Stack

(Last In First Out)
(LIFO)

Identifications:

When in the brute force solution nested for loop is dependent upon the outer for loop.

↓ Optimization.

[Stack]

2. Stack

(Conceptual + Questions and variations)

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

(5.) Stock span problem.
(6.) maximum 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 parantheis.
(13.) Iterative Tower Of Hanoi.

2.

Nearest Greater to right (NGR)
or
Next Largest Element

Given an array,

arr[] = [1, 3, 2, 4]

O/p ⇒ [3, 4, 4, -1]

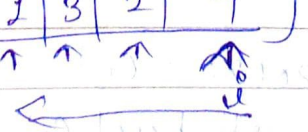
1) Brute force ⇒

for i = 0 ⟶ n-1
  for j = i+1 ⟶ n.

So here we can see loops one dependent on each other, hence we can think of about stack.

2) Using Stack ⇒

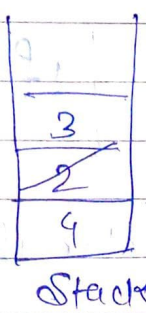
arr = [1 | 3 | 2 | 4]



res = [-2, 4, 4, 3]

reverse(res)

res = [3, 4, 4, -1]



Stack

Conditions

operation	res[]
(1) stack empty = -1	
(2) st.top() > arr[i] =	st.top()
(3) st.top() <= arr[i] → pop	• stack empty • st.top() greater than arr[i]

CODE

```
vector<int> res;
stack< int> st;

for( int i = arr.size()-1;  i >= 0;   i--)
{
```

cond1 ←
```
if (st.size() == 0)
    res.push_back(-1);
```

cond2 ←
```
else if(st.size() > 0  && st.top() > arr[i])
    res.push_back (st.top());
```

Cond3 ←

```
else if(st.size()>0 && st.top()<=arr[i])
{
```

popping out the smaller element ←

```
    while(st.size>0 && st.top()<=arr[i])
    {
        st.pop();
    }
```

```
if(st.size()==0)
    res.push_back(-1);

else
    res.push_back(st.top());
```

pushing it on stack →

```
    st.push(arr[i]);
}
```

reversing the array ←

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

Answer ←

```
return res;
```

```
}
```

3.

Nearest Greater to left (NGL)

minor variation of previous solution.

(1) Iterate from left → right.

(2) No need to reverse the vector at the end.

CODE

```
vector<int> res;
stack<int> st;

for(int i=0; i<arr.size(); i++)
{

    if(st.size()==0)
        res.push_back(-1);

else if(st.size()>0 && st.top()>arr[i])

    res.push_back(st.top());
```

```
elseif( st.size()>0 && st.top()<=arr[i])
{

    while( st.size()>0 && st.top()<=arr[i])
        {
            st.pop();
        }

    if(st.size()==0)
        res.push_back(-1);

    else
        res.push_back(st.top());

}

st.push(arr[i]);

}

return res;
```

Given arr

arr[] = | 1 | 3 | 2 | 4 |

o/p is | -1 | -1 | 3 | -1 |

res

4.

Nearest Smaller to left (NSL)

or

Next Smallest Element.

Given an array

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

o/p => | -1 | 4 | -1 | 2 | 2 |

CODE

```
vector<int> res;
stack<int> st;

for(int i=0; i<arr.size(); i++)
{

    if(st.size() == 0)
        res.push_back(-1);

else if(st.size() > 0 && st.top() < arr[i])

    res.push_back(st.top());
```

```
else if(st.size()>0 && st.top() >= arr[i])
{

    while(st.size()>0 && st.top() >= arr[i])
    {
        st.pop();
    }

  if(st.size()==0)
    res.push_back(-1);

  else
    res.push_back(st.top());

}

st.push(arr[i]);

}

return res;
```

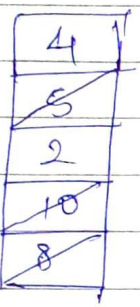
5.

Nearest Smaller to Right (NSR)

Given an array:

arr[] ⇒ | 4 | 5 | 2 | 10 | 8 |

O/P | 2 | 2 | -1 | 8 | -1 |



Stack

| -1 | 8 | -1 | 2 | 2 |

res

reverse

| 2 | 2 | -1 | 8 | -1 |

CODE

```
vector<int> res;
stack<int> st;

for(int i = arr.size()-1; i >= 0; i--)
{

  if (st.size() == 0)
      res.push_back(-1);
```

```
else if(st.size()>0 && st.top()<arr[i])
 {
    res.push_back(st.top());
 }

elseif(st.size()>0 && st.top()>=arr[i])
{
    while(st.size()>0 && st.top()>=arr[i])
   {
      st.pop();
   }

   if (st.size()==0)
      res.push_back(-1);

   else
      res.push_back(st.top());
 }

 st.push(arr[i]);
}

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

 return res;
```

6. **Stock Span Problem**

Given an array.

arr[] =

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

o/p =

1	1	1	2	1	4	6

Cond.) Consecutive smaller or equal element before it.

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

↓ NGL (Index)

res)

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

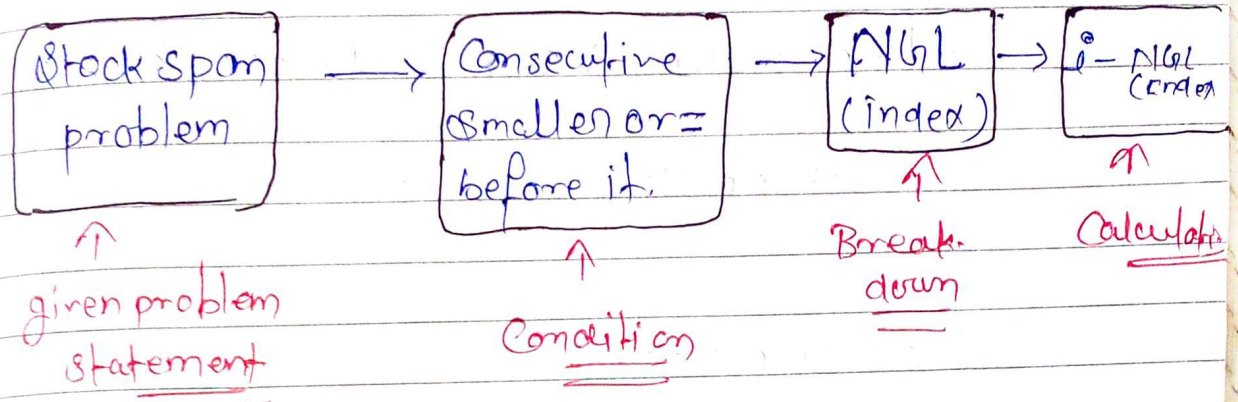
↓ res[i] = i - res(i)

res)

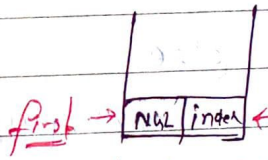
1	1	1	2	1	4	6

⇒ o/p

Sum up =>



CODE



stack< pair<int,int>> st;

```
vector<int> res;
stack< pair<int,int> st;

for(int i=0; i<arr.size(); i++)
{
    if(st.size()==0)
        res.push_back(-1);

    else if(st.size()>0 && st.top().first > arr[i])
        res.push_back(st.top().second);
```

```
else if (st.size() > 0 && st.top().first <= arr[i])
{
    while(st.size() > 0 && st.top().first <= arr[i])
    {
        st.pop();
    }

    if (st.size() == 0)
        res.push_back(-1);

    else
        res.push_back(st.top().second);
}

st.push({arr[i], i});
}

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

return res;
```

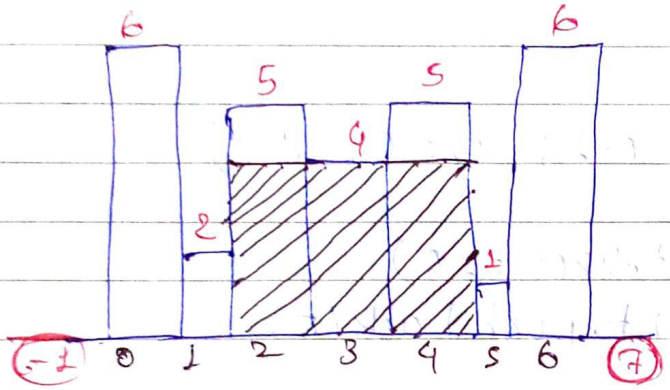
7.

Maximum Area Histogram.
(MAH)

Given an array of height:

O/P ⇒ 12

arr[] = | 6 | 2 | 5 | 4 | 5 | 1 | 6 | → Height.



max area = 4 × 3 = 12

Solution: steps ⇒

Arrays ⇒

(1.) NSR(index)

(2.) NSL.(index)

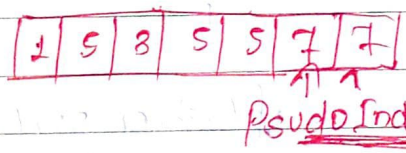
(3.) Width ⇒ NSR(index) − NSL(index) − 1

(4.) Area ⇒ (arr[i] * width[i]) max

CODE

```
vector<int> nsr(int arr[], int n)
{

    vector<int> right;
  stack<pair<int,int>> st;

  for(int i= n-1; i>=0; i--)
  {
      if(st.size()==0)
       right.push_back(n);

  else if(!st.empty() && st.top().first<arr[i])
  {
      right.push_back(st.top().second);
  }

  else
  {
      while(!st.empty() && st.top().first>=arr[i])
       {
          st.pop();
       }
```

```
        if( st.empty() )
            right.push_back(n);

        else
            right.push_back( st.top().second );

    }

    st.push( {arr[i], i} );

}

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

return right;
```




```
}

vector<int> nsl(int arr[], int n)
{
    vector<int> left;
    stack<pair<int, int>> st;

for( int i=0;  i<n;  i++)
{
    if( st.size()==0)
        left.push_back(-1);
```

```
else if( !st.empty() && st.top().first < arr[i])
{
    left.push_back(st.top().second);
}

else
{
  while(!st.empty() && st.top().first >= arr[i])
  {
      st.pop();
  }

  if(st.empty())
     left.push_back(-1);

  else
     left.push_back(st.top().second);
}

  st.push({arr[i], i});
}

return left;
}
```

→ | -1 | -1 | 2 | 1 | 3 | -1 | 5 |

```
int main()
{
    arr[n] = {6, 2, 5, 4, 5, 1, 6};
vector<int> right, left, width, area;

right = nsr(arr, n);

left = nsl(arr, n);
```

width → | 1 | 4 | 2 | 3 | 2 | 9 | 2 |

Calculating width

```
for (int i=0; i<n; i++)
{
    width[i] = right[i] - left[i] - 1;
}
```

area: | 6 | 8 | 5 | 12 | 5 | 7 | 6 |

Calculating area

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

finding max area

```
cout << *max_element(area.begin(), area.end());
```

//12

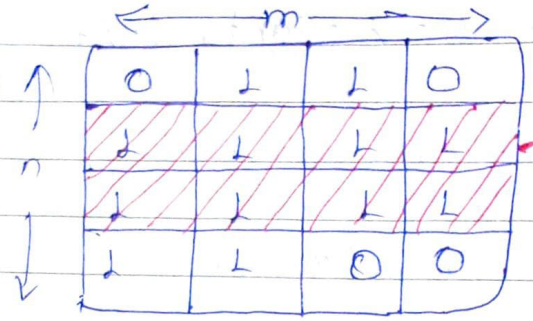
```
return 0;
}
```

8.

Max Area Rectangle In binary matrix.

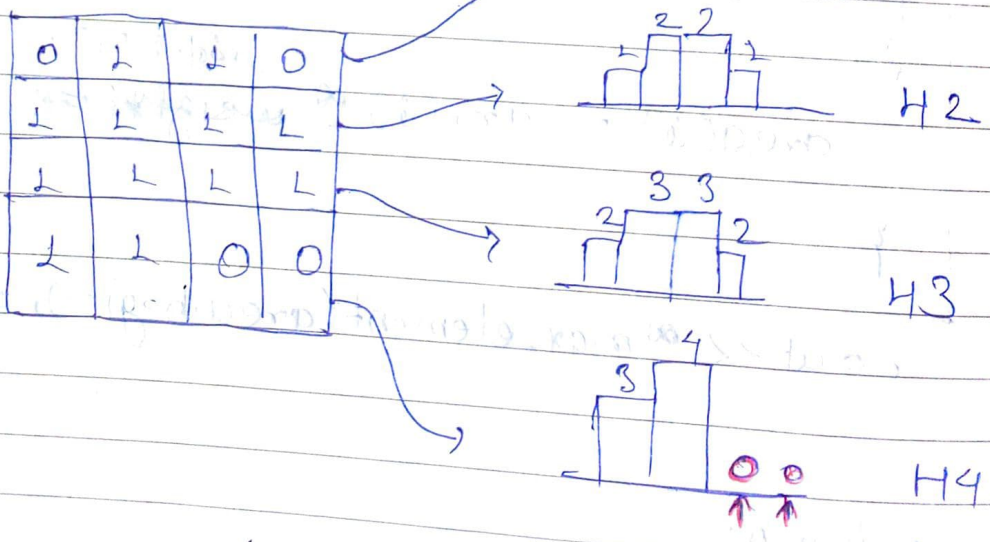
Given a 2D matrix

arr[4][4] = arr[n][m]



→ 2 × 4 = 8

O/P → 8



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

CODE

```
vector<int> v;

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

→ storing first row in vector

maximum element in first row →

```
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] = v[j] + arr[i][j];
}
mx = max(mx, MAH(v));
}
return mx;
```

MAH ← from every row

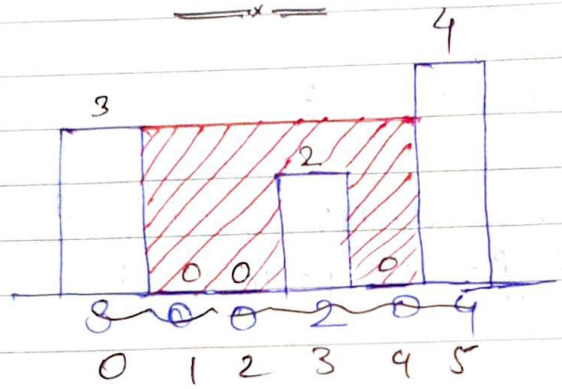
Q. **Rain Water Trapping**

I/P

arr[] = [3 | 0 | 0 | 2 | 0 | 4]

O/P

10



At every building we have to check two things:

(1.) (2.) (max. building to left. / max. building to right.) min − arr[i]

	3	0	0	2	0	4
mxl[]	3	3	3	3	3	4
mxr[]	4	4	4	4	4	4
min:	3	3	3	3	3	4
− arr[i]:	3	0	0	2	0	4
sum:	0	3	3	1	3	0

⇒ 10

CODE

```
int n= arr.size();
int mxl[n];
int mxr[n];
```

Initializing mxl with first building

```
mxl[0] = arr[0];
```

// mxl of every building

```
for(int i=1; i<n; i++)
{
    mxl[i] = max( mxl[i-1], arr[i] );
}
```

```
mxr[n-1] = arr[n-1];
```

Initializing mxr with the last building

mxr of every building

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

Calculating water at every building

```
int water[n];

for(int i = 0;  i<n;  i++)
{
    water[i] = min (mxl[i] , mxr[i]) - arr[i];
}
```

total water

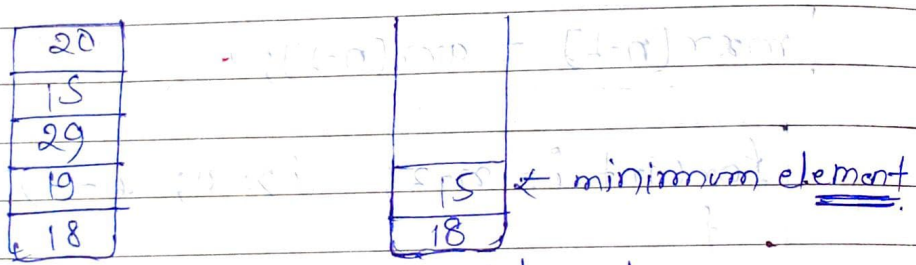
```
int sum= 0
for(int i=0; i<n; i++)
    sum += water[i];

return sum;
```

10. # Minimum Element In Stack With Extra Space.

Given,

arr[] : | 18 | 19 | 29 | 15 | 20 |

O/p → min. element each time.



Stack (S) | Supporting stack (SS)

Code

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

```
int getMin()
{
    if(ss.size()==0)
        return -1;
    return ss.top();
}
```

→ It returns min element

```
void push(int a)
{
    s.push(a);
    if(ss.size()==0 || a<=ss.top())
    {
        ss.push(a);
    }
    return;
}
```

→ push operation

```
int pop()
{
    if(s.size()==0)
        return -1;

    int ans = s.top();
    s.pop();

    if(ss.top()==ans)
        ss.pop();

    return ans;
}
```

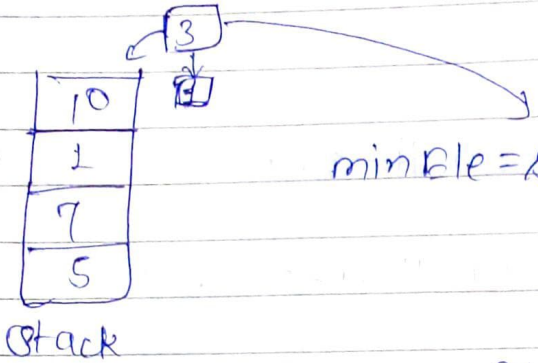
→ pop operation

#1.

Minimum Number in Stack

O(1) space

5	7	3	10



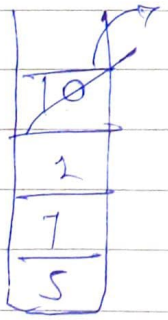
pushing ⇒

minEle = ~~5~~ 3

```
if (x < minEle)
    s.push(2*x - minEle)
```

2*3 - 5 ⇒ 6 - 5 ⇒ 1

pop ⇒




```
if (s.top < minEle)
```

CODE

```
void push(int x)
{
    if(s.size() == 0)
    {
        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() >= minEle)
            s.pop();

      else if(s.top() < minEle)
      { 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;
}
```