(Monotonic stack / queue)

notonic stack :- Which specify some order

Greater Element :-

12	5	3	1	2	5	3	1	2	4	6	I/P
-1	6	5	2	5	6	4	2	4	6	-1	O/P

pt:- Decreasing order

le:-

```
<int> find NGE (arr[])
{  nge[n], stack st
   for(i=n-1→0)
   {
      while(!st.empty() && st.top<=arr[i])
            st.pop()
      if (st.empty())  nge[i] = -1
      else   nge[i] = st.top()
      st.push(arr[i])
   }
return nge
```

TC:- O(2N)

SC:- O(N)

v) Next Greater element - v :- (Circular)

arr [2, 10, 12, 1, 11] → [10, 12, -1, 11, 12]

→ Brute :-

arr = [2,10,12,1,11] [2,10 12 1 11]
0 1 2 3 4 5 6 7 8 9
→ hypothetical copy

⇒ nge[n]

```
for (i=0 → n-1) {
    for(j=i+1 → i+N-1)
    { ind = j % N
      if(arr[ind] > arr[j]) {
          nge[i] = arr[ind]
          break }
    }
}
```

TC :- O(N²)
SC :- O(N)

→ Optimal :-

```
for (i=n-1 → 0) {
    While (!st.empty() && st.top() <= arr[i%N]) {
        st.pop(); }
    if(i<N) → real array
    {
      nge[i] = st.empty() ? -1 : st.top()
    }
    st.push(arr[i%N]) → Push every element real or hypothetical
}
return nge
```

due to hypo array
TC :- O(4N
SC :- O(2

smaller Element :-

arr = [4, 5, 2, 10, 8] → [-1, 4, -1, 2, 2]

ose increasing order

ute :-

ge[n]

r (i=0 → n-1) {

for (j=i-1 → 0) {

if (arr[j] < arr[i])

nge[i] = arr[j], break

}

TC :- O(N²)

SC :- O(N)

urn nge

imal :-

(i=0 → n-1) {

while (!st.empty() && st.top() >= arr[i])

{

st.pop();

}

nge[i] = st.empty ? -1 : st.top()

st.push(arr[i]);

}

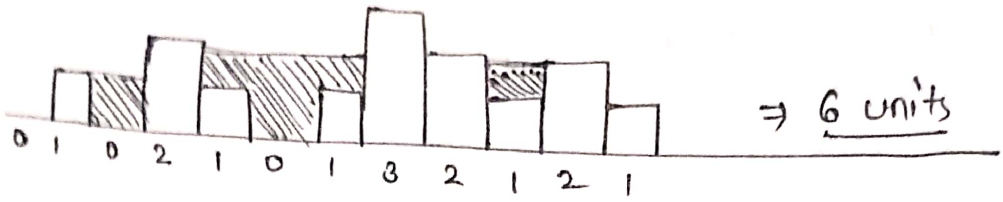
TC :- O(2N)

SC :- O(N)

turn nge;

4) Trapping Rain water :-

arr = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]



⇒ 6 units

⇒ By Prefix sum Arrays (left max / right max) :-

$$\sum_{i=0}^{n-1} \min(\text{leftmax} - \text{rightmax}) - arr[i]$$

```
* total=0

for (i=0 ⟶ n-1) {  ——— O(N)
    leftmax = PrefixMax(i)   rightmax = SuffixMax(i)
    if (arr[i] < leftmax && arr[i] < rightmax)
        total += min(leftmax, rightmax) - arr[i];
}

return total
```

⟶ Prefix Max :-

arr = [2, 1, 0, 5, 3]

prefix = [2, 2, 2, 5, 5]

TC :- ≤ O(3N)

SC :- ≤ O(2N)

```
prefix[0] = arr[0]

for (i=1 ⟶ n-1) {  ——— O(N)
    prefix[i] = max(prefix[i-1], arr[i])
}
```

arr[] = [1, 11, 2, 10]

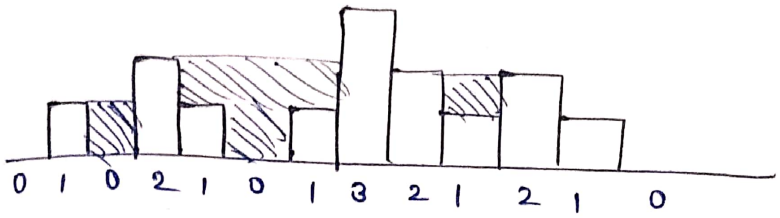
suffix = [11, 11, 10, 10]

suffix[n-1] = arr[n-1] ——— O(N)

for(i=n-2 → 0) {

suffix[i] = max(suffix[i-1], arr[i])

2) Pointers Approach :-



lmax = rmax = total = 0 l=0, r=n-1

while(l<r) { ——— O(N)

if [arr[l] <= arr[r]) {

if (lmax > arr[l])

total += lmax - arr[l];

else

lmax = arr[l]

l = l+1

}

else {

if(rmax > arr[r])

total += rmax - arr[r]

else

rmax = arr[r]

r = r-1

}

TC :- ≤ O(N)

SC :- ≤ O(1)