Day-8

Consider the following program written in pseudo-code. Assume that x and y are integers.

```
Count (x, y) {
    if (y !=1 ) {
        if (x !=1) {
            print("*");
            Count (x/2, y);
        }
        else {
            y=y-1;
            Count (1024, y);
        }
    }
}
```

The number of times that the $print$ statement is executed by the call $Count(1024, 1024)$ is ____

```
Count (x, y) {
    if (y !=1 ) {
        if (x !=1) {
            print("*");
            Count (x/2, y);
        }
        else {
            y=y-1;
            Count (1024, y);
        }
    }
}
```

$x = 8$ $y = 8$

$8 \to 2^3$

Given

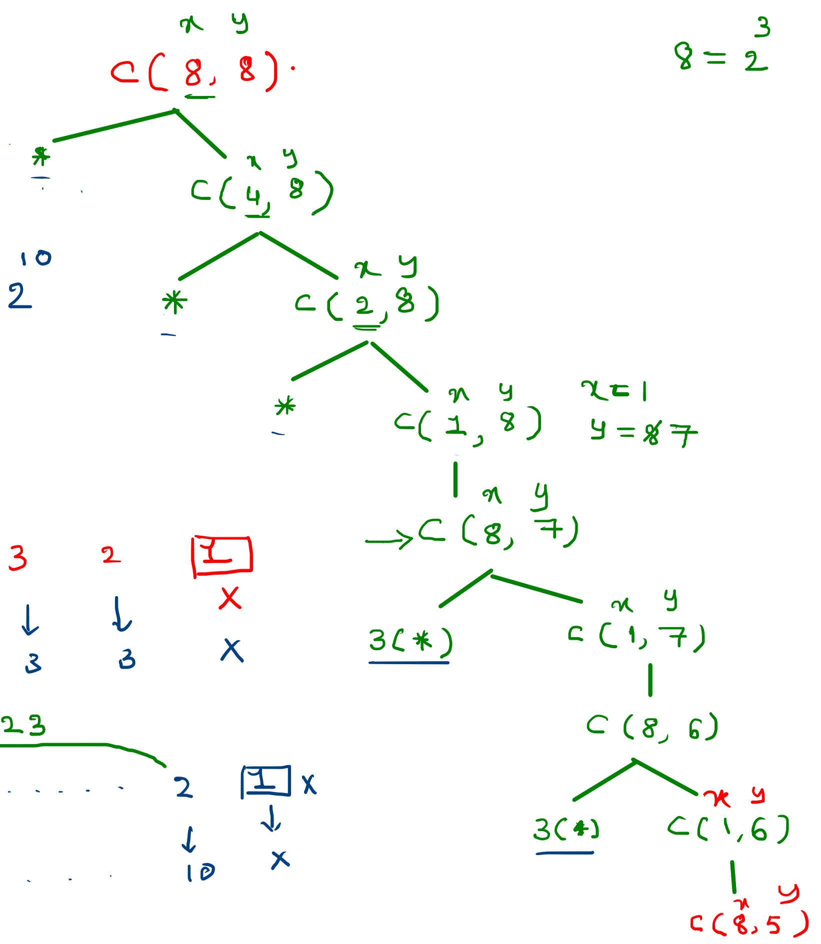
$x = 1024 \to 2^{10}$

$y = 1024$

C(1024, 1024)

$2^{10} = 1024$

$8 = 2^3$



y =	8	7	6	5	4	3	2	$\boxed{1}$ X
# of (*)	3	3	3	3	3	3	3	X

$$\to y = \underbrace{1024, \; 1023, \; 1022, \; \ldots \; 2}_{1023} \;\; \boxed{1}\ X$$

10   10   10 ... 10   X

$= 1023 \times 10 = 10230$

Largest Rectangular Area in a Histogram

n = 7

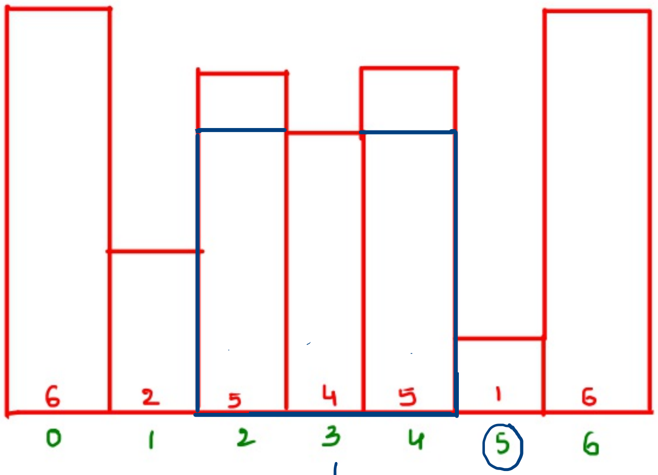
i/p

0	1	2	3	4	5	6
6	2	5	4	5	1	6

Area = l x b

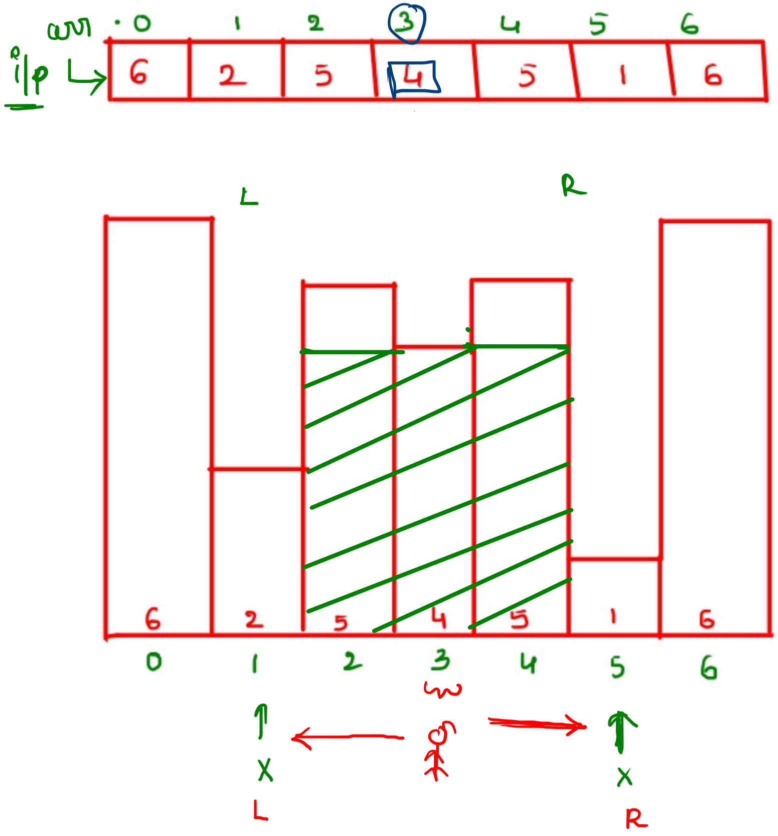
arr[1] = 5 x 2 = 10 units

→ largest Area possible.



3 x 4 = 12 units

↳ 7 x 1 = 7 units



$arr[3] = 4$

↳ Area = ?

↳ $\underbrace{\text{length}}_{3} \times \underbrace{\text{breadth}}_{4}$

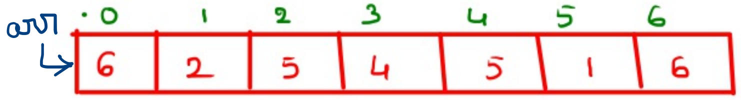
L = 1 ✓

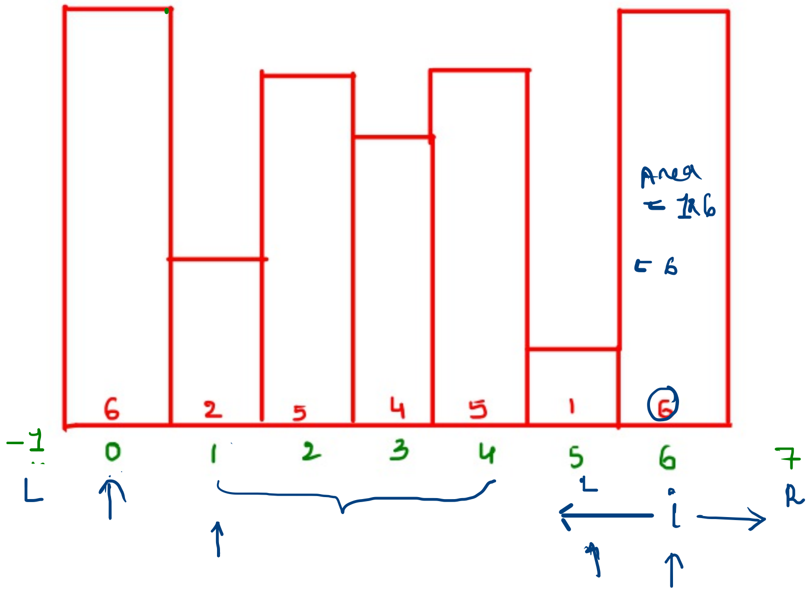
R = 5 ✓

5-1-1 = 3

length = (R-L-1)

BF





Length = (R − L − 1)

for(i=1; i<n-1; i++) 1 − −1 − 1

{

2 − 1 = 1

arr[0] = 6 → Area = ?

↳ length = ? 1

↳ Breadth = 6

Length = 7 − 5 − 1 = 1 ✓

$\rightarrow$ for( ) $\rightarrow n$

{

$\rightarrow$ for( ) $\rightarrow n$

{

}

$\rightarrow$ for( ) $\rightarrow n$

{

}

}

for( )

{

}

$n[n+n] + n$

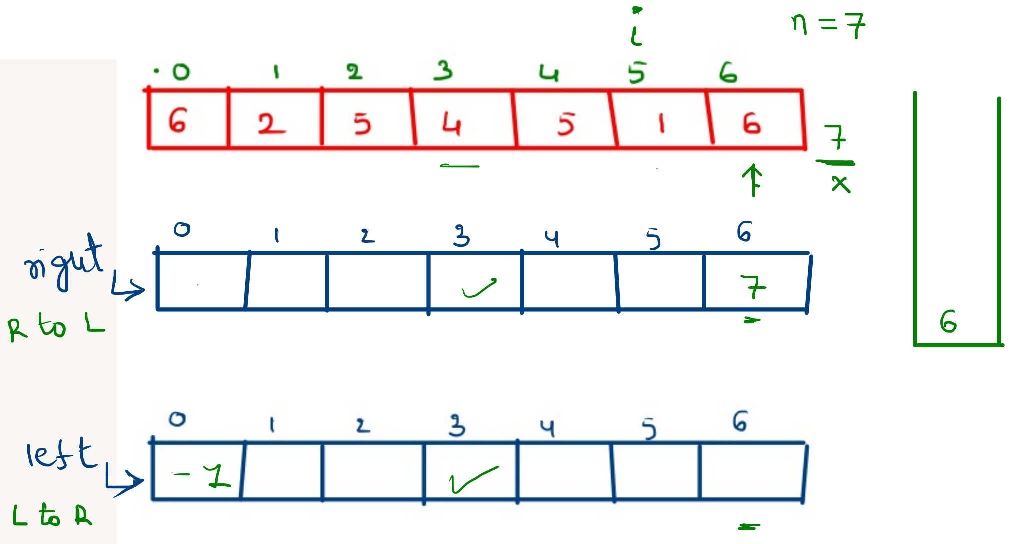
$= n^2+n^2+n = 2n^2+n$

$\therefore O(n^2) \rightarrow$ T.C

$\therefore O(1) \rightarrow$ S.C

→ Stack Approach

```
int maximumArea(int arr[],int n)
{
  stack<Integer> st = new Stack<Integer>
  int right[n] //nse index on right
  st.push(arr.length-1);
  right[arr.length-1]=arr.length;
  for(i=n-1;i>=0;i--)
  {
      while(st.size>0 && arr[i]<arr[st.peek()])
      {
          st.pop()
      }
      if(st.size()==0)
          right[i]=arr.length()
      else
          right[i]=st.peek()
      st.push(i)
  }
```



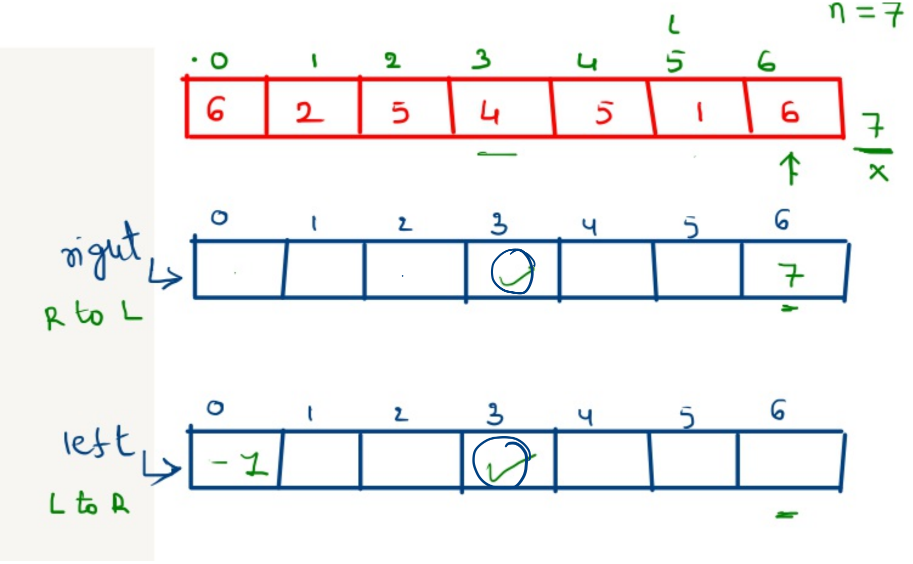
Note:- Right, Left arr, we are storing array indices

```
int left[n] // nse index on left
st=new Stack<>()
st.push(0)
left[0]=-1
for(i=1;i<arr.length;i++)
{
     while(st.size()>0 && arr[i]<arr[st.peek()])
     {
            st.pop()
     }
     if(st.size()==0)
     {
           left[i]=-1
      }
      else
      {
           left[i]=st.peek()
      }
   st.push(i)
}
```

~~width~~ = R - L - 1

```
maxArea=0
for(i=0;i<arr.length;i++)
{
    width=right[i]-left[i]-1;
    area=arr[i]*width
    if(area>maxArea)
    {
        maxArea=area
    }
}
return maxArea;
}
```

width → length



-1

T.C :- $n + n + n = 3n$

$\hookrightarrow O(n)$

S.C : $n + n = 2n$

$\hookrightarrow O(n)$

*** ↦ ( ), { }, [ ]

② Balanced parentheses problem → pure Application of stack

↳ every compiler ;

" [{ }] " → ✓

① # of open bracket = # of closed

② start with open bracket

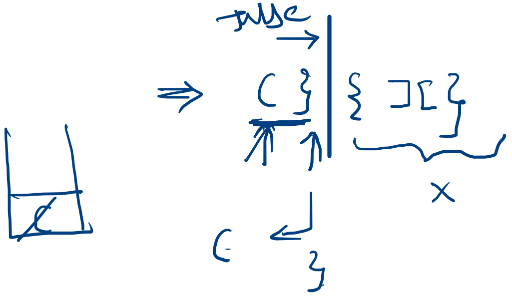
③ order

Ex1 :- ({) } ⇒ ✗

↳ ( ) { }

Ex2 :- [ ] ({ }) ⇒ ✓

```
function isBalanced(s[],n)
{
*   let st be a empty stack
    for(i=0;i<n;i++)
    {
        if(s[i]=='(' || s[i]=='[' || s[i]=='{')
        {
            st.push(s[i])
        }
```




```
        else
        {
            if(stack.size==0)
                return false
            switch (x)
            {
              case ')':
               check = stack.pop();
               if (check == '{' || check == '[')
                   return false;
               break;

              case '}':
               check = stack.pop();
               if (check == '(' || check == '[')
                   return false;
               break;

             case ']':
              check = stack.pop();
              if (check == '(' || check == '{')
                  return false;
              break;
            }
        }
    }
    return (st.size()==0);
*
}
```

(} [ ] ( ) { }

?

Minimum number of Platforms required

Assume time is 24 h format

$n = 6$

$t_1 \quad t_2 \quad t_3 \quad t_4 \quad t_5 \quad t_6$

```
arr[] = { 900, 940, 950, 1100, 1500, 1800 }
dep[] = { 910, 1200, 1120, 1130, 1900, 2000 }
```

BF

Max = 6 PF are required so that No collision

1

PF1 PF2 PF3

~~t_1~~ ~~t_3~~ t_4

~~t_2~~ t_6

15:00 → t_5

⇒ 3 PF are sufficient

T.C ⇒ O(n²)

S.C ⇒ O(1)

→ # of trains

```
function findPlatform(arr[], dep[],n)
{
      plat_needed = 1, result = 1;
      for  (i = 0; i < n; i++)
      {
        plat_needed = 1;
        for  (j = i + 1; j < n; j++)
        {
            // check for overlap
            if  ((arr[i] <= arr[j] && arr[i] <= dep[j]) || (arr[j] >= arr[i]&& arr[j] <= dep[i]))
                plat_needed++;
        }

        // update result
        result = Math.max(result, plat_needed);
      }

      return result;
}
```

n → (outer loop) n → (inner loop)

```
arr[] = { 900,  940,   950,   1100,   1500,   1800  }

dep[] = { 910,  1200,  1120,  1130,   1900,   2000  }
```

0 1 2 3 4 5

i →

Local sum

Max_sum = ( )

↑ final result

→ Greedy ← optimization(longest / smallest / min / max etc. . . .)

arr[] = { 900, 940, 950, 1100, 1500, 1800 }

dep[] = { 910, 1200, 1120, 1130, 1900, 2000 }

t_1 t_2 t_3 t_4 t_5 t_6

sorted

t_1 t_2 t_3 t_4

arr[] = { 900, 940, 950, 1100, 1500, 1800 }

dep[] = { 910, ~~1120~~, 1130, 1200, 1900, 2000 }

t_3

use

→ DP → optimization

\+

→ choices(include / don't include

polynomial T.C

Exponential T.C → { $O(2^n)$ or $O(3^n)$ final soln)

Time	Event Type	Total Platforms Needed at this Time
9:00	Arrival	$\boxed{1}$
9:10	Departure	$\boxed{0}$
9:40 t_2	Arrival	1
9:50 t_3	Arrival	2
11:00	Arrival	3
11:20	Departure	$\boxed{2}$
11:30	Departure	$\boxed{1}$
12:00	Departure	$\boxed{0}$
15:00	Arrival	1
18:00	Arrival	2
19:00	Departure	$\boxed{1}$
20:00	Departure	$\boxed{0}$

3

Minimum Platforms needed on railway station

= Maximum platforms needed at any time

= 3

```
function findPlatform(arr[], dep[], n)
{

    Arrays.sort(arr);
    Arrays.sort(dep);

    plat_needed = 1, result = 1;
    i = 1, j = 0;

    while (i < n && j < n)
    {

        if (arr[i] <= dep[j])
        {
            plat_needed++;
            i++;
        }

        // Else decrement count of platforms needed
        else if (arr[i] > dep[j])
        {
            plat_needed--;
            j++;
        }


        if (plat_needed > result)
            result = plat_needed;
    }

    return result;
}
```

(1) (2) → nlogn

n

T.C O(nlogn)

S.C O(1)