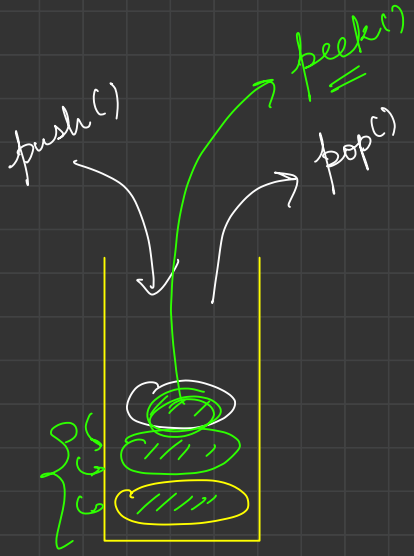
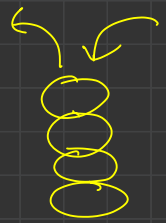
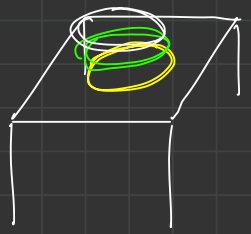
Stacks







last in, first out
(LIFO)

Stacks

→ push() → adds data in stack from top

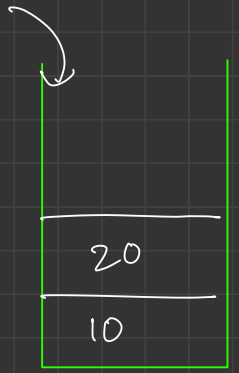
→ pop() → removes data from stack, from top pos.

→ peek() → access/view data on top

→ size() → gives the size of the stack

→ Stack follows LIFO behaviour

last in, first out.



push(10)

push(20)

peek() → 20

push(30)

pop() → remove 30.

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

↳ wrapper classes,
user defined classes

Extra brackets

exp = "( )" → extra

"(a+b)" → NO

"((a+b))" → extra

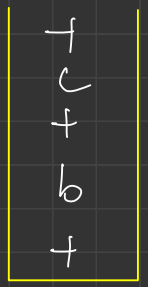
"((a) + (b))" → NO

A Bracket is useful when it has an exp. in Between.

exp = "((a+b) + (c))"

→ false!

exp = "(a) + b + c + ()"



→ extra bracket pair

↓

true!

TC: O(N)

SC: O(N)

Next Greater Element On Right

arr[] = { 2, 5, 9, 3, 1, 12, 6, 8, 7 }

nger[] = { 5, 9, 12, 12, 12, -1, 8, -1, -1 }

Approach 1 → try to pre process next greater element of each position.

{ Right to left }

arr[] = { 2, 5, 9, 3, 1, 12, 6, 8, 7 }

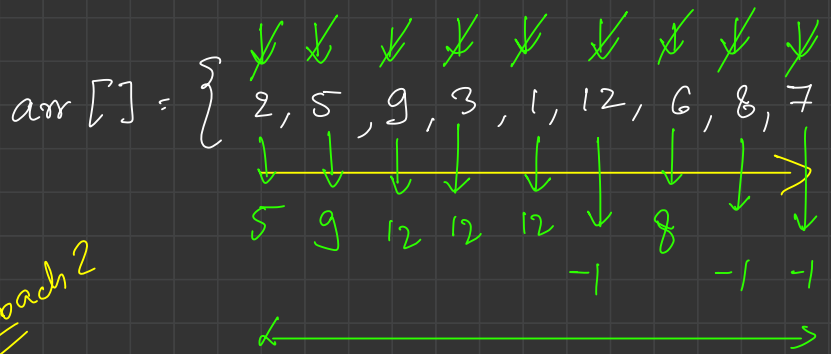
{ 5 9 12 12 12 -1 8 -1 -1 }

TC : O(N)

SC : O(N)

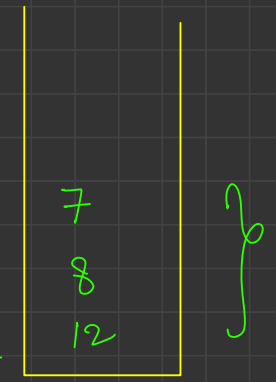
Stack

I have people who can be nger.



nger

Approach 2

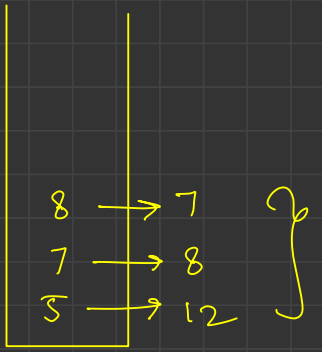


NOTE:

{ Store index instead of values.

Stack → {people looking for nger}

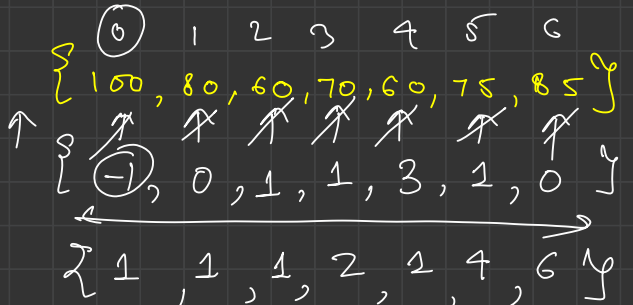
```java
public static long[] nextLargerElement(long[] arr, int n) {
    // Stack has people looking for there next greater element on right
    Stack<Integer> stack = new Stack<>();

    long[] nger = new long[arr.length];

    for (int i = 0; i < arr.length; i++) {
        long ele = arr[i];

        // can I be anyone's nger
        while (stack.size() > 0 && ele > arr[stack.peek()]) {
            int idx = stack.peek();
            stack.pop();
            nger[idx] = ele;
        }

        // now I will also looking for my nger
        stack.push(i);
    }

    // people left in stack doesn't have their next nger
    while (stack.size() > 0) {
        int idx = stack.pop();
        nger[idx] = -1;
    }

    return nger;
}
```



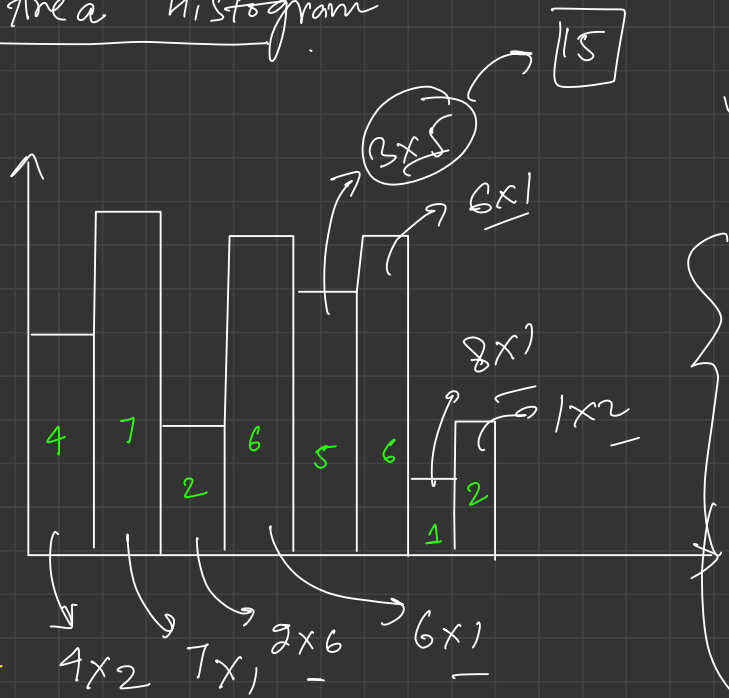
Stack → people looking nger!

{ TC : O(N) SC : O(N) }

```java
static int[] nextGreaterElementOnLeftIndex(int[] a) {
    int[] ngeli = new int[a.length];

    // Stack of people looking for there ngel
    Stack<Integer> stack = new Stack<Integer>();

    for (int i = a.length - 1; i >= 0; i--) {
        int ele = a[i];
        while (stack.size() > 0 && ele > a[stack.peek()]) {
            int idx = stack.peek();
            stack.pop();
            ngeli[idx] = i;
        }

        stack.push(i);
    }

    while (stack.size() > 0) {
        int idx = stack.pop();
        ngeli[idx] = -1;
    }

    return ngeli;
}

// TC: O(N), SC: O(N)
static int[] stockSpan(int[] a) {
    int[] ngeli = nextGreaterElementOnLeftIndex(a);

    int[] span = new int[a.length];
    for (int i = 0; i < a.length; i++) {
        span[i] = i - ngeli[i];
    }

    return span;
}
```



Largest Area Histogram



O(N)

① get nseli, nseri;

② width dist b/w nseli, nseri

③ Area = width × h

④ Maximise area

O(N)

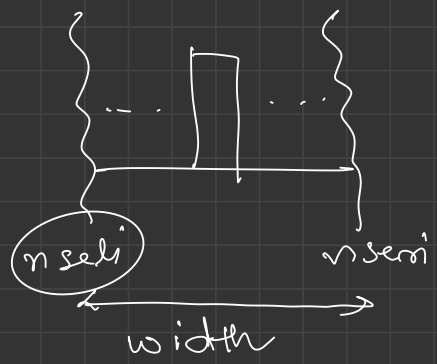
when not present

nseli = -1

nseri = n

TC: O(N)

	0	1	2	3	4	5	6	7
height[] :	4	7	2	6	5	6	1	2
nseli[] :	-1	0	-1	2	2	4	-1	6
nseri[] :	2	2	6	4	6	6	8	8



width = nseri - nseli - 1

Max area = ~~0~~ ~~8~~ ~~12~~ 15

```java
public static long maximumArea(long hist[], long n) {
    Stack<Integer> stack = new Stack<Integer>();
    long maxArea = 0;
    for (int i = 0; i < (int) n; i++) {
        long ele = hist[i];
        while (stack.size() > 0 && ele < hist[stack.peek()]) {
            int idx = stack.pop();
            int rb = i;
            int lb = -1;
            if (stack.size() > 0) {
                lb = stack.peek();
            }
            int width = rb - lb - 1;
            long area = hist[idx] * width;
            maxArea = Math.max(maxArea, area);
        }

        stack.push(i);
    }

    while (stack.size() > 0) {
        int idx = stack.pop();
        int rb = (int) n;
        int lb = -1;
        if (stack.size() > 0) {
            lb = stack.peek();
        }
        int width = rb - lb - 1;
        long area = hist[idx] * width;
        maxArea = Math.max(maxArea, area);
    }

    return maxArea;
}
```

nsemi

height[] : { 4, 7, 2, 6, 5, 6, 1, 2 }

0 1 2 3 4 5 6 7