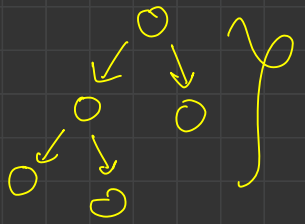
Stacks

✱ linear data structure

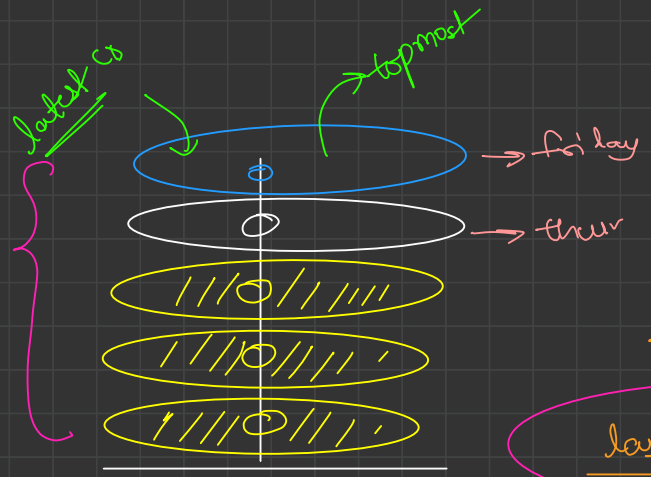
- → Arrays
- → Arraylist
- → LinkedList
- → Queues
- → Stacks

✱ Non-Linear data Structures

- → HashMap | HashSet
- → trees, tries, heaps
- → graph



Stacks

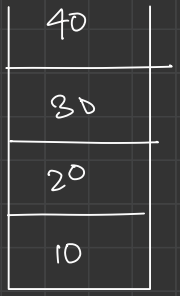


{LIFO}

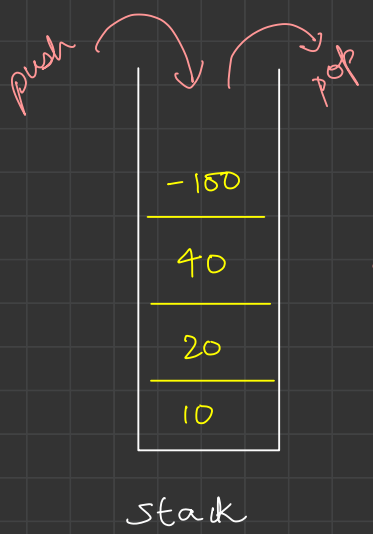
last in, first out

Stack

data: 10, 20, 30, 40



off 40, 30, 20, 10



Stack {LIFO}

- → push(int v) : Add value v to topmost of stack
- → pop() : Remove the topmost Element
- → peek() : you are able to view topmost Element
- → size() : Size of the Stacks

TC : O(1) → for each operation

```
class Stack
{
    ArrayList<Integer> list;
    int size;

    Stack()
    {
        list = new ArrayList();
        size = 0;
    }

    void push(int v)        TC: O(1)
    {
        list.add(v);
        size++;
    }
```

```
    int pop()               TC: O(1)
    {
        if(size > 0)
        {
            int ele = list.remove(size-1);
            size--;
            return ele;
        }
        else
        {
            System.out.println("Stack underflow");
            return -1;
        }
    }

    int size()              TC: O(1)
    {
        return size;
    }
}
```

Practical Examples of Stack

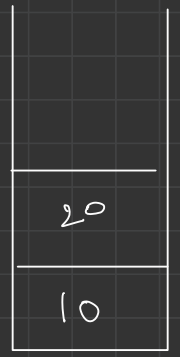
- Recursion
- Memory Managment
- Cache
- Browser History | Back functionality

Syntax

```
Stack<G> st = new Stack<>();
```

- `Stack` → Java Stack class
- `G` → Wrapper Classes / User defined classes
- `st` → reference variable

```
st.push(10);
st.push(20);
st.push(-100);
print(st.peek());   ~> -100
st.pop();
print(st.peek());   ~> 20
print(st.size());   ~> 2
```



Q Extra Brackets

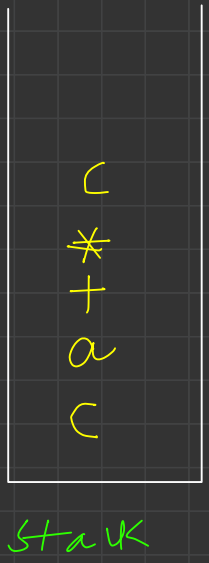
String str = "(a+b)"; → NO

= "((a+b))"; → yes

= "(a+b)+(c+d+(e*f)*(c))"; → yes

= "((a)+(b))"; → NO

str = "(a+(b*d+f-(m1+n-o)*(fj()))"



```java
public boolean ExtraBrackets(String exp) {
    // Write your code here

    Stack<Character> st = new Stack<>();

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

        if (ch != ')') {
            st.push(ch);
        } else {
            // find corresponding opening bracket

            if (st.peek() == '(') {
                // no exp in between
                return true;
            } else {
                // remove exp
                while (st.size() != 0 && st.peek() != '(') {
                    st.pop();
                }

                // as we have a exp in between, so this pair is not a extra bracket
                st.pop();
            }
        }
    }

    return false;
}
```

exp = " ((a+b) * (d*f))

Stack

TC: O(N)

SC: O(N)

Next Grater Element On Right

	0	1	2	3	4	5	6	7	8	9
arr[] = {	3,	6,	1,	2,	7,	3,	4,	1,	2,	5 }

nger[] = { 6, 7, 2, 7, -1, 4, 5, 2, 5, -1 }

Brute force

Nested loop

TC : O(N²)

SC : O(1)

TC : O(N) ?


```
// potential next greater elements on right
Stack<Long> st = new Stack<>();
long[] nger = new long[n];

for (int i = n - 1; i >= 0; i--) {
    // do you have people in stack
    // if there are people in stack, are they big enough to be mine next greater on right
    // if not remove them
    while (st.size() > 0 && st.peek() <= arr[i]) {
        st.pop();
    }

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

    // I can also be a potential nger for left people
    st.push(arr[i]);
}

return nger;
```

O(1)

O(N) ~ In life time

N times

once

n-1 n-2 ---------- 0

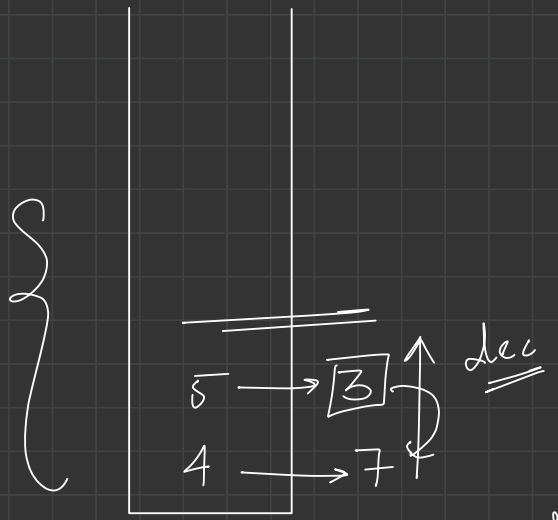
a b ---------- z

$\sum$ work = a + b + c + ---------- + z

= O(N)

TC : O(N)
SC : O(N)

Monotonic stack



Adv

1. you get index
2. you can compute nger, ngel → same iteration!

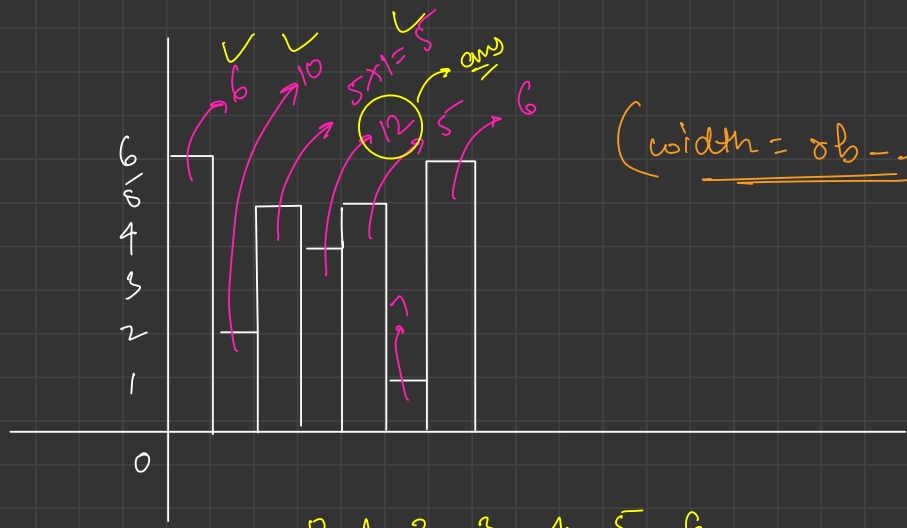
Stock Span Problem

0	1	2	3	4	5	6
100	80	60	70	60	75	85
1	1	1	2	1	4	6

ngeli { -1, 0, 1, 1, 3, 1, 0 }

↳ $\Theta(n^2)$

largest Area Histogram.



(width = rb − lb − 1)

heights [] = { 0 6, 1 2, 2 5, 3 4, 4 5, 5 1, 6 6 }

nseli [] = { -1, -1, 1, 1, 3, -1, 5 }

nseri [] = { 1, 5, 3, 5, 5, 7, 7 }

1 − (−1) − 1 = 1 6×1 = 6

5 − (−1) − 1 = 5 5×2 = 10

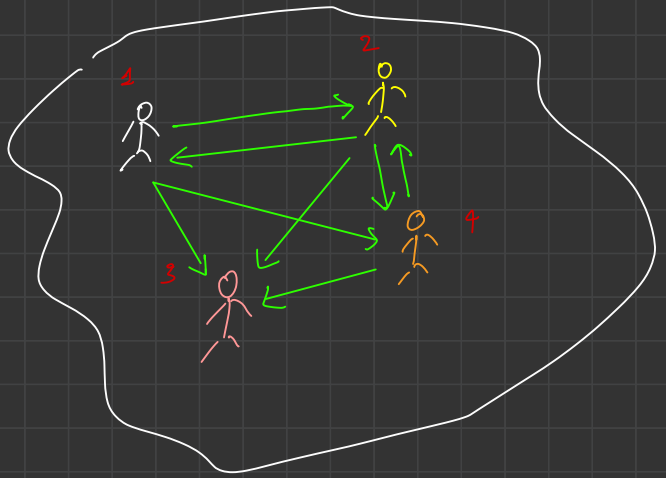
3 − 1 − 1 = 1 5×1 = 5

Steps

1. get next smaller on left index
2. get next smaller on right index
3. compute width = rb - lb - 1
   compute area = width * height
4. max area is your ans

```java
public static long maximumArea(long hist[], long n) {
    //Your code here

    // get nseli
    int[] nseli = getNSELI(hist, (int) n);

    // get nseri
    int[] nseri = getNSERI(hist, (int) n);

    // get max area
    long maxArea = 0;
    for (int i = 0; i < (int) n; i++) {
        long h = hist[i];
        int w = nseri[i] - nseli[i] - 1;
        long area = h * w;

        maxArea = Math.max(maxArea, area);
    }

    return maxArea;
}
```

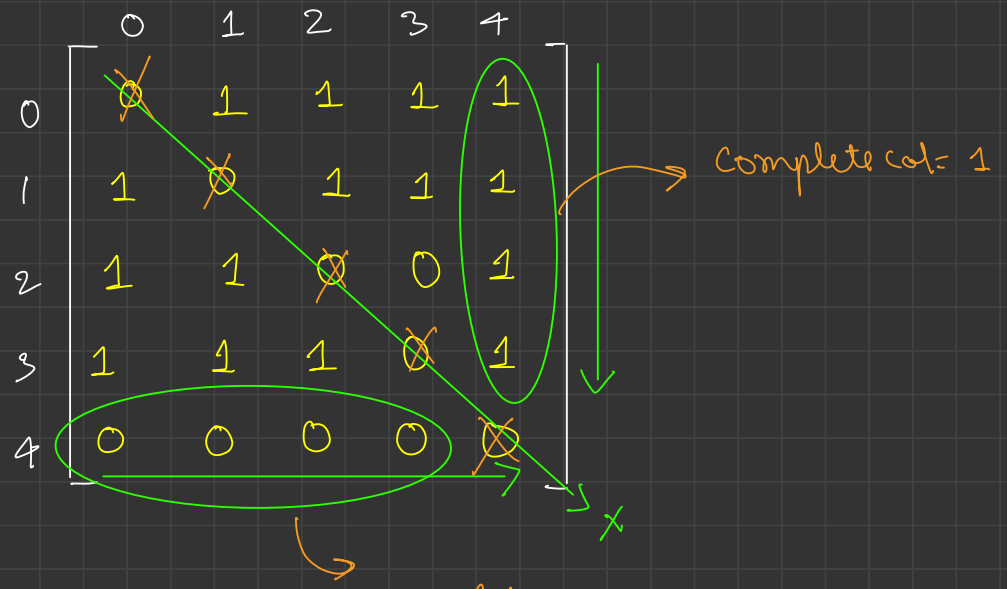
Celebrity Problem

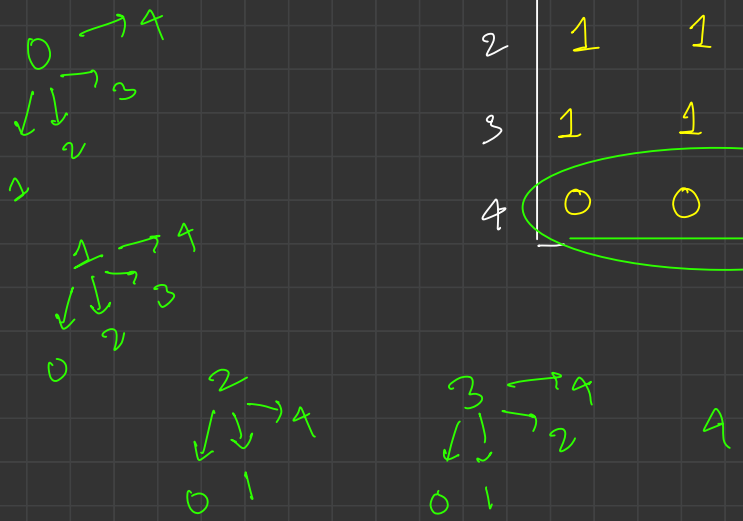


person		knows
1	→	2, 3, 4
2	→	1, 3, 4
3	→	
4	→	2, 3

Celebrity is a person who is known by everyone and he doesn't knows anyone!

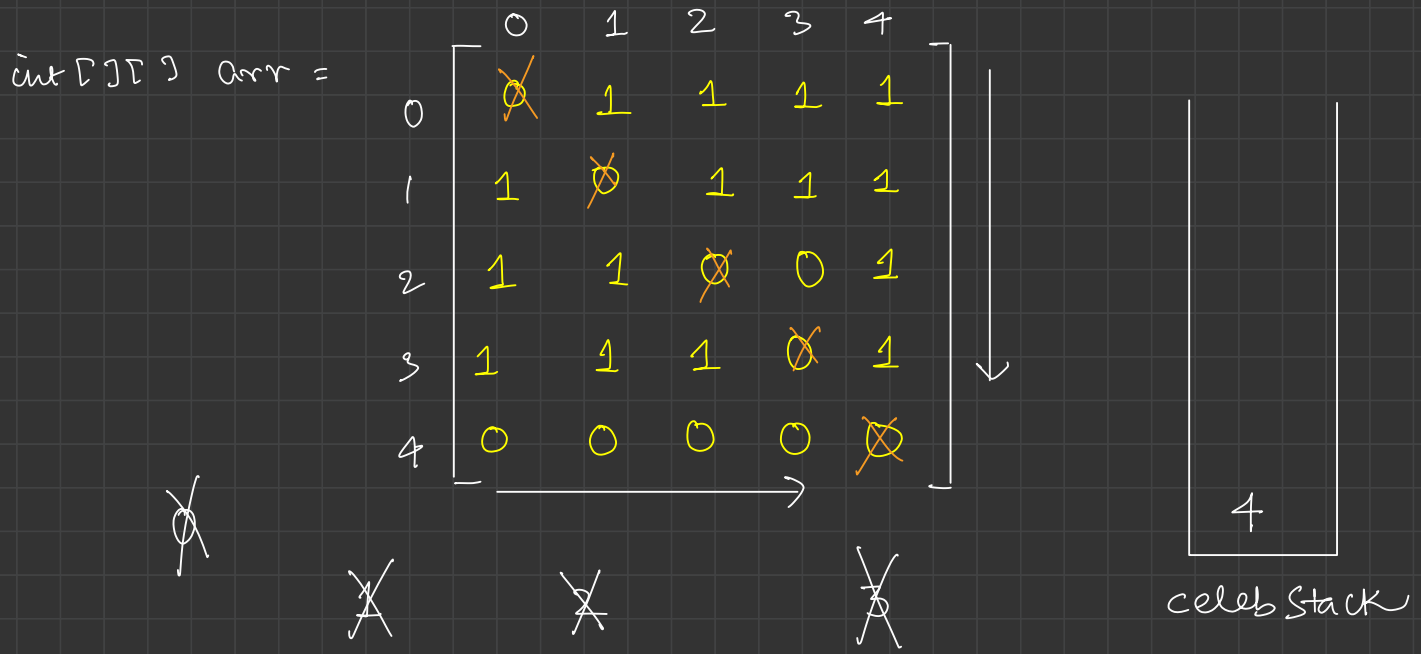
int[][] arr =





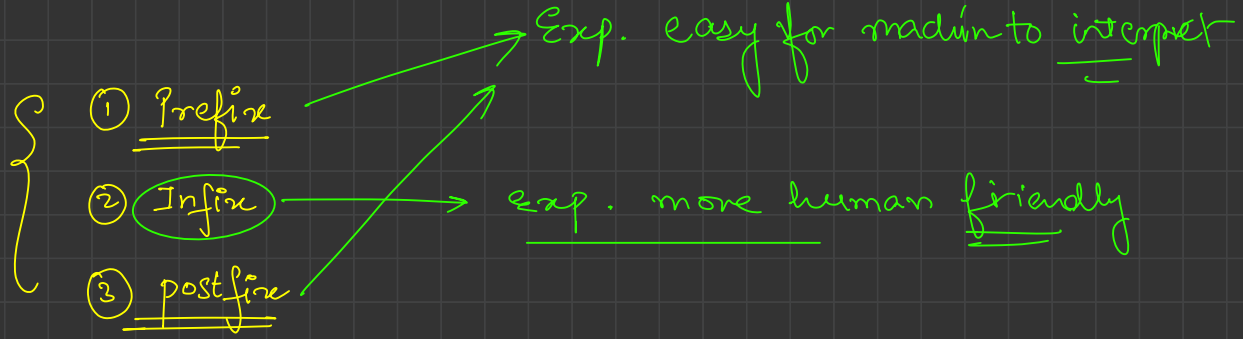
Brute force

{ TC: O(n^2) }
{ SC: O(1) }


int[][] arr =
0 1 2 3 4
0 1 1 1 1
1 0 1 1 1
1 1 0 1 1
1 1 1 0 1
0 0 0 0 0
4
celebStack

Infix Evaluation

Infix Expression



eg ⇒

V1 O V2

Infix : a + b

prefix : + ab → O V1 V2

postfix : ab + → V1 V2 O

all 3 exp. are same

Exp : ((2 + ((6 * 4) / 8)) - 3)

= ((2 + (24 / 8)) - 3)

= ((2 + 3) - 3)

= (5 - 3)

= 2

Evaluate

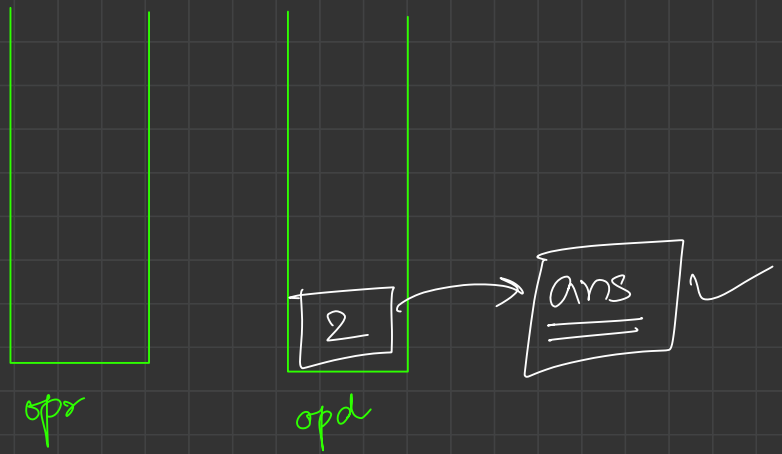
precedence

inc. ↑

*, /

+, −

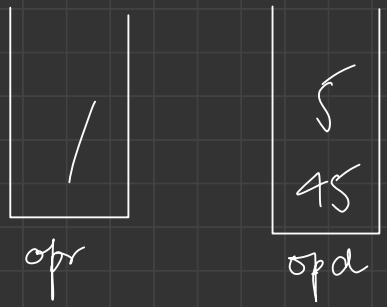
Exp : ((2+((6*4)/8))-3)



exp: (15*3)/5

45/5

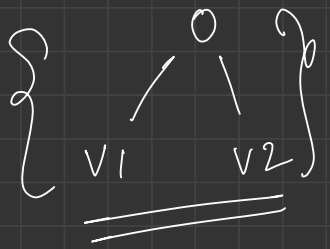
=9

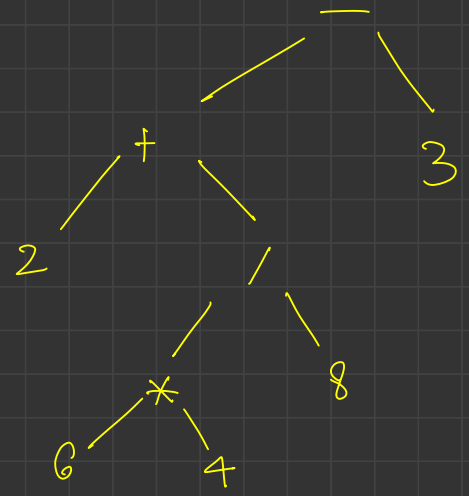


15*3

= 45

Exp. ((2+(((6*4)/8))-3)





V1 O V2

- + 2 / * 6 4 8 3

↳ prefix

Infix = V1 O V2

prefix = O V1 V2

postfix = V1 V2 O

$4 * 3 + 5$