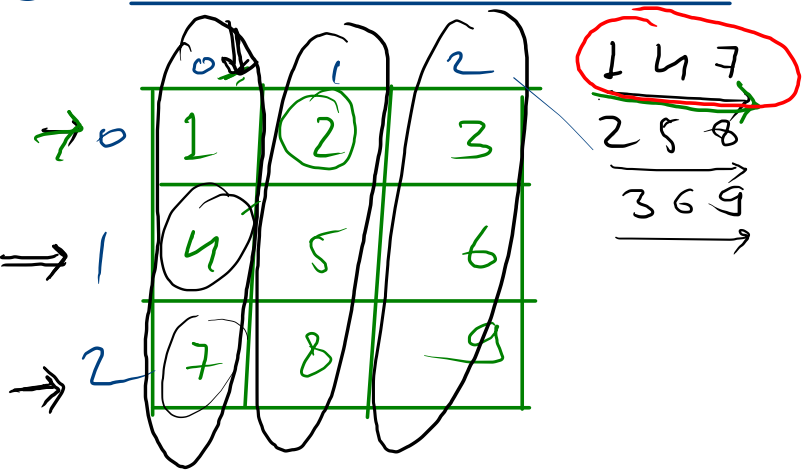
(1) Column Major :-



n x m ⇒ 3 x 3

A [n] [m]

for(i=0; i<3; i++) (column)

{

⇒ for(j=0; j<n; j++) (row)

{ A[j][i]

}

}

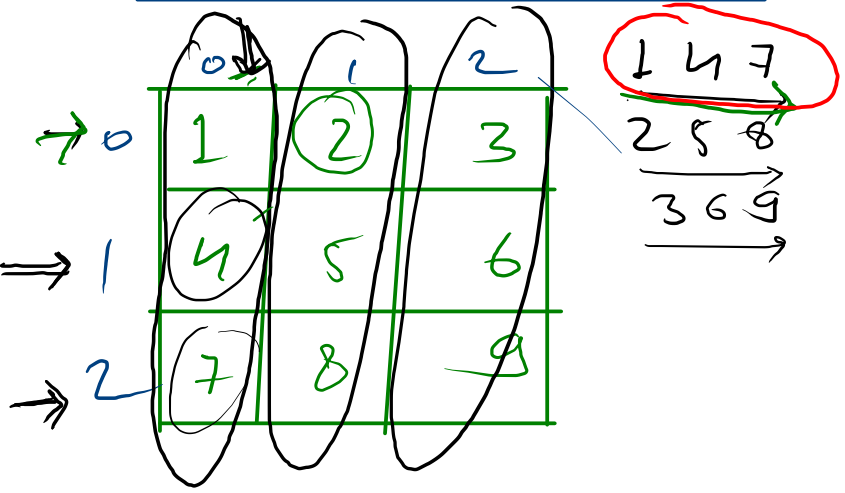
for each column traverse all the rows items.

i=0, j=0	i=0, j=1
A[0][0] → 1	A[0][1] → 2

row, col.

i=0	i=0	i=0	i=0
j=0	j=1	j=2	j=3
A[j][i]	A[j][i]	A[j][i]	3<3 ✗
A[0][0]	A[1][0]	A[2][0]	
↓	↓	↓	
1	4	7	

(1) Column Min :-



n x m ⇒ 3 x 3

ans = [ ] A [n] [m]

for(i=0; i<m ; i++) (column) (i<3)

{ min = Infinity ← max. Integer

⇒ for(j=0; j<n; j++)

{ if (min > A[j][i])

{ min = A[j][i]

}

}

ans.push(min).

}

print(ans[])

for each column traverse all the rows items.

i=0, j=0

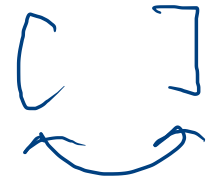
A[0][0] → 1

i=0, j=1

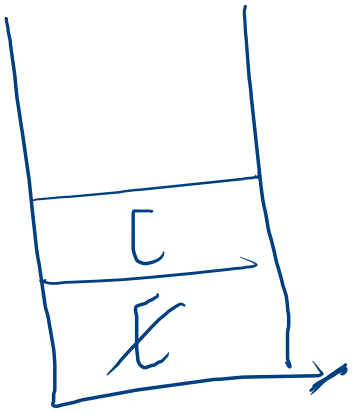
A[0][1] → 2

row, col^n.

dry run on custom i/p



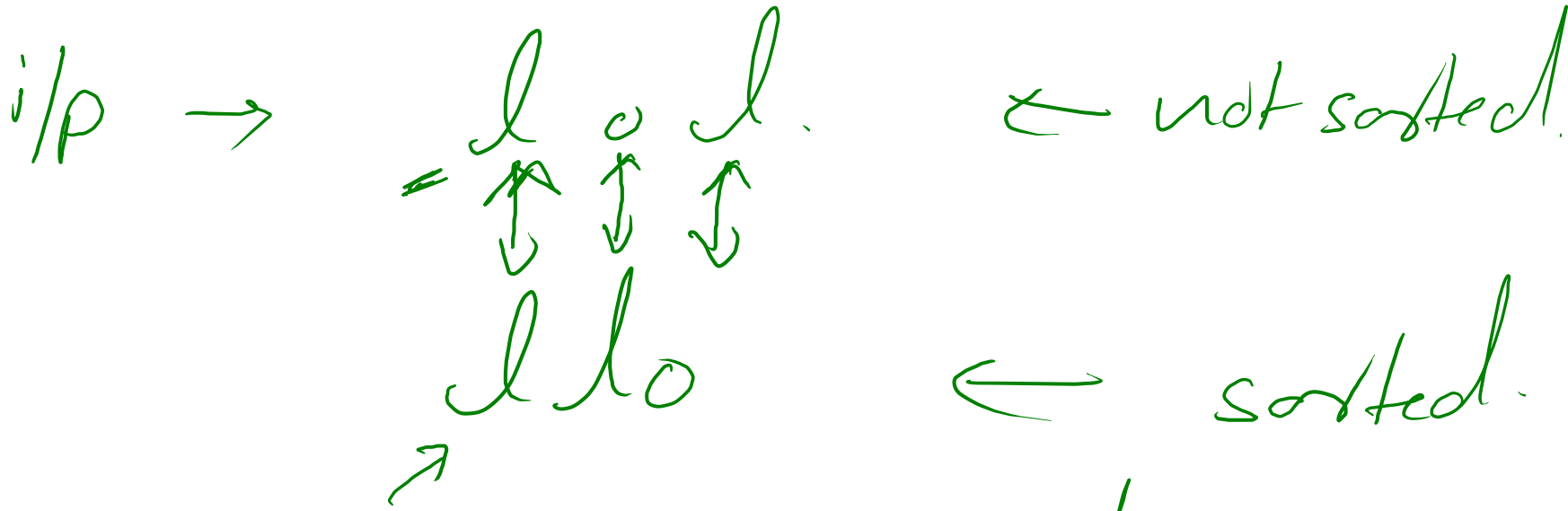
two items in 1 swap are swapped




```
ans = str.length()
if(ans%2==0)
{
    print(ans/2)
}else
{
    print((ans+1)/2)
}
```

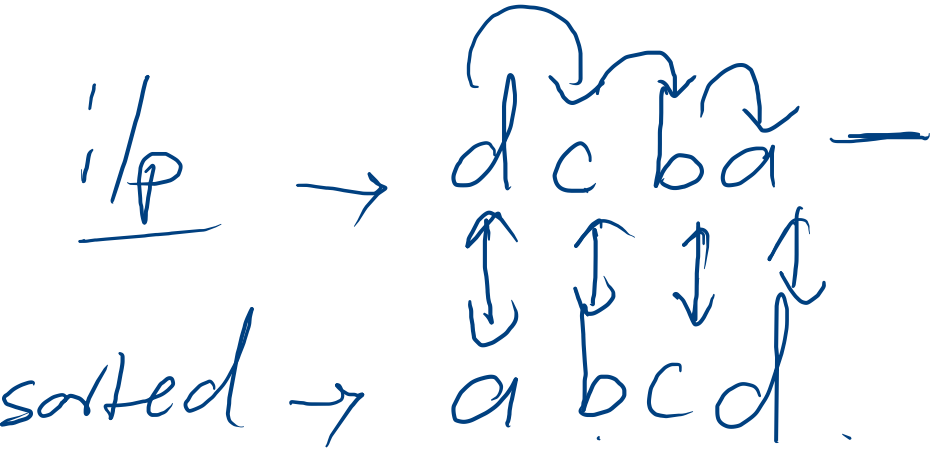
(3) Permutate & Sort :

we need to make an arrangement where the i/p string becomes sorted, tell me min value of k. took to sort the string!



How many positions did I need to arrange to get it sorted ?

↳ answer = 2



XZ3 4

→ compare characters by characters

↳ if not a match, means it got rearranged

↳ o/p → 4 here

i/p aaaa

sorted aaaa

0 rearrangements

∴ k = 0

```
sorted(str)
{
  s2 = str.sort()
  ans = 0
  for(i=0 to n-1)
  {
    if(str[i] != s2[i])
    {
      ans++
    }
  }
  print ans
}
```

1) Balanced Parenthesis :-

5 min. break.

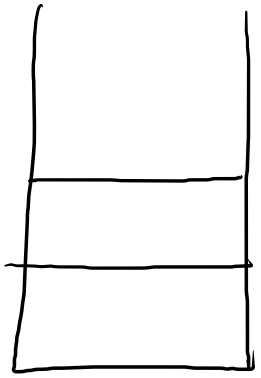
till 11:47 am →

Pseudo Code :-

```
stack st = []

for(i=0 ; i<str.length ; i++)
{
  if(st.isEmpty())          // if there is nothing to compare, put the current item on stack
  {  st.push(str[i])
  }
  else if((st.top()=='{' && str[i]=='}') ||
          (st.top()=='(' && str[i]==')') ||
          (st.top()=='[' && str[i]==']'))
  {   st.pop()
  }
  else
      st.push(str[i])
}
if(st.isEmpty()){return true}
else{return false}
```

length = n

str = "{( ){[ ]"



T.C →

S.C →

Can You Find The Sum.

Given :-

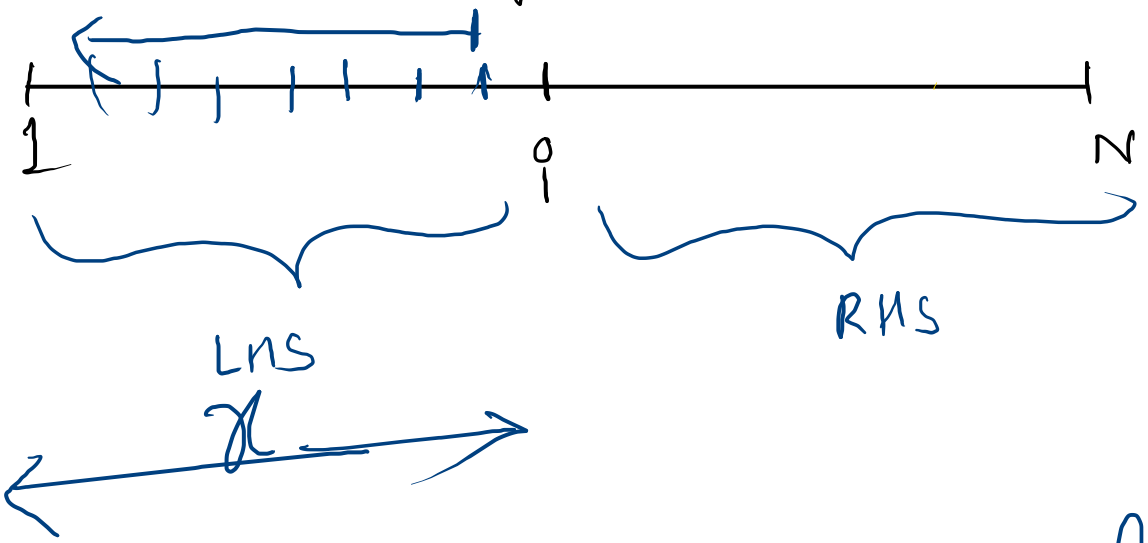
i, x & y are indices

as A[x] > A[i] &

A[y] > A[i]

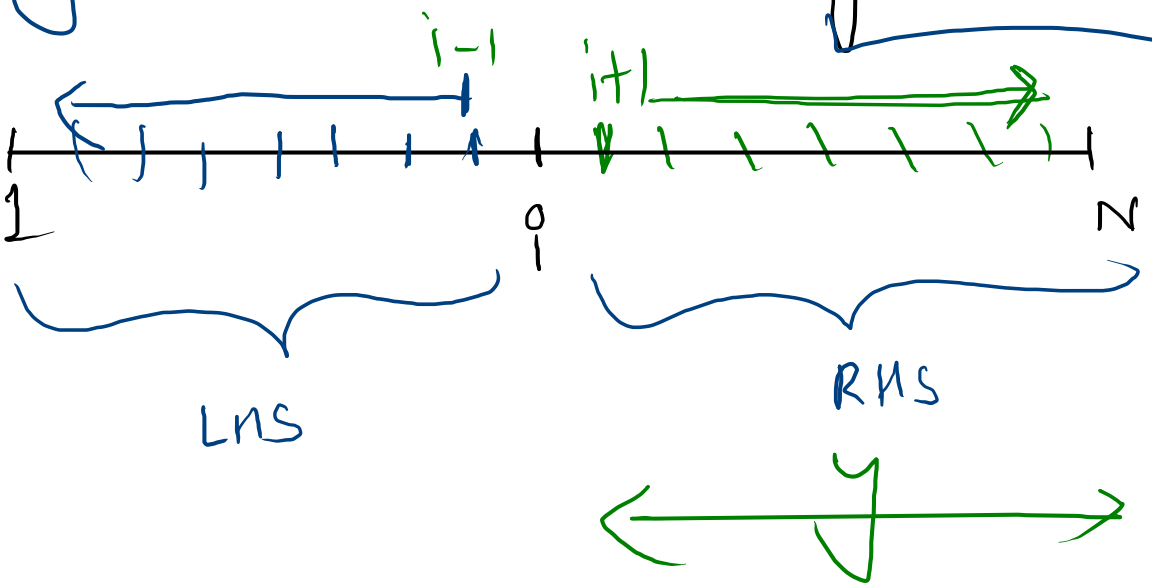
→ 1 ≤ i ≤ N ← for o/p consider 1-based indexing instead of 0-based indexing

→ x is largest # less than i



i − 1 is the 1st largest possible value for x

→ y is smallest # greater than i



$i+1$ is the 1st smallest possible value for y.

x → find the nearest neighbor > A[i]

on LHS

y → find the nearest neighbor > A[i]

on RHS

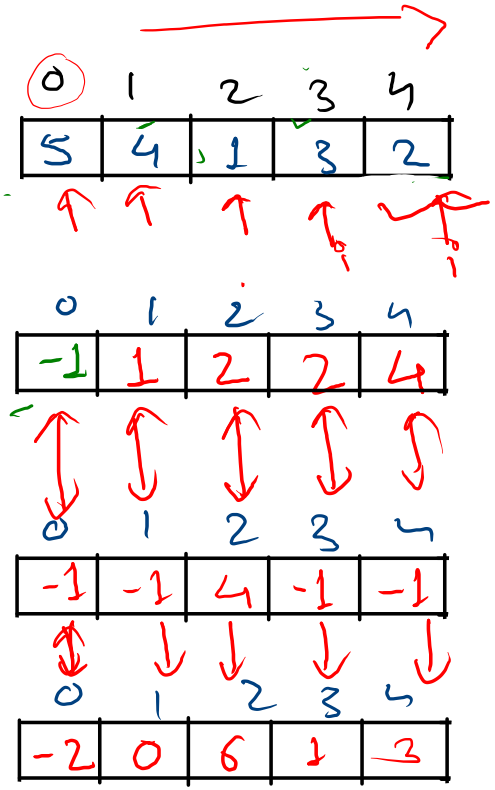
(2) Can You Find The Sum :-

x → find nearest neighbor > A[i] in LHS.

y → find nearest neighbor > A[i] in R.H.S.

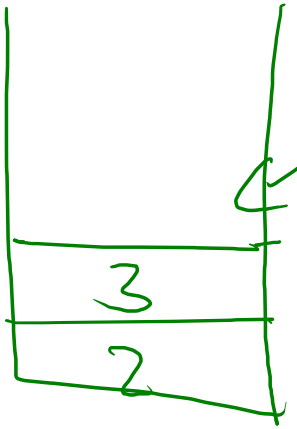
x → Left [] →

y → Right[] →

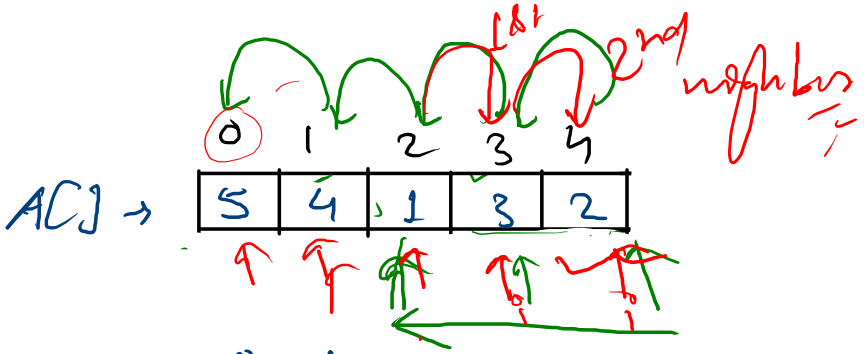


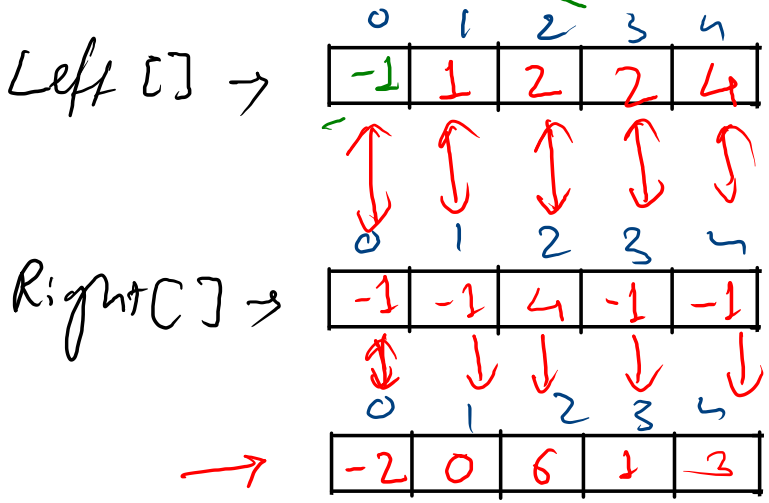
why Stack DS here?

↳ LIFO ⇐



last read item would be at the top always





→ maintaining the items of the stack.
is it worth it to check all item again & again

→ we should maintain neighbor which is greater only

Pseudo Code : (Left Greater Neighbor)

```
Left_neighbor(A[])
{
  stack stk = []
  res = []
  for(i=0 ; i<A.length ; i++)
  {
    while( !stk.isEmpty() &&
           A[stk.peek()] <= A[i] )
    {
      stk.pop()
    }
    if(stk.isEmpty())
    {  res[i] = -1
    }
    else
    {  res[i] = stk.peek() + 1
    }
    stk.push(i)
  }
  return res
}
```

smaller neighbour popped out

A[] →

0	1	2	3	4
5	4	1	3	2

left res [] →

0	1	2	3	4
-1	1	2	2	4

A[0] <= A[1]
5 <= 4 X

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

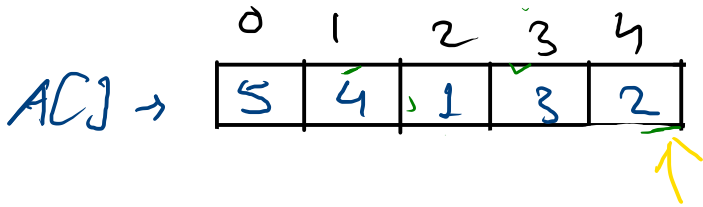
A[2] <= A[3]
1 <= 3 ✓

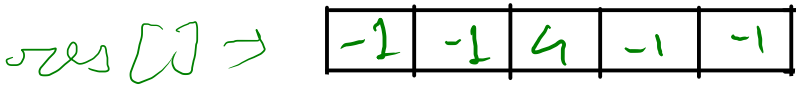
A[3] <= A[4]
3 <= 2 X

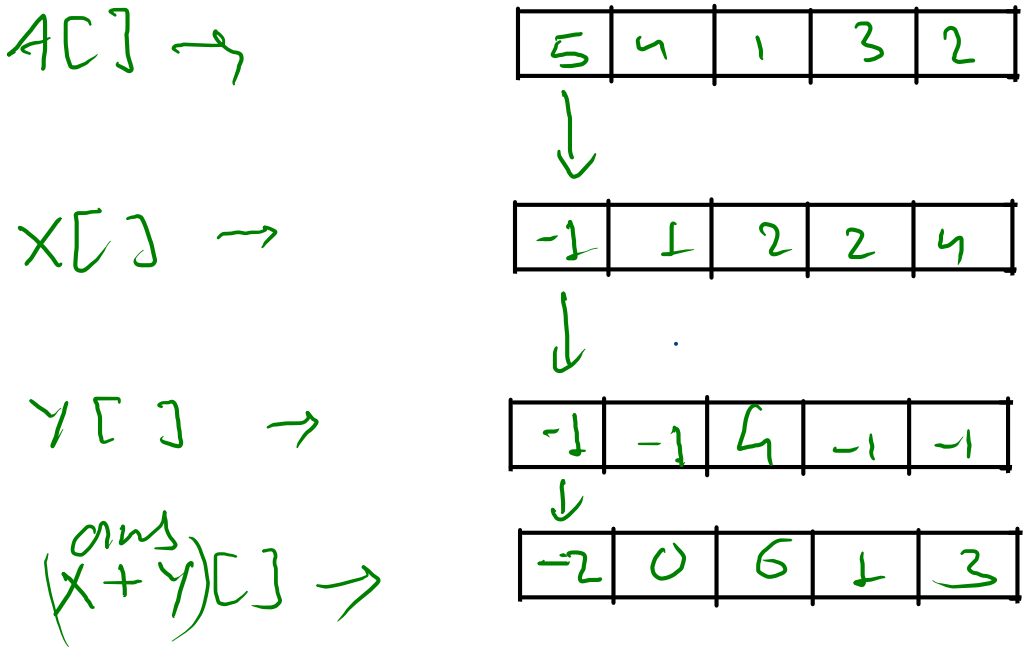
A[] <= A[3]
4 <= 3 X

to avoid repetition of neighbor check stk.

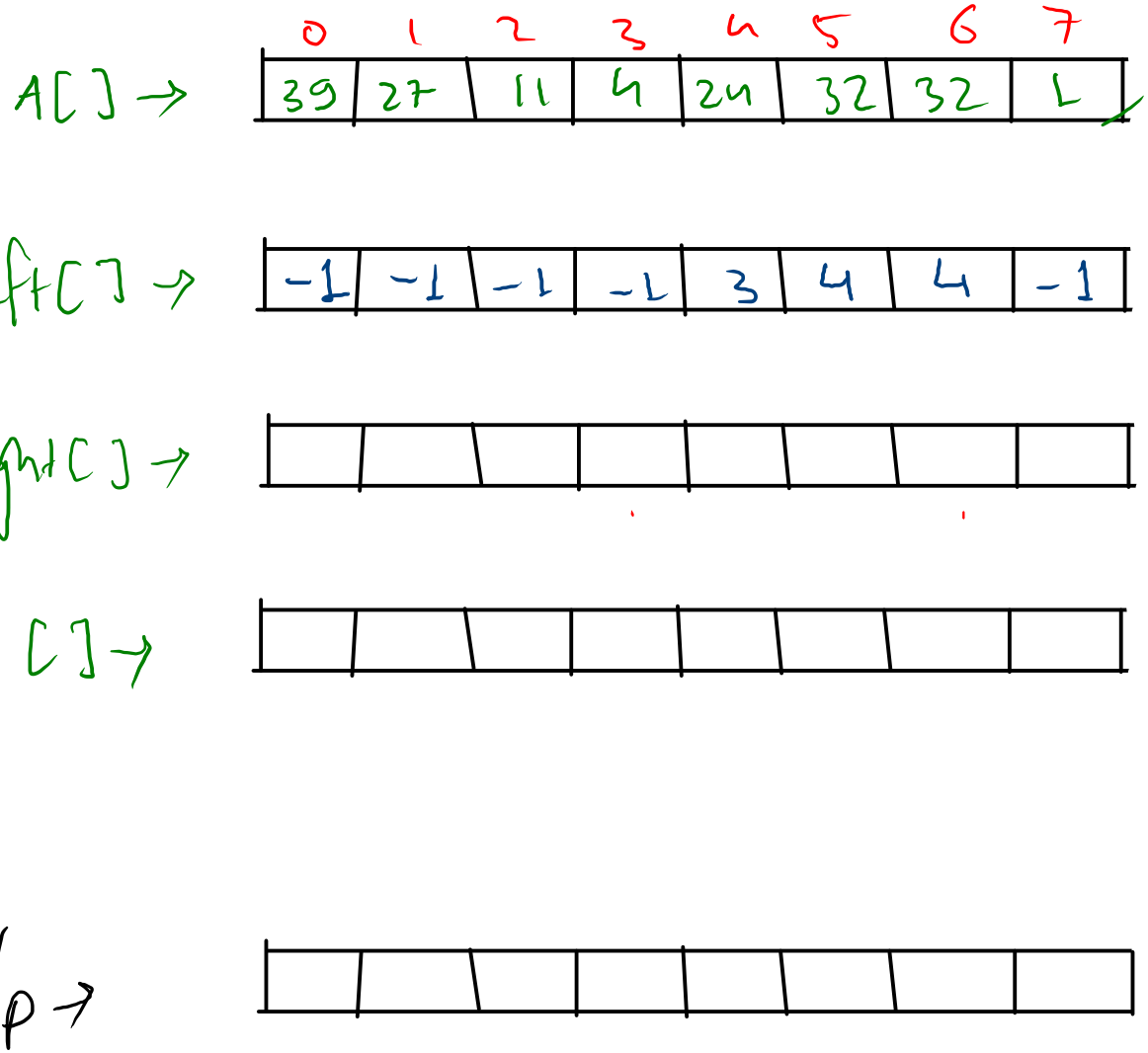
```
Right_neighbour(A[])
{
  stack stk = []
  res = []
  for(i = A.length-1; i>=0; i--)
  {
    while( !stk.isEmpty() &&
           A[stk.peek()] <= A[i])
    {
      stk.pop()
    }
    if(stk.isEmpty())
    {
      res[i] = -1
    }
    else
    {
      res[i] = stk.peek()+1
    }
    stk.push(i)
  }
  return res
}
```






A[] →
5 4 1 3 2
X[] →
-1 1 2 2 4
Y[] →
-1 -1 4 -1 -1
ans
(X+Y)[] →
-2 0 6 1 3

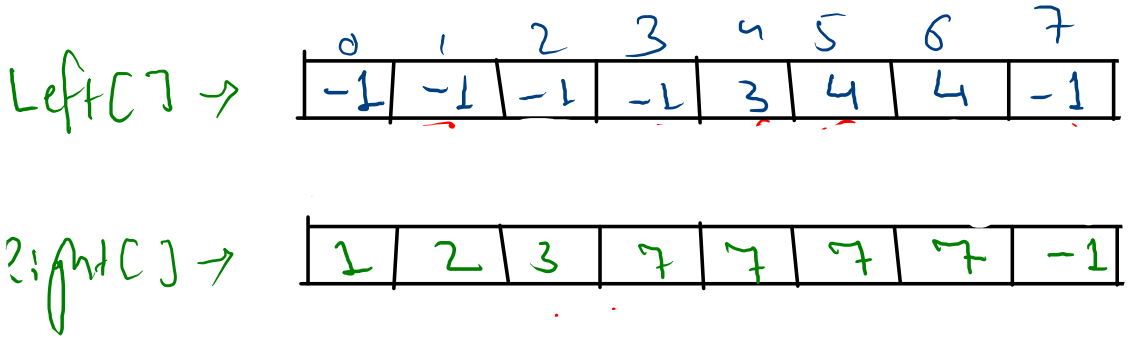
⇒ Nearest Smaller Element :-



```
nearestSmallerLeft(A[], n) {
	stack stk = []
	left = []

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

		if(tsk.isEmpty()) {
			left.push(-1)
		} else {
			left.push(stk.peek())
		}

		stk.push(i)
	}

	return left
}
```

```
nearestSmallerRight(A[], n) {
	stack stk = []
	right = []

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

		if(tsk.isEmpty()) {
			right.push(-1)
		} else {
			right.push(stk.peek())
		}

		stk.push(i)
	}
	reverse(right)
	return right
}
```

```
ans = []
for(i = 0; i < n; i++) {
        if(left[i] == -1 && right[i] == -1) {
                ans.push(-1)
        } else if(left[i] == -1 && right[i] != -1) {
                ans.push(right[i])
        } else if (left[i] != -1 && right[i] == -1) {
                ans.push(left[i])
        } else {
                d1 = abs(i - left[i])
                d2 = abs(i - right[i])

                if(d1 == d2 || d1 < d2) {
                        ans.push(left[i])
                } else if(d1 > d2) {
                        ans.push(right[i])
                }
        }
}
```




```
for(i=0; i<n; i++)
{ if(ans[i] != -1)
    print(A[ans[i]]);
  else print(-1)
}
```