Stack

~ Arbind

Stack

1) Nearest greater to left
2) Nearest greater to right
3) Nearest smaller to left
4) Nearest smaller to right

1 concept

5) Stock Span Problem
6) Max. area of histogram

7) Max area of rectangle in Binary Matrix.

8) Rain water Trapping
9) Implementing a min stack
10) Implement stack using heap
11) The celebrity problem
12) Longest Valid Parenthesis
13) Iterative Tower of Hanoi

Identification

1) Array
2) $O(n^2)$

j → 0 to i
j → i to 0
j → i to n
j → n to i

```
for (int i = 0 ; i < n ; i++)
    for (int j = 0 ; j < i ; j++)
```

'j' is dependent on i → stack

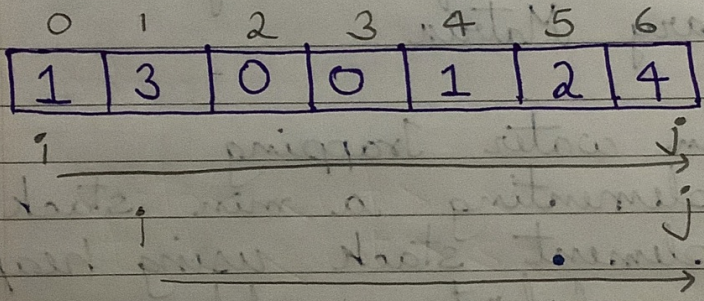
Q> Next largest element aka nearest greater to Right

arr[]: 1 3 2 4

3 4 4 -1

arr[]: 1 3 0 0 1 2 4

3 4 1 1 2 4 -1

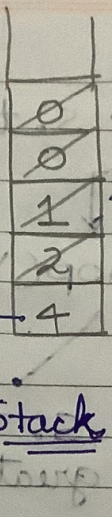


Brute force:

```
for (int i=0; i<n-1; i++)
  for(int j=i+1; j<n ; j++)
```

⇒ 1 3 ~~0 0 1 2~~ 4

4



LIFO

↳ So we must start pushing the elements from the last element of the array

s.top() > arr[i]

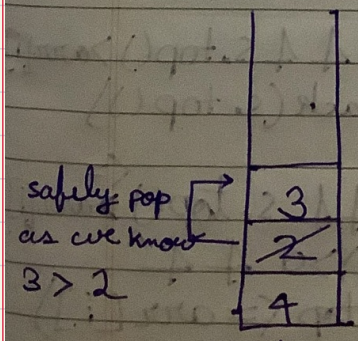
Note:

→ if stack is not empty then s.top() is the ans

→ if stack is empty the required element doesn't exists, so return -1

1 3 2 4 O/P: 3 4 4 -1

vector



→ check in stack then push that element in that stack

vector v:

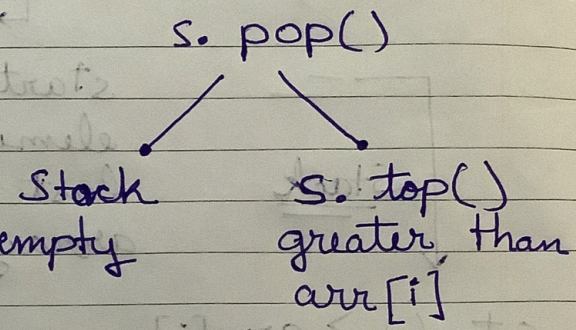
-1	4	4	3

vector v:

reverse(v.begin(), v.end());

Concept :

→ stack empty → -1
→ s.top() > arr[i] → s.top()
→ s.top() <= arr[i] →



code :

```
vector <int> v;
stack <int> s;

for(int i = size - 1 ; i >= 0 ; i--)
    if (s.size() == 0)
        v.push_back(-1);

    else if(s.size()>0 && s.top()>arr[i])
        v.push_back(s.top())

    else if(s.size()>0 && s.top() <= arr[i])
        while(s.size()>0 &&
              s.top <= arr[i])
            s.pop()
```

```
if(s.size() == 0)
    v.push_back(-1);
else
    v.push_back(s.top());
```

```
s.push(arr[i]);
```

```
reverse(v.begin(), v.end());
```

Q) Next largest element aka nearest greater to Left

arr[] :

1	3	2	4
↑	↑	↑	↑
-1	-1	3	-1

Brute force:

```
for(int i=0 ; i<n ; i++)
    for(int j=i-1; j>=0; j--)
```

stack

1	3	2	4

vector

-1	-1	3	-1

~~2~~
~~3~~
~~1~~

Change

1) Traversed the arr from Left to Right

2) No Reversal Required

Code Changes:

```
for (int i=0 ; i < size ; i++)
```

changes in the for loop

```
reverse (v.begin(),v.end());
```

this is removes

2) Next smaller to left + Nearest smaller element

arr [] :

4	5	2	10	8
↑	↑	↑	↑	↑
-1	4	-1	2	2

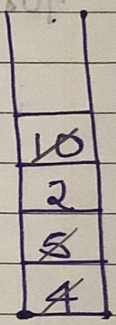
Brute force :

```
for(int i=0 ; i<n ; i++)
    for(int j=i-1 ; j>0 ; j--)
```

stack

arr[] : | 4 | 5 | 2 | 10 | 8 |

vector : | -1 | 4 | -1 | 2 | 2 |



Stack

Changes
1) Greater → Smaller
2) No need to reverse
3) Left Traverse

Changes in Code :-

```
for(int i=0 ; i<size ; i++)        → change in iteration
else if (s.size()>0 && s.top()<arr[i])
        v.push_back(s.top())
else if (s.size()>0 && s.top()>=arr[i])
    while(s.size()>0 && s.top()!=arr[i])
reverse(v.begin(), v.end());
```

↳ this is removes

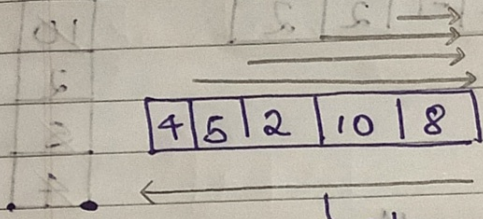
Q> Nearest Smallest To: Right ?

arr: | 4 | 5 | 2 | 10 | 8 |

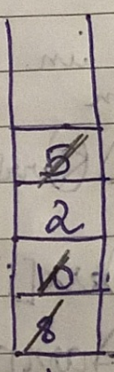
↑	↑	↑	↑	↑
2	2	-1	8	-1

Brute Force:

```
for(int i=0 ; i<n ; i++)
    for(int j=i+1 ; j<n ; j++)
```



arr: | 4 | 5 | 2 | 10 | 8 |



vector: | -1 | 8 | -1 | 2 | 2 |

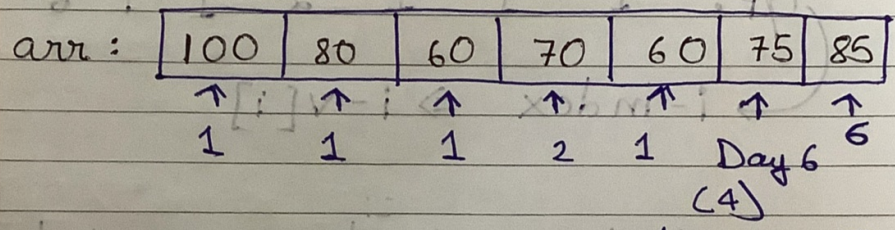
Reverse ↓

2 2 -1 8 -1

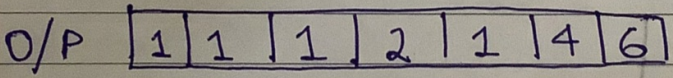
Changes

1) Right to left
2) Smaller
3) Vector is reversed

Stock Span Problem



consecutive smaller (or) equal before it



Concept getting used :-

Nearest greatest to left

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

consecutive smaller (or) equal to before it = Nearest greater to left

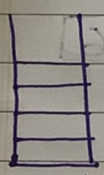
$\left(\frac{index}{NGL} - i\right)$

i:	0	1	2	3	4	5	6
	100	80	60	70	60	75	85
index vector → V:	-1	0	1	1	3	1	0

O/P: 1 1 1 2 1 4 6

→ i - index ⇒ i - V[i]

→ We have to store the index of NGL index



Stack

(NGL, index) ⇒ pair

Code Changes:

NGL index

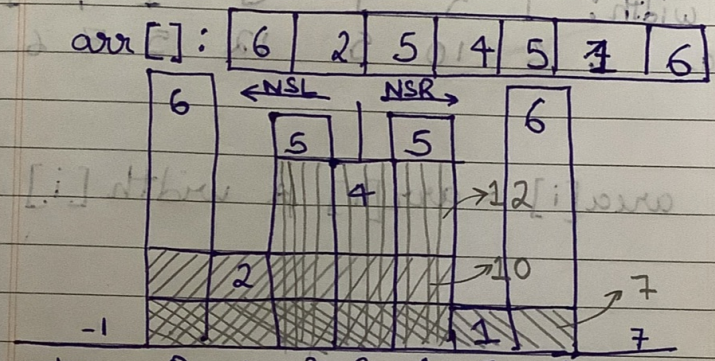
→ stack <pair<int, int>> s;

→ s.top().second → s.top().first > arr

→ s.push({arr[i], i});

```
for (int i=0; i< v.size(); i++)
{
    v[i] = i - v[i]
}
return v;
```

Maximum Area Histogram



Concept:

⇒ A building can be expanded into building only when other building >= current building

So (5 - 1) = 1

NSR index		0	1	2	3	4	5	6 → 7
	arr :	6	2	5	4	5	1	6
	↳ right :	1	5	5	5	5	7	7
[i]	↳ left :	-1	-1	1	1	3	-1	5
		1	5	1	3	1	7	1

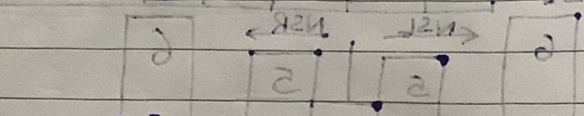
NSL index

width[i] = right[i] - left[i] - 1

Steps

1) NSL index
2) NSR index
3) width
4) area
5) max in area

arr :	6	2	5	4	5	1	6
width :	1	5	1	3	1	7	1
Area :	6	10	5	12	5	7	6



area[i] = arr[i] * width[i]

Code

NSL

```
int pseudoindex = -1;
```

NSR

```
int pseudoindex = -1;
```

⇒

```
for(width[i];){
width[i] = right[i] - left[i] - 1;
}

for(int i = 0 ; i < size ; i++){
    area[i] = width[i] * arr[i];
}
```

area[i] ⇒ max

Max Area Rectangle in Binary Matrix

Revise:

```
MAH(int arr[], int size)
{
    right[] → NSR(arr, size)
    left[] → NSL(arr, size)
    width[i] = right[i] - left[i] - 1
    area[i] = arr[i] * width[i]
    return max in area[i]
}
```

Question:

0	1	1	0
1	1	1	1
1	1	1	1
1	1	0	0

3 * 2 = 6

4 * 2 = 8

4 * 1 = 4

Stages:

2D → 1D

① H1 (1×4)

0	1	1	0
0	1	1	0

GL

② H2 (2×4)

0	1	1	0
1	1	1	1
1	2	2	1

GL

③ H3 (3×4)

0	1	1	0
1	1	1	1
1	1	1	1
2	3	3	2

GL

④ H4 (4×4)

0	1	1	0
1	1	1	1
1	1	1	1
1	1	0	0
3	4	0	0

GL

ans = max(MAH(H1) → 2, MAH(H2) → 4, MAH(H3) → 8, MAH(H4) → 6) → ⑧ Ans

⇒

0	1	1	0	→	0	1	1	0
1	1	1	1	→	1	2	2	1
1	1	1	1	→	2	3	3	2
1	1	0	0	→	3	4	0	0

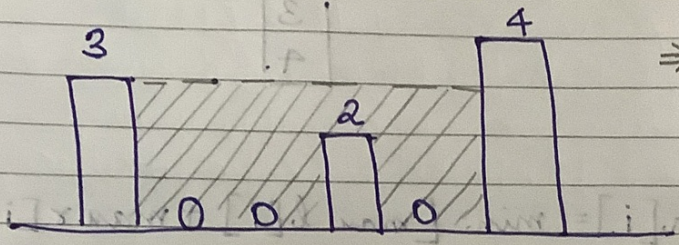
Code :

```
vector <int> v;
for(int j=0 ; j<m ;j++)
        v.push_back(arr[0][j]

int mx = MAH(v);

for(int i=1 ; i<n ; i++){
    for(int j=0; j<m ;j++){
    if (arr[i][j]==0)
            v[j]=0;
    else
        v[j] += arr[i][j]

    mx= max(mx, MAH(v))
    }
}
```

Rain Water Trapping

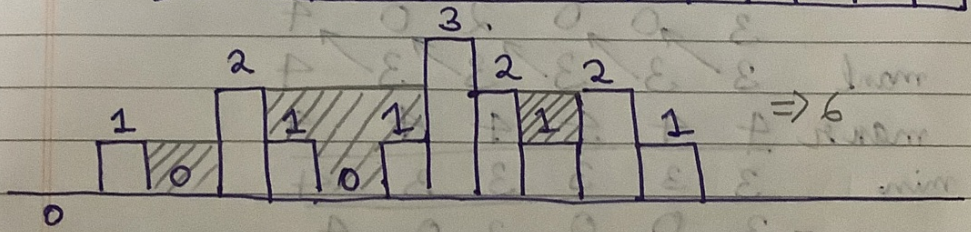
- Min Stack

⇒ arr[]: 3 0 0 2 0 4

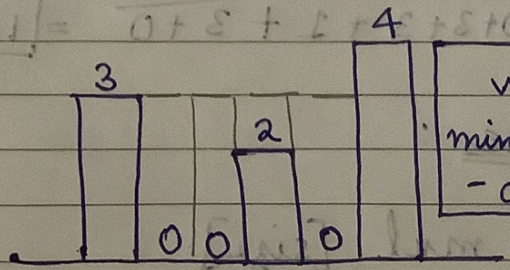


⇒ 10

⇒ arr[]: 0 1 0 2 1 0 1 3 2 1 2 1



⇒ 6



water[i] = min(maxL, maxR) - arr[i]

0 3 3 1 3 = 10

Total Unit of Water = Σ water on each of the building

$\min(3,4) - 2$

$= 3 - 2$

$= 1$

3	0	0	2	0	4

maxl:

maxr: 3, 4

Σ water[i] = min [maxl[i] + maxr[i]] - arr[i]

	3	0	0	2	0	4
maxl	3	3	3	3	3	4
maxr	4	4	4	4	4	4
min	3	3	3	3	3	4
-	3	0	0	2	0	4

water[i] = 0 + 3 + 3 + 1 + 3 + 0 = **10**

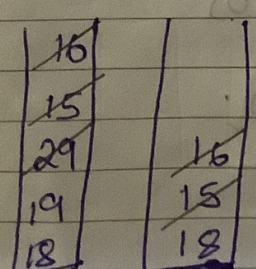
Code :

```
int mxl [size] . . .
int mxr [size]

mxl[0] = arr [0]
for(int i=1 ; i< size ; i++)
    mxl[i] = max(mxl [i-1] , arr[i]);
```

```
mxr[size-1] = arr[size-1];
for(int = n-2; i >=0; i--)
    mxr[i] = max(mxr[i+1], arr[i]);

int water[size];

for(int i=0; i<size; i++)
    water[i] = min[mxl[i], mxr[i]]
                - arr[i];

int sum = 0;
for(int i=0; i<size; i++)
    sum += water[i];
```

Min stack with extra space

18 19 29 15 16



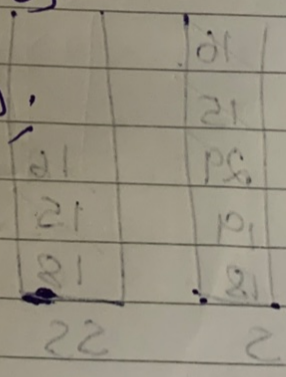
pop()
pop()
pop()

Code

```
stack <int>s;
stack <int>ss;

void push(int a)
{ s.push(a)
   if(ss.size[]==0||a <= ss.top())
      ss.push(a);
}
int pop ()
{ int temp = s.top();
  s.pop()
  if (temp == ss.top())
     ss.pop()
  return temp;
}
int getmin()
{
   if(ss.size()==0)
      return -1
   return ss.top();
}
```

edge condition

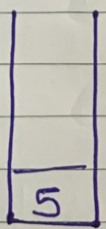
```
if(s.empty())
return -1;
```



Min Stack Implementation in O(1) space

O(1)

variable : int minEle




```
int getMin()
{
if (s.size() == 0)
  return -1
return Min ele;
}
```

push

>= min_element <

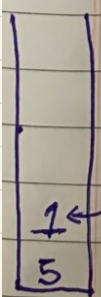
1
3
3
7
5

Min_Ele = ~~5~~ ~~3~~ 1

2x - Min_Ele

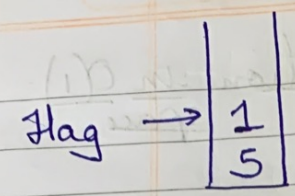
We will push = 3 * 2 - 5

= 6 - 5 = 1



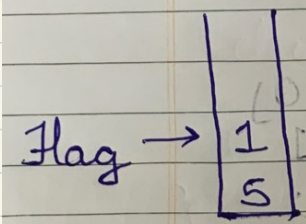
Find the element

3 * 2 - 1 = 5



Min Ele = '3'

New Min Ele = 3*2 - 1
= 6 - 1
= 5



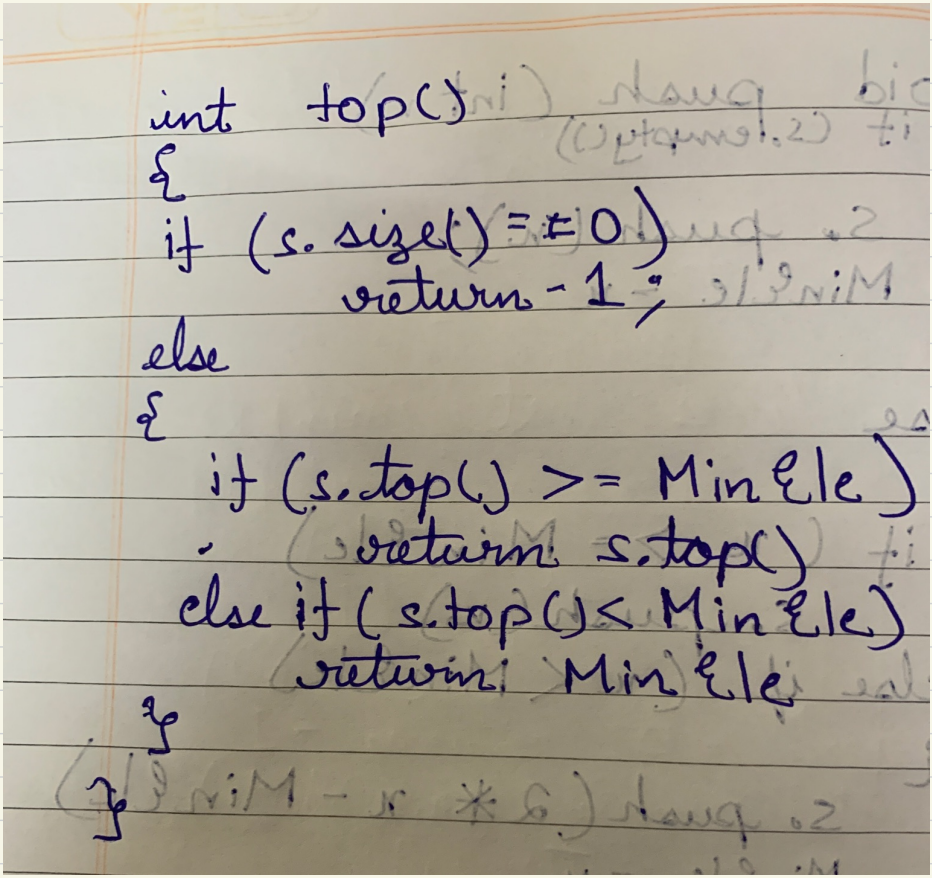
Min Ele = '3'

s.top() ⇒ if s.top() is small than Min Ele then we return Min Ele.

Code

```
int getMin()
{
    if(s.empty())
        return -1
    return MinEle;
}
```

```
void push (int x)
{ if (s.empty())
    s.push(x);
    MinEle = x
}
else
{
   if (x >= MinEle)
     s.push(x)
   else if (x < MinEle)
   {
      s.push(2 * x - MinEle)
      MinEle = x
   }
}

void pop()
{
 if (s.size == 0)
     return -1;
else
{
  if (s.top() >= ME)
       s.pop
else if (s.top() < ME)
    MinEle = 2 * MinEle - s.top()
    s.pop
}
}
```



```
int top()
{
if (s.size() == 0)
        return -1;
else
{
   if (s.top() >= MinEle)
      return s.top()
   else if (s.top() < MinEle)
        return MinEle
}
}
```