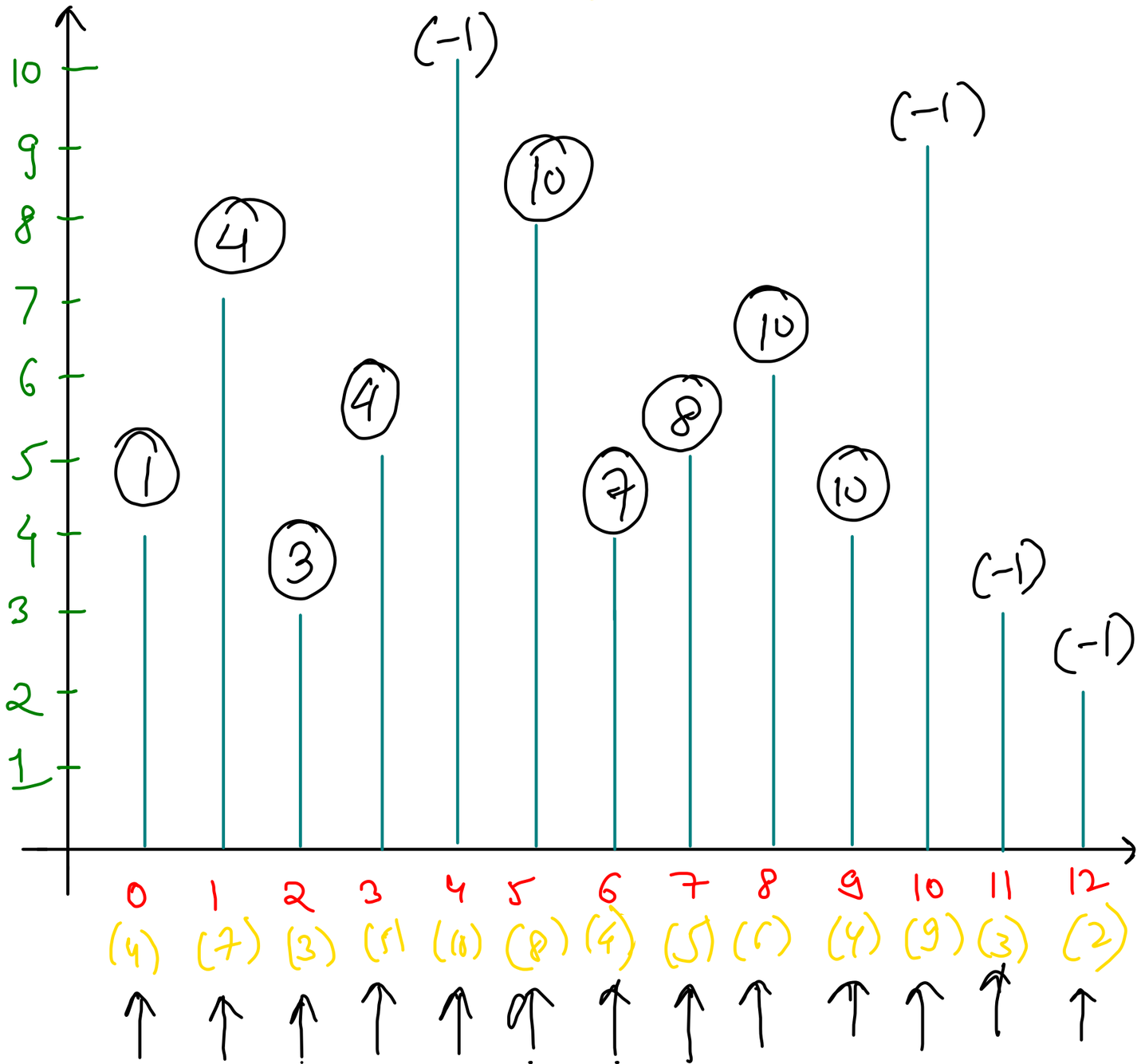
O(N) Time Comp

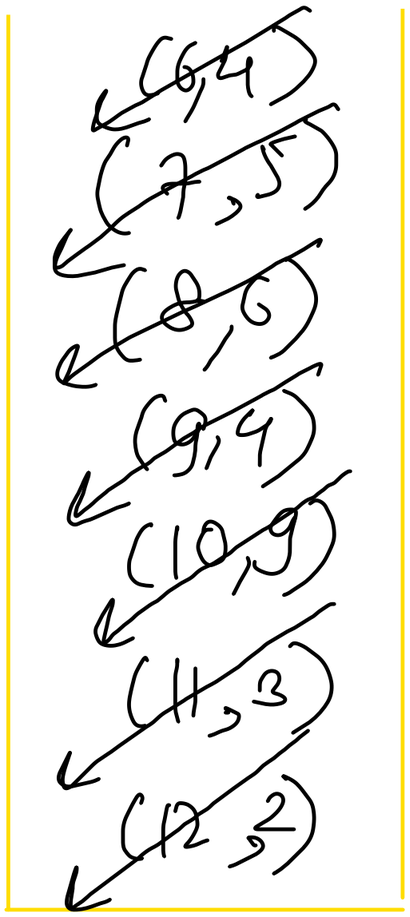
O(N) auxiliary space { stack }

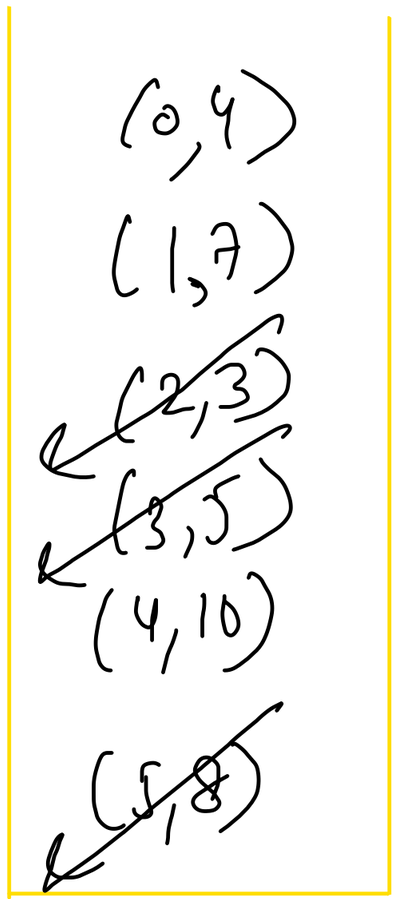
O(N) output space { nge array }

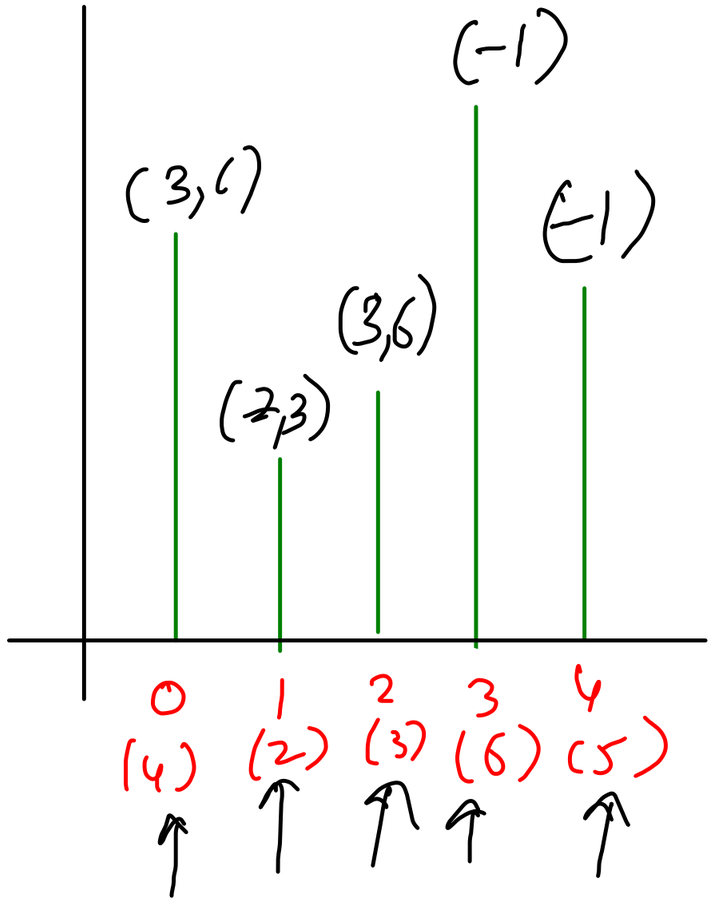


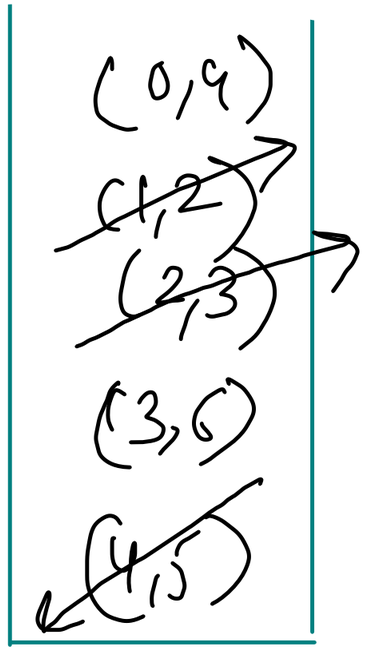
Next greater to the right (2nd approach)







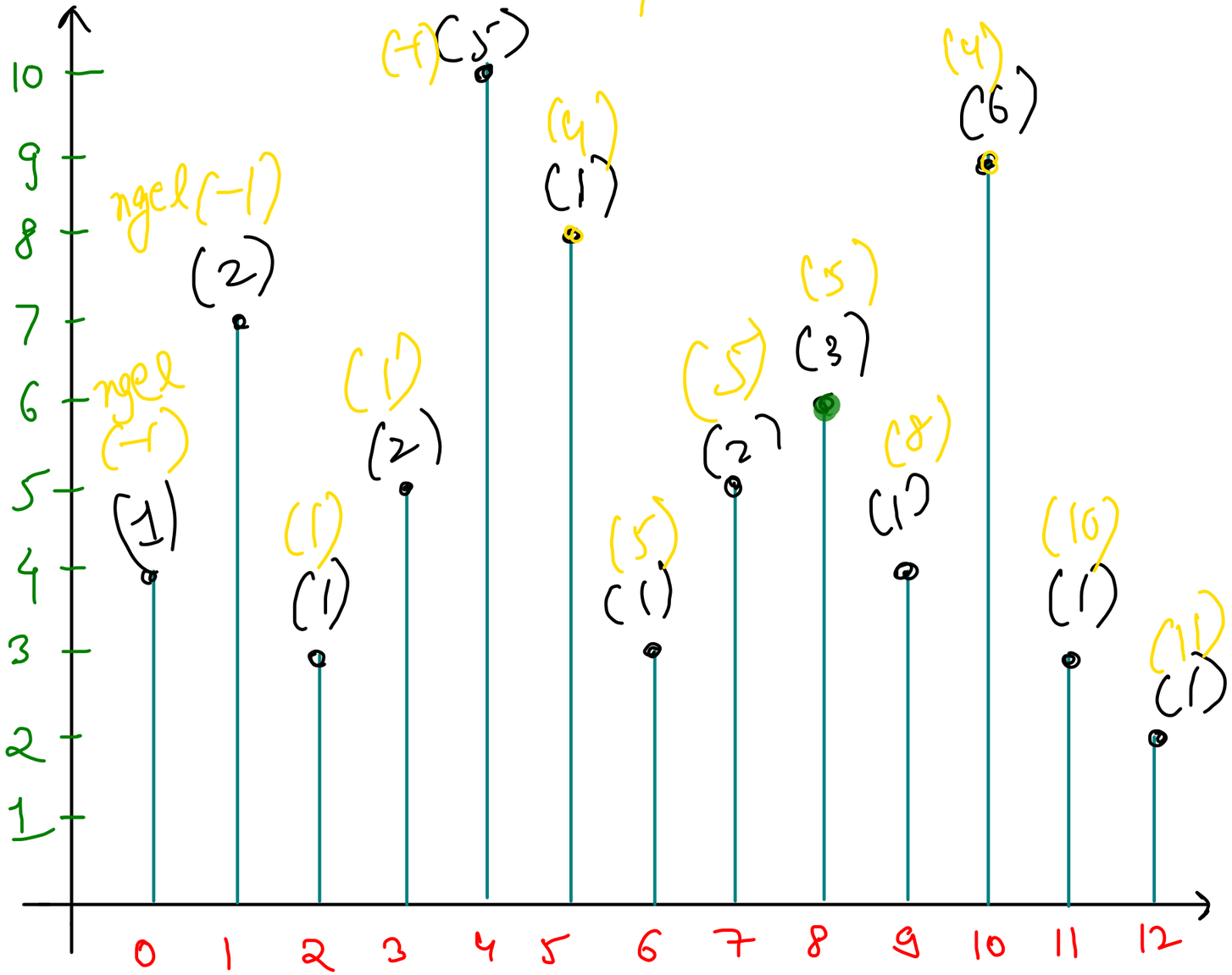




O(N)

```java
int[] nge = new int[arr.length];
Arrays.fill(nge, -1);

Stack<Integer> stk = new Stack<>();
for (int i = arr.length - 1; i >= 0; i--) {
      // pop smaller or equal elements to the right (in the stack)
      while (stk.size() > 0 && arr[i] >= arr[stk.peek()]) {
        stk.pop();
      }

      if(stk.size() > 0)
        nge[i] = arr[stk.peek()];

      // push ourself to find our nge to the right
      stk.push(i);
}
return nge;
```

Stock Span



nge

1. nge to the right / left
2. nge[i] = index / value

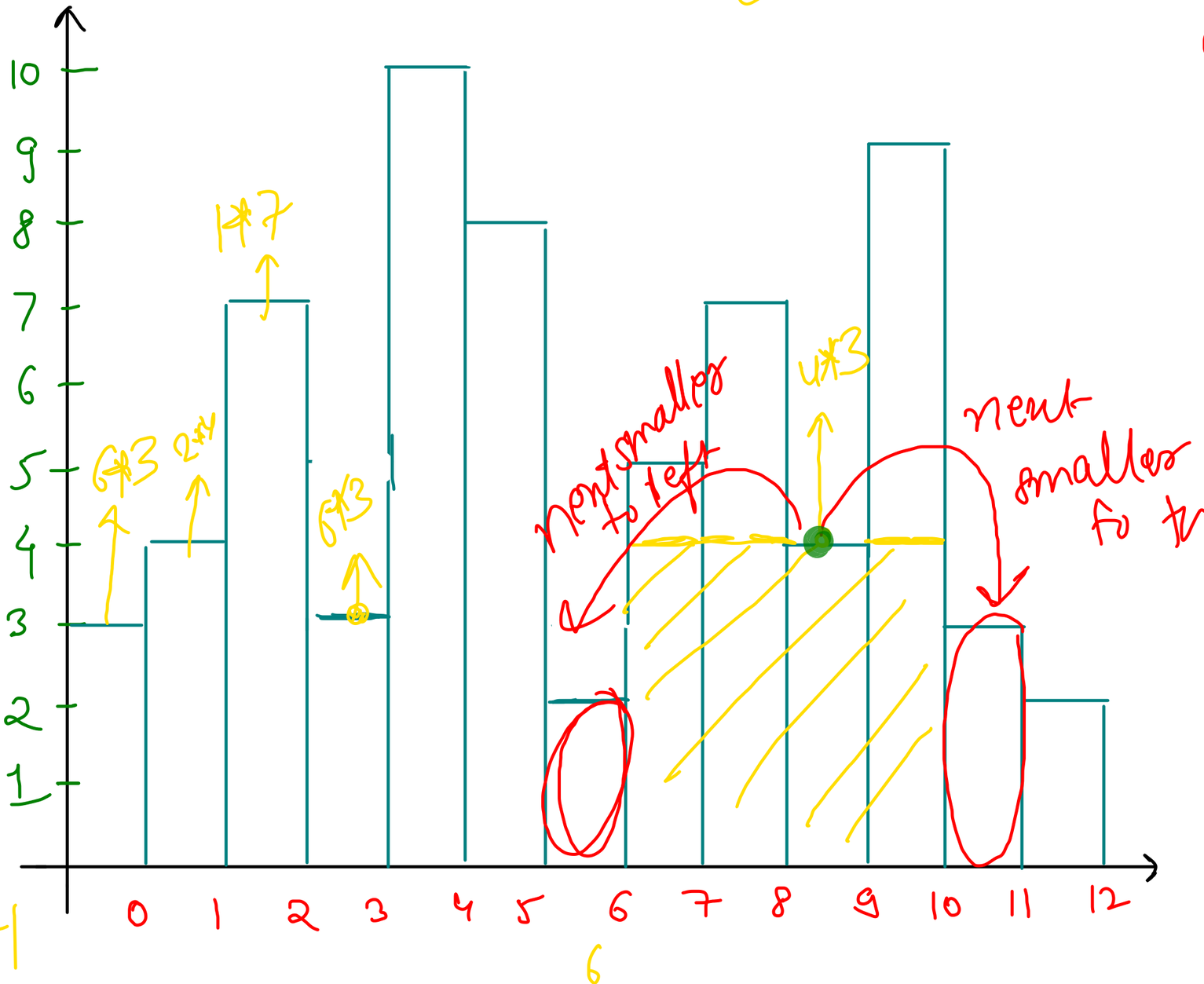
stock span[i] = diff b/w i & 2nd nge[i]'s index

```java
public static int[] solve(int[] arr){
    int[] stock = new int[arr.length];

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

          if(stk.size() > 0)
            stock[i] = (i - stk.peek());
          else stock[i] = (i + 1);

          stk.push(i);
    }
    return stock;
}
```

O(N) Time Complexity

O(N) extra space {stack}

Max Area Histogram



$$area = \{ nser[i] - nsel[i] - 1 \} * arr[i]$$

length

height

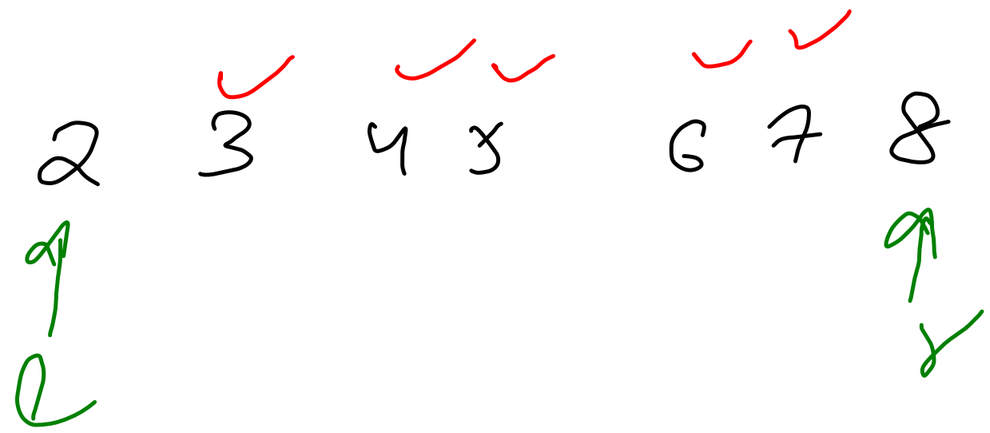
```java
public static int[] nsr(int[] arr) {
  int[] nse = new int[arr.length];
  Arrays.fill(nse, arr.length);

  Stack<Integer> stk = new Stack<>();
  for (int i = arr.length - 1; i >= 0; i--) {
        while (stk.size() > 0 && arr[i] <= arr[stk.peek()]) {
          stk.pop();
        }

        if(stk.size() > 0)
          nse[i] = stk.peek();

        stk.push(i);
  }
  return nse;
}
```

```java
public static int[] nsl(int[] arr){
  int[] nse = new int[arr.length];
  Arrays.fill(nse, -1);

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

        if(stk.size() > 0)
          nse[i] = stk.peek();

        stk.push(i);
  }
  return nse;
}
```

```java
// code
int[] nsl = nsl(arr);→O(N)
int[] nsr = nsr(arr);→O(N)

int maxArea = 0;
for(int i=0; i<arr.length; i++){→O(N)
    int currArea = (nsr[i] - nsl[i] - 1) * arr[i];
    maxArea = Math.max(maxArea, currArea);
}

System.out.println(maxArea);
```

O(N) time Complexity

O(N) extra space of Stack, nsl, nsr



$[2,8] \Rightarrow 8-2+1 \Rightarrow r-l+1$

$(2,8] \Rightarrow 8-2 \Rightarrow r-l$

$[2,8) \Rightarrow 8-2 \Rightarrow r-l$

$(2,8) \Rightarrow 8-2-1 \Rightarrow r-l-1$

Today's Questions

- largest area Histogram
- sliding window Maximum

- Celebrity Problem
- Minimum Stack
  - → with Extra Space
  - → w/o Extra Space
- Smallest No following Pattern

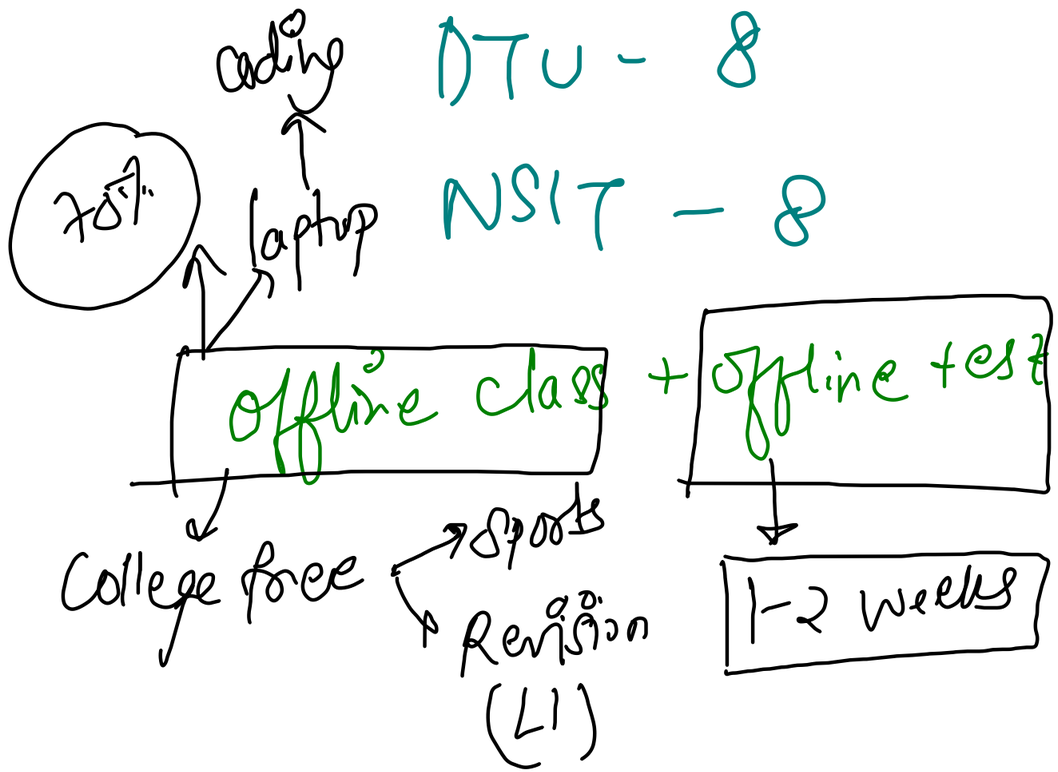
Offline

MSIT - 14

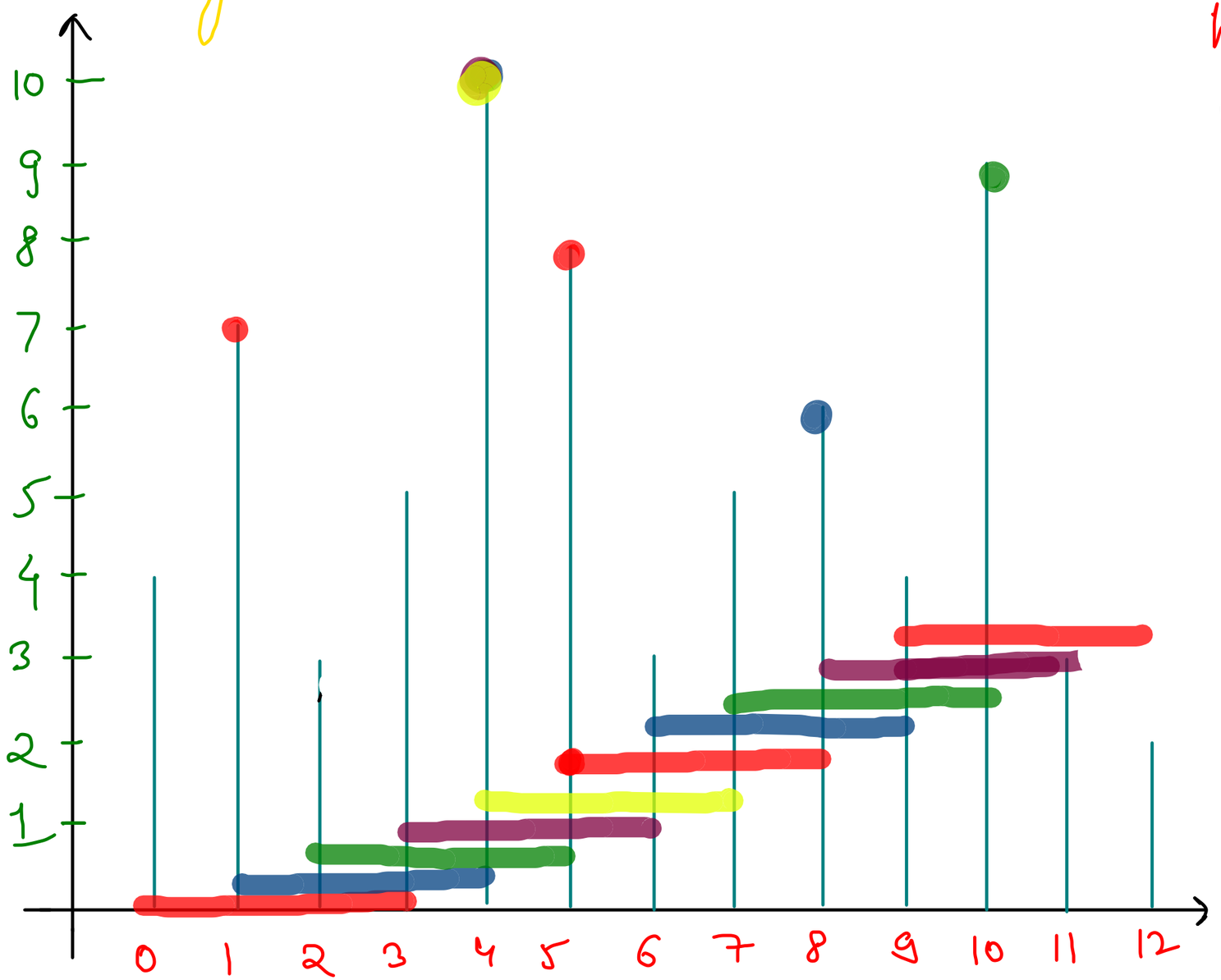
MAIT - 21

DTU - 8

NSIT - 8



Sliding Window Maximum



window = 4

[0,3] ⇒ 7

[1,4] ⇒ 10

[2,5] ⇒ 10

[3,6] ⇒ 10

[4,7] ⇒ 10

[5,8] ⇒ 8

[6,9] ⇒ 6

[7,10] ⇒ 9

[8,11] ⇒ 9

[9,12] ⇒ 9

(1) Bruteforce
O(N*k)
Nested loops

(2) using Doubly Ended Queue (Deque)

(3) Using Stack (NGE)

addFirst()

O(1)

addLast() O(1)

removeFirst()

O(1)

removeLast()

O(1)

Doubly Ended (Deque)

{Doubly linked list}

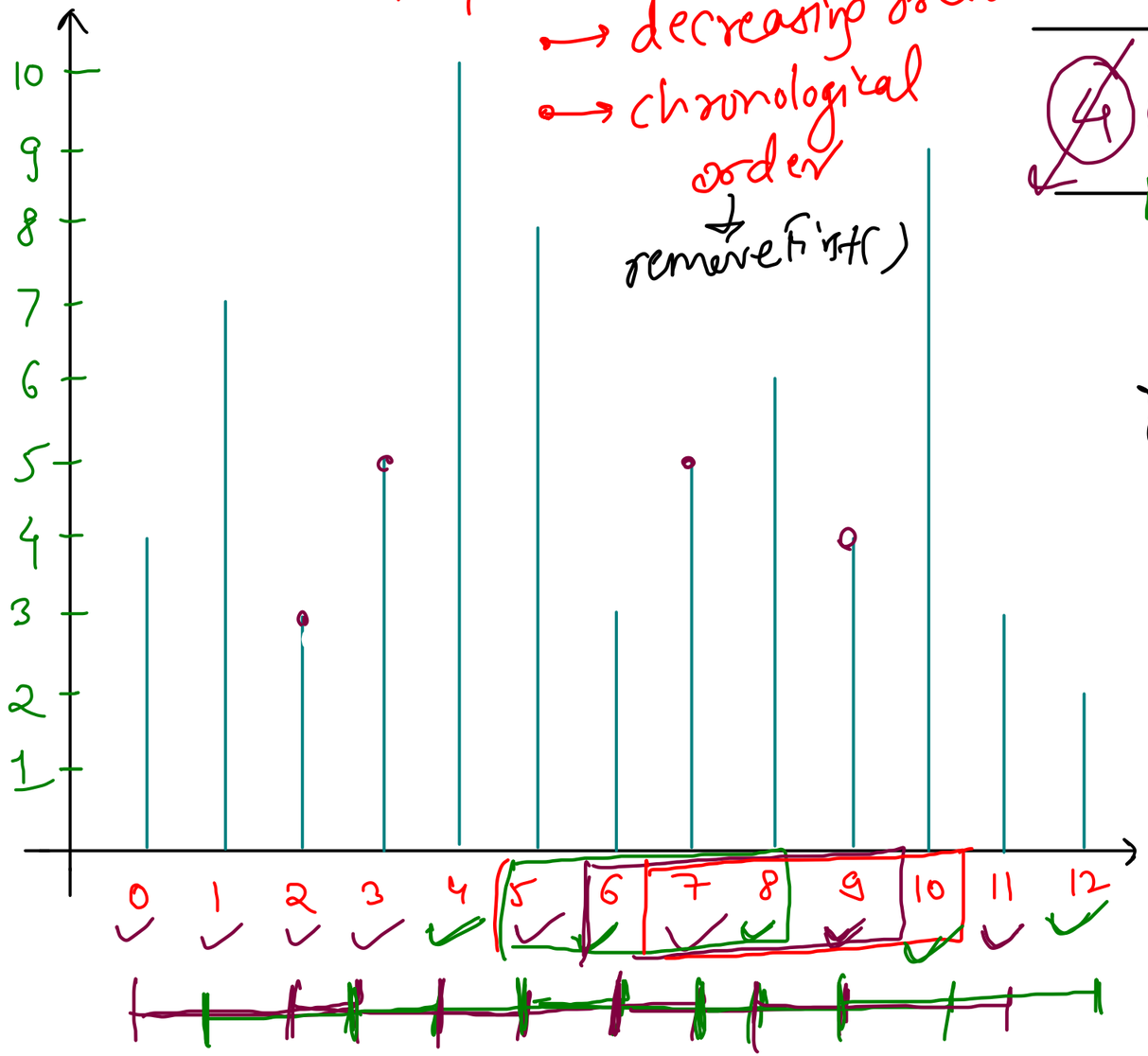
Queue q = new ArrayDeque<>();

Deque dq = new ArrayDeque<>();

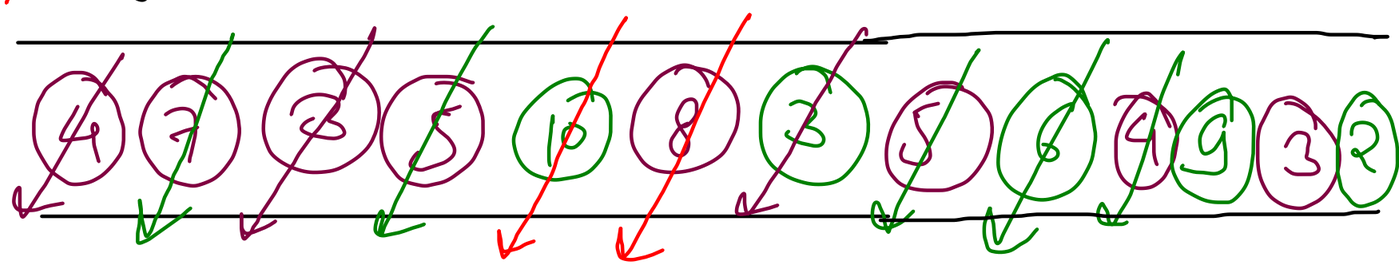
addFirst

removeLast

win =



Deque is in
→ decreasing order → getFirst() - maxm
→ chronological order
↓
removeFirst()



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

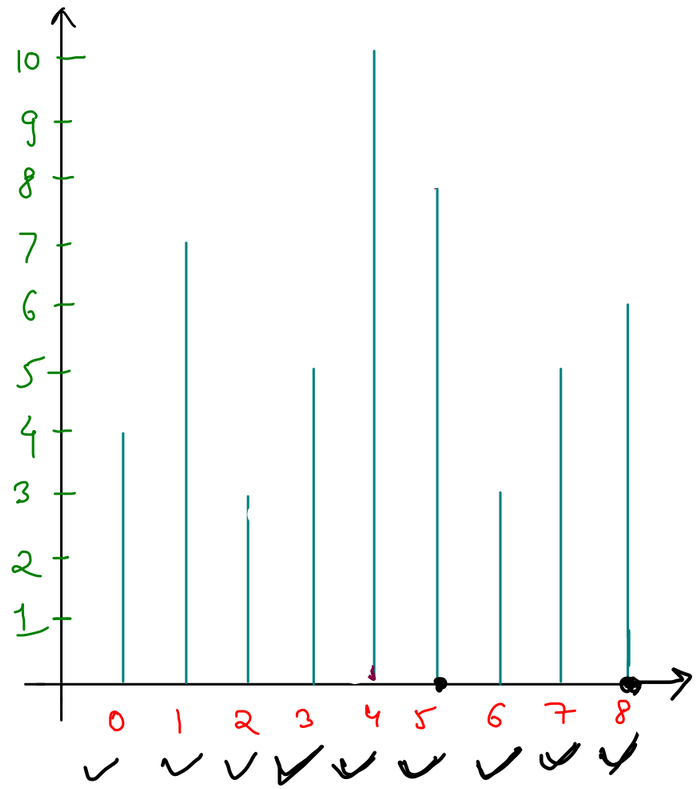
getFirst → maximum of current window

① removeFirst → which got excluded from the window

① removeLast → Smaller than ourself (can't be max of any window)

② addLast → we may the max of upcoming windows

```java
public int[] maxSlidingWindow(int[] nums, int k) {
    int[] res = new int[nums.length - k + 1];
    int idx = 0;

    Deque<Integer> q = new ArrayDeque<>();

    for(int i=0; i<nums.length; i++){

        // removeFirst -> when front element is excluded from current window
        if(q.size() > 0 && q.getFirst() <= i - k){
            q.removeFirst();
        }

        // removeLast -> smaller than our current element
        while(q.size() > 0 && nums[q.getLast()] < nums[i]){
            q.removeLast();
        }

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

        if(i >= k - 1){
            // current window's maximum is at front of deque
            res[idx++] = nums[q.getFirst()];
        }
    }

    return res;
}
```

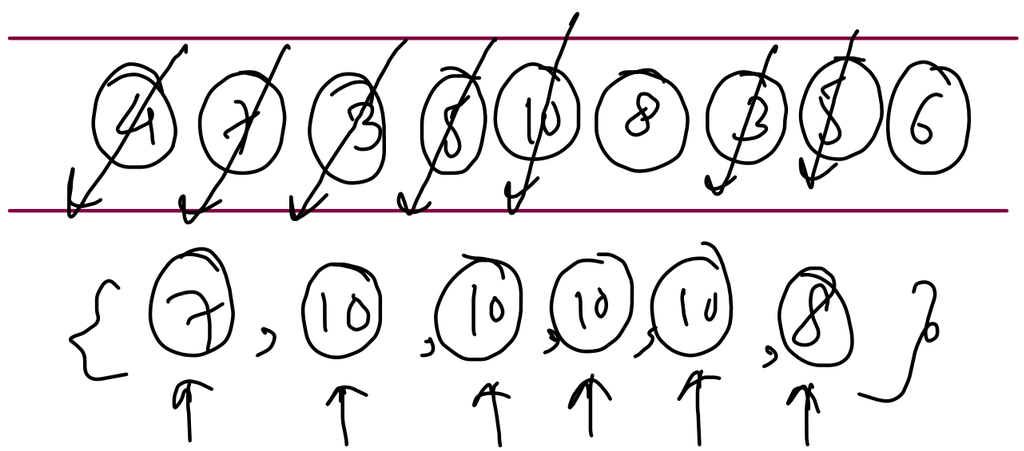


win = k = 4

O(N) time

O(k) space

Deque



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

(1) No celebrity (T/F)

(2) More than 1 celebrity (T/F)

↳ Assumption

b.n are celebrity

Boolean matrix

mat[i][j]

→ 1 ith person knows jth person

→ 0 ith person doesn't know jth person

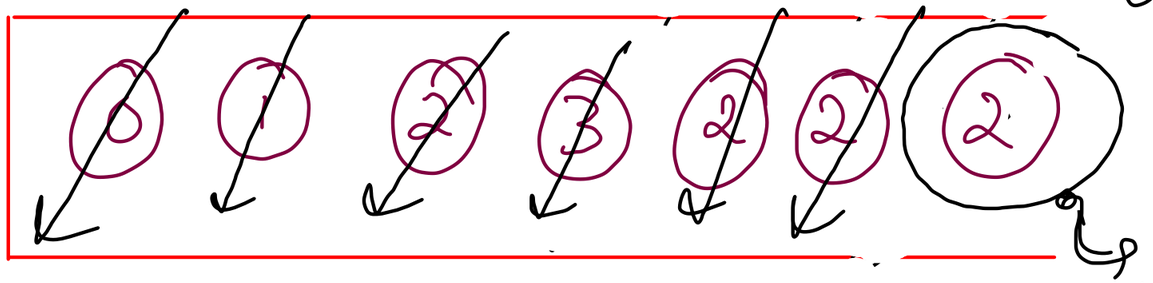
Celebrity → Who knows nobody

All knows him/her

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

(2 pop, 1 push)

$O(n)$



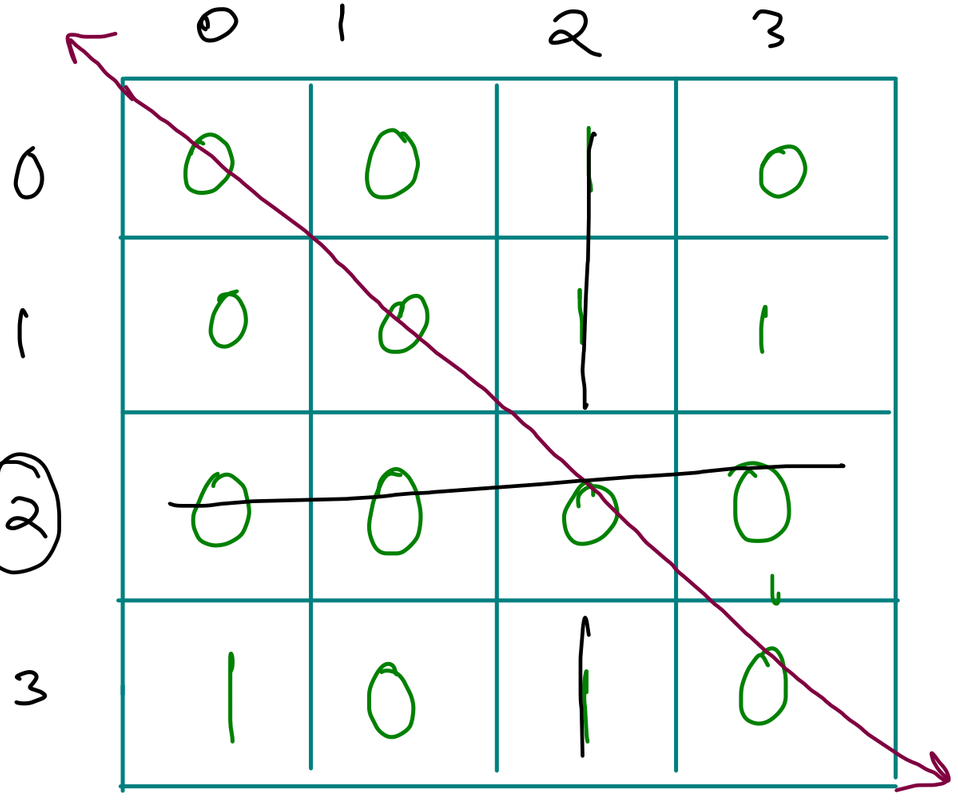
col == 1 && row == 0

confirm cand

stk.size() == 1

→ Potential celebrity

Possible Celebrities



{ x , y }

if mat[x][y] = 1 → x knows y

⇒ x can't be celebrity

else mat[x][y] = 0 → x doesn't know y

⇒ y can't be a celebrity

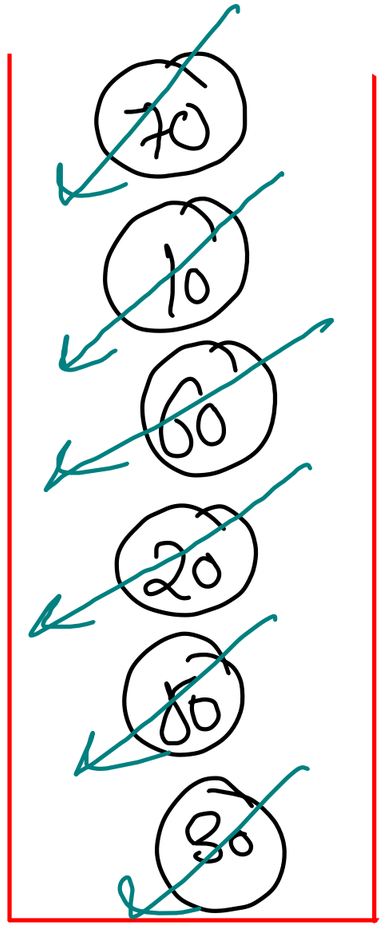
O(N) time, O(N) extra space (stack)

```java
Stack<Integer> potentialCeleb = new Stack<>();
for(int i=0; i<arr.length; i++)
    potentialCeleb.push(i);

while(potentialCeleb.size() > 1){
    int y = potentialCeleb.pop();
    int x = potentialCeleb.pop();

    if(arr[x][y] == 1){
        // x knows y -> x can't be a celebrity
        potentialCeleb.push(y); // y may or may not be a celebrity
    } else {
        // x does not know y -> y can't be a celebrity
        potentialCeleb.push(x); // x may or may not be a celebrity
    }
}
```

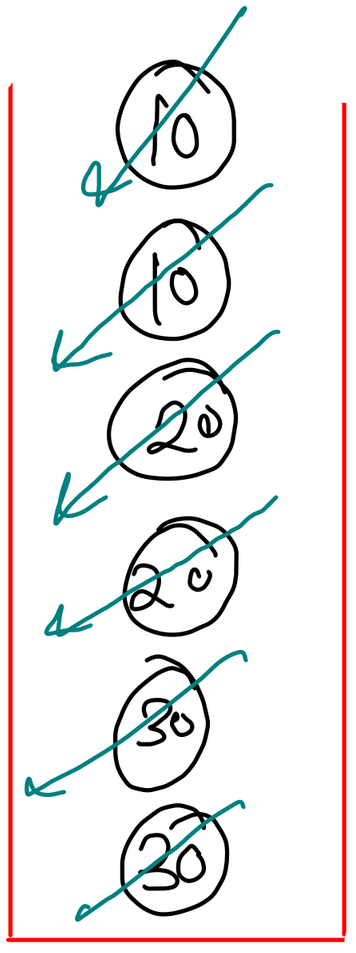
```java
// Confirmation
int x = potentialCeleb.pop();

// Its row should be 0 (X should know nobody)
for(int j=0; j<arr.length; j++){
    if(j == x) continue; ← mat(i)(i)
    if(arr[x][j] == 1) {
        System.out.println("none"); // No celebrity
        return;
    }
}

// Its column should be 1 (Everybody knows X)
for(int i=0; i<arr.length; i++){
    if(i == x) continue; ← mat(i)(i)
    if(arr[i][x] == 0){
        System.out.println("none");
        return;
    }
}

System.out.println(x);
```

Minimum Stack { with extra space }

	peek() O(1)	min() O(1)
push(30) O(1)	30	30
push(50)	50	30
push(20)	20	20
push(60)	60	20
push(10)	10	10
push(70)	70	10
pop() O(1)		



val



min

```java
public static class MinStack {
  Stack<Integer> allData;
  Stack<Integer> minData;

  public MinStack() {
    allData = new Stack<>();
    minData = new Stack<>();
    minData.push(Integer.MAX_VALUE);
  }


  int size() {
      return allData.size();
  }

  void push(int val) {
      allData.push(val);
      minData.push(Math.min(minData.peek(), val));
  }

  int pop() {
      if(size() == 0){
          return -1; // underflow
      }

      minData.pop();
      return allData.pop();
  }
```

O(1)

O(1)

```java
  int top() {
      if(size() == 0){
          return -1; // underflow
      }

      return allData.peek();
  }

  int min(){
      if(size() == 0){
          return -1; // underflow
      }

      return minData.peek();
  }
```

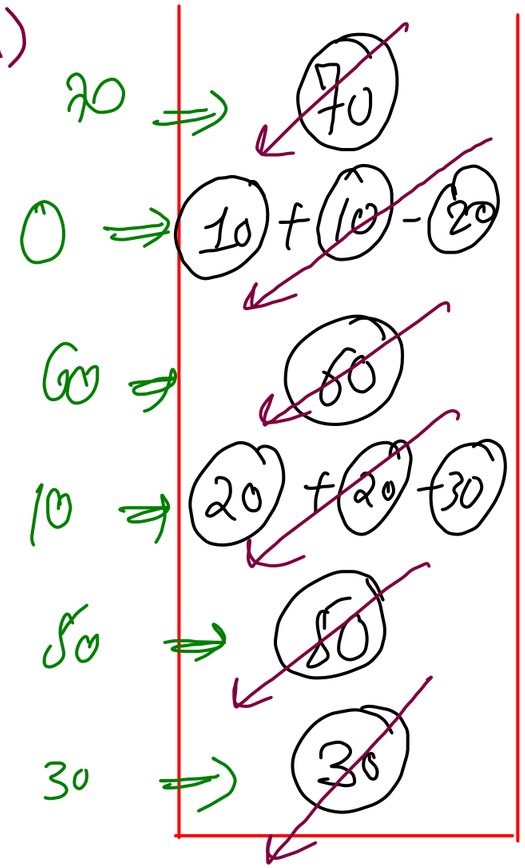
O(1)

O(1)

{without extra space}

	peek()	min()
push(30)	30	30
push(50)	50	30
push(20)	20	20
push(60)	60	20
push(10)	10	10
push(70)	70	10

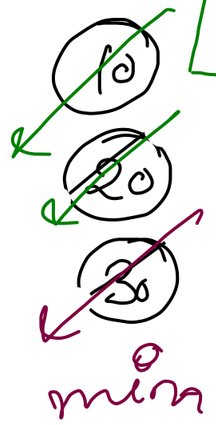
pop() O(1)



Decryption

top = min + min - prev

prev = 2*min - top



encryption (Y/N)

Encrypt ⇒ minimum node

val + (val - min)

20 + (20 - 30)

20 + -ve

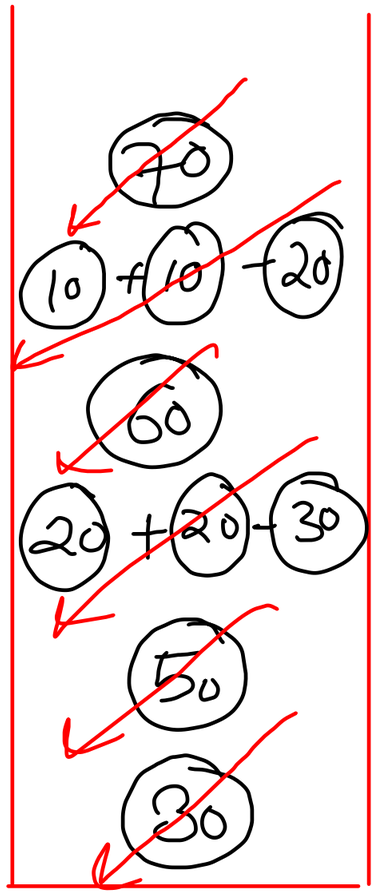
< 20

0 in integer Range

without extra space

	peek() O(1)	min() O(1)
push(30) O(1)	30	30
push(50)	50	30
push(20)	20	20
push(60)	60	20
push(10)	10	10
push(70)	70	10

pop() O(1)



encrypt

val & min

val + (val - min)

-ve

decrypt

prev

= 2 * min - top

+∞

min

O(1) extra space

```java
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 if(val < min){
        stk.push(2l * val - min);
        min = (long)val;
    } else {
        stk.push(1l * val);
    }
}
```

O(1)

2l => 2 in long

2l = (long)2

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

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

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

    // check if top is encrypted value
    if(stk.peek() < min) return (int)min;
    long top = stk.peek();
    return (int)top;
}

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

O(1)

O(1)

O(1)