① Storing With Parenthesis

{ ( ) } [ ]

It's balanced or not

count of opening bracket == count of closing bracket

not balanced

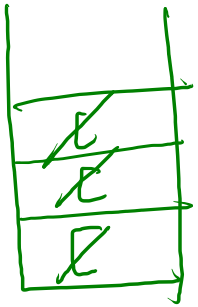
∴ can't rely on counting because the order matters.

=> i/p → [one[two[three]]]

```
function(str) {
    stack stk = []
    str2 = ""

    for(i = 0; i < str.length; i++) {
        if(str[i] == "(" || str[i] == "{" || str[i] == "[" ||
        str[i] == ")" || str[i] == "}" || str[i] == "]") {
            str2 += str[i]
        }
    }

    for(i = 0; i < str2.length; i++) {
        if (str2[i] == "(" || str2[i] == "{" || str2[i] == "[") {
            stk.push(str2[i])
        } else if (str2[i] == ")" || str2[i] == "}" || str2[i] == "]") {
            if(stk.isEmpty()) {
                return false
            }

            if((str2[i] == ")" && stk.top() == "(") ||
              (str2[i] == "}" && stk.top() == "{") ||
              (str2[i] == "]" && stk.top() == "[") ) {
                stk.pop()
            } else {
                return false
            }
        }
    }

    if(stk.isEmpty()) {
        return true
    } else {
        return false
    }
}
```

str → [one [two [three]]]

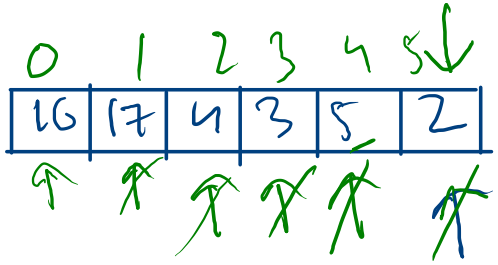
str2 → [[[]]]



stk.

② Marks & Competition

→ find the student/marks whose marks are greater than all the students on their R.H.S.

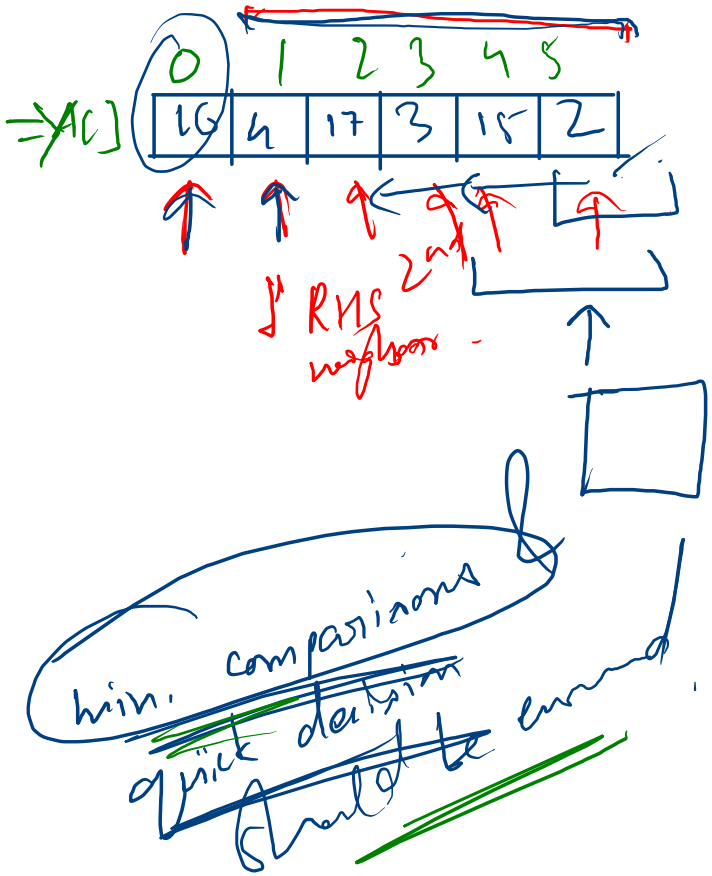


A[5] >= all in R.H.S ✓
A[4] >= all in R.H.S ✓
A[3] >= " " " ✗
A[2] >= " " " ✗
A[1] >= all in R.H.S ✓
A[0] >= " " " ✗

for each student there shouldn't be any neighbor who have greater marks & in R.H.S.

if true then include ans

ans = [2 | 5 | 17] ← reverse & print



for each item do you want to compare it with all the neighbours on R.H.S or if I give you a way that-

↳ ensures only greater item till time in what you need to compare with instead of all the items

min. comparisons
quick decision should be ensured

→ If I maintain neighborhood of greater items only & that too on a stack → means closest would be at the top of stack.

[I am trying to find the marks which are greater than all the marks in their R.H.S].

```
res = []
stack stk = []

for(i = n - 1; i >= 0; i--) {
    while(!stk.isEmpty() && A[stk.peek()] <= A[i]) {
        stk.pop()
    }

    if(stk.isEmpty()) {
        res.push(A[i])
    }

    stk.push(i)
}

return (reverse(res))
```

is there a neighbor

comparing marks

0	1	2	3	4	5
16	4	17	3	5	2

T.C O(n)

if my neighbor is smaller than me than I will pop him out & continue comparison with next neighbor till I found one greater than me or no more neighbor left to compare

LIFO

0
1
2
3
4
5

stk

A[2] <= A[1]
17 <= 4 X

A[1] <= A[0]
4 <= 16 ✓

A[2] <= A[0]
17 <= 16 X

contain R.H.S neighbors

If stack becomes empty means no neighbor to compare

comparator criteria & our reaction on result

means am greater than all in RHS

res → | 2 | 5 | 17 |

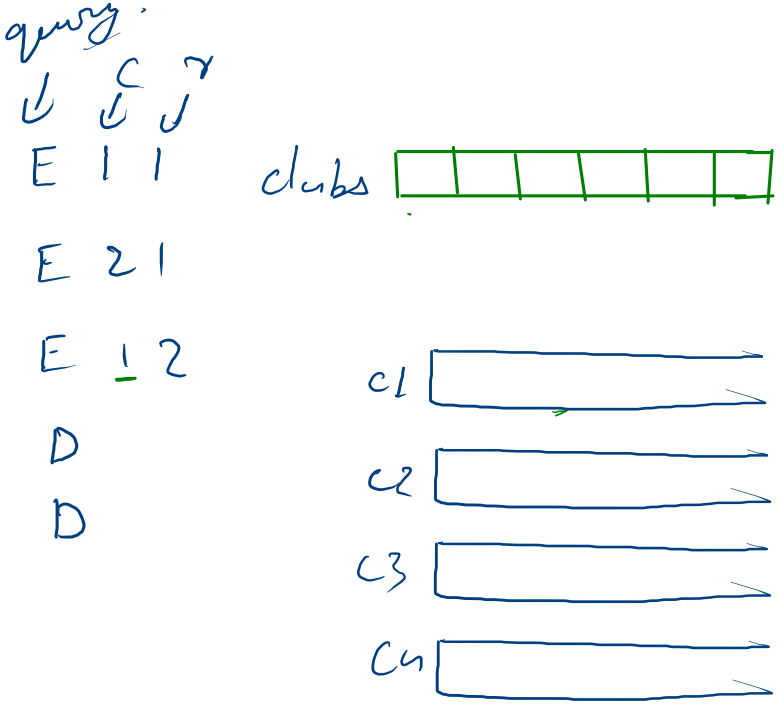
Break till

11:00 am

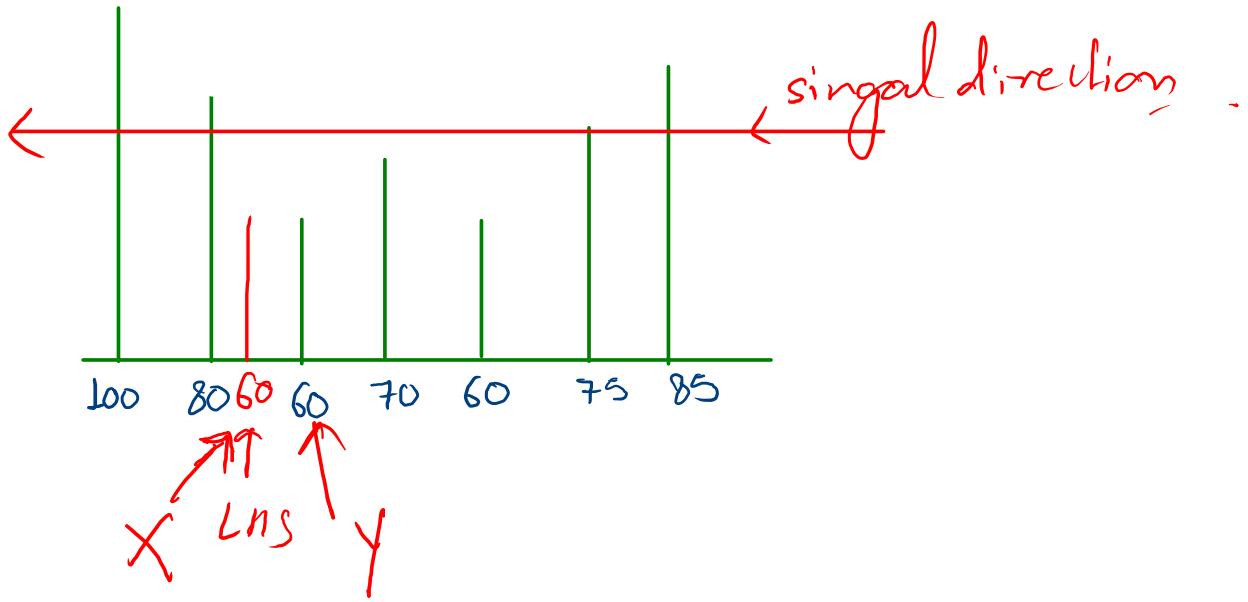
(3) Masai Coding Competition

Pseudo Code

```
clubs = [] //array to maintain clubs order
c1 = [], c2 = [], c3 = [], c4 = []  //4 queues to decicated clubs

if(query == 'E') {
        check = clubs.findIndex(c)  //if item not present , it returns -1

        if(check == -1) {
                clubs.push(c)
        }

        switch(c) {
                case 1 :
                        c1.push(r)
                        break
                case 2 :
                        c2.push(r)
                        break
                case 3 :
                        c3.push(r)
                        break
                case 4 :
                        c4.push(r)
                        break
        }

} else if(query == 'D') {
        switch(clubs[0]) {
                case 1:
                        print(clubs[0] + " " + c1.front())
                        c1.pop()
                        if(c1.length == 0) {
                                clubs.shift()
                        }
                        break
                case 2:
                        print(clubs[0] + " " + c2.front())
                        c2.pop()
                        if(c2.length == 0) {
                                clubs.shift()
                        }
                        break
                case 3:
                        print(clubs[0] + " " + c3.front())
                        c3.pop()
                        if(c3.length == 0) {
                                clubs.shift()
                        }
                        break
                case 4:
                        print(clubs[0] + " " + c4.front())
                        c4.pop()
                        if(c4.length == 0) {
                                clubs.shift()
                        }
                        break
        }
}
```



4 Signal's Capacity :-



no X

yes ✓

to block x taller than Y

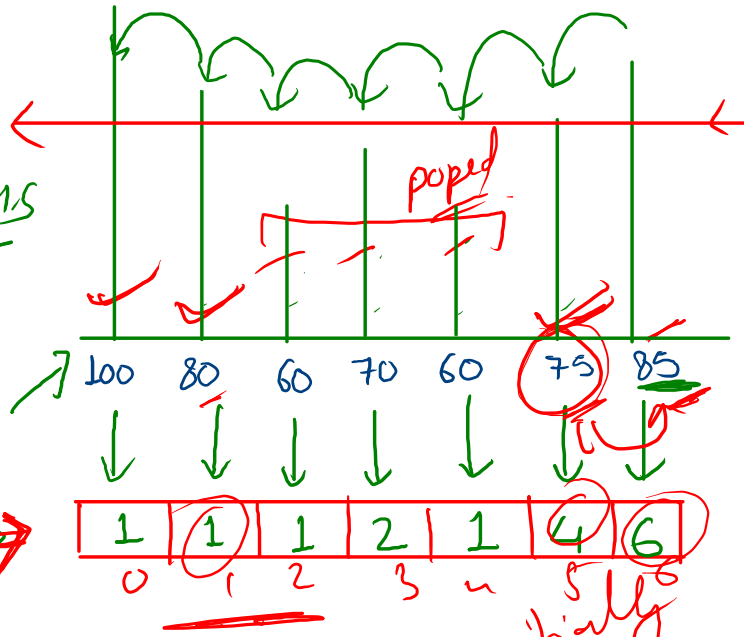
height of x ≥ height of Y

can X block Y's signal ? no !

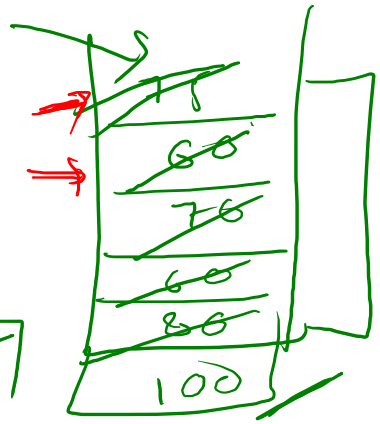
4 Signal's Capacity :-

Don't forget "contiguous" in LHS

start with closest neighbor in LHS.



A[stk.peek()] <= A[i]



=> were you going to count initially = 1+4+5+1+6

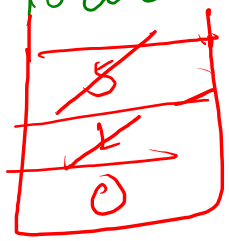
neighbor who are greater will be in stack

# of contiguous towers or not ?

75 <= 85 ✓ popped

80 <= 85 ✓ popped

100 <= 85 ✗



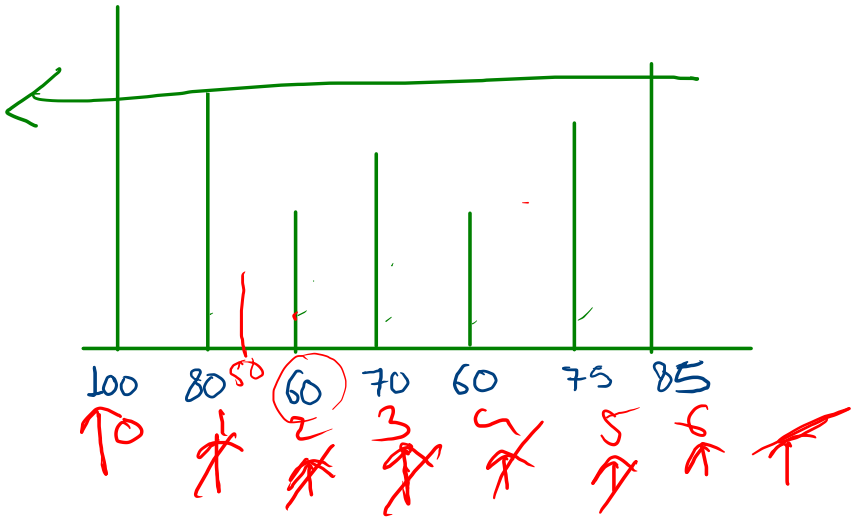
cont += ans[stk.top()]

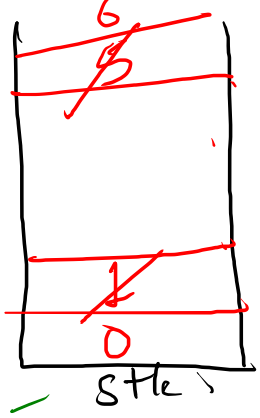
counting ;

```
stack stk = []
ans = []   //fill 0 initially (size is same as i/p array)

for(i = 0; i < n; i++) {
    count = 1
    while(!stk.isEmpty() && A[stk.peek()] <= A[i]) {
        count = count + ans[stk.peek()]
        stk.pop()
    }
    ans[i] = count
    stk.push(i)
}

print(ans)
```

Store the strength of original for others

T.C → O(n)





c = 1

A[5] <= A[6]

75 <= 85 ✓

c = 1 + 4 = 5

A[1] <= A[6]

80 <= 85 ✓

c = 5 + 1 = 6

A[0] <= A[6]

100 <= 85 X

(5) Unique Gift :

```
uniqueGift(str) {
	obj = {}
	queue Q
	ans = ""

	for(i = 0; i < str.length; i++) {
		if(obj[str[i]] == undefined) {
			obj[str[i]] = 1
		} else {
			obj[str[i]] += 1
		}

		Q.push(str[i])

		while(!Q.isEmpty()) {
			if(obj[Q.front()] == 1) {
				break
			}
			Q.pop()
		}

		if(Q.isEmpty()) {
			ans += "#"
		} else {
			ans += Q.front()
		}
	}

	return ans
}
```