Today's agenda

- Prefix Sum.
- Sum in a range for multiple query.
- Equilibrium index.
- Product of array except itself
- number of odd numbers in given range. ← easy ← HW
- Rain water trapping → Hard leetcode

Q) Given N array elements, return pf[] where pf[i] = Sum{arr[0], arr[1], ... arr[i]}, for all i.

Ex: arr[5] = { 4 1 6 -2 7 }
(0 1 2 3 4)

pf[5] = { 4 5 11 9 16 }

arr[10] = { -2 5 1 3 4 1 7 -8 2 0 }
(0 1 2 3 4 5 6 7 8 9)

pf[10] = { -2 3 4 7 11 12 19 11 13 13 }

// Brute force

↳ for every index i, run a loop from 0th to ith index and add all the elements.

arr[5] = { 4 1 6 -2 7 }
(0 1 2 3 4) ← i

pf[5] = { 4 5 11 9 16 }

//Pseudo Code

```
int[] PrefixSum (int arr[N]){
    int Pf[N];
    for (int i=0; i<N; i++){
        int Sum=0;
        for (int j=0; j<=i; j++){
            Sum = Sum + arr[j];
        }
        Pf[i] = Sum;
    }
    return Pf;
}
```

T.C: $O(N^2)$

S.C: $O(1)$

i

arr[5] = { 4 1 6 -2 7 }

Pf[5] : 4 5 11

Sum: 0+4+1+6

//Optimal approach

↳ arr[N]

pf[0] = arr[0]

pf[1] = arr[0] + arr[1] = pf[0] + arr[1]
↳ pf[0]

pf[2] = arr[0] + arr[1] + arr[2] = pf[1] + arr[2]
↳ pf[1]

pf[3] = arr[0] + arr[1] + arr[2] + arr[3]
↳ pf[2]
= pf[2] + arr[3]

∴ i

arr[5] : { 4 1 6 -2 7 }
0 1 2 3 4

pf[5] : { 4 5 11 9 16 }

pf[i] = pf[i-1] + arr[i] for all i

//Psuedo Code

```
int[] PrefixOptimal(int arr[N]){
    int pf[N];
    pf[0] = arr[0];

    for(int i=1; i<N; i++){
        pf[i] = pf[i-1] + arr[i];
    }

    return pf;
}
```

T.C: O(N)

S.C: O(1)

Q) Given N array elements and Q queries on an array. for each query calculate sum of all elements in given range.

Ex: arr[10] = | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| -2 | 5 | 1 | 3 | 4 | 1 | 7 | -8 | 2 | 0 |

Q = 5

i	0	1	Sum
0	3	8	9
1	2	4	8
2	0	3	7
3	5	9	
4	6	6	

//Ideal (Brute force)

Iterate in the range for every query.

```
void SumQuery(int arr[N], int queries[Q][2]){
    for(int i=0; i<Q; i++){
        int L = Queries[i][0];
        int R = Queries[i][1];

        int Sum = 0;
        for(int j=L; j<=R; j++){
            Sum = Sum + arr[j];
        }

        Print(Sum);
    }
}
```

T.C : O(Q*N)

S.C : O(1)

Optimal approach

	0	1	2	3	4	5	6	7	8	9
arr[10]:	-2	5	1	3	4	1	7	-8	2	0
pf[10]:	-2	3	4	7	11	12	19	11	13	13

Sum(3, 8) = pf[8] - pf[2]
= Sum(0, 8) - Sum(0, 2)

Sum(2, 4) = pf[4] - pf[1]
= Sum(0, 4) - Sum(0, 1)

Sum(0, 3) = pf[3]
= Sum(0, 3)

Sum(L, R) = pf[R] - pf[L-1] if (L > 0)
= Sum(0, R) - Sum(0, L-1)

Sum(0, R) = pf[R] if (L == 0)
= Sum(0, R)

//Pseudo code

```
void SumQuery(int arr[N], int queries[Q][2]){
    int pf[] = PrefixOptimal(arr);
    for(int i=0; i<Q; i++){
        int L = Queries[i][0];
        int R = Queries[i][1];
        if (L>0) {
            Print(pf[R] - pf[L-1]);
        }
        else{ Print(pf[R]); }
    }
}
```

T.C : $O(N) + O(Q)$
$\simeq O(N+Q)$

S.C : $O(N)$

$O(N*Q)$ | $O(N+Q)$

$N == Q$

$O(N^2)$ | $O(N+N) = O(2N) \simeq O(N)$

Q) Equilibrium Index (pivot)

↳ Given N array elements, count no. of equilibrium index. An index i is said to be equilibrium index if?

Sum of all the elements before ith index = Sum of all the elements after ith index.

Sum(0, i-1) = Sum(i+1, N-1)

Ex: arr[4]:

	0	1	2	3
	-2	6	3	4
leftSum:	0	-2	4	7
rightSum:	13	7	4	0

⇒ ans = 1

arr[7]:

	0	1	2	3	4	5	6
	-7	1	5	2	-4	3	0
leftSum:	0	-7	-6	-1	1	-3	0
rightSum:	7	6	1	-1	3	0	0

⇒ ans: 2

//Optimal approach

	0	1	2	3 (N-1)
arr[4]:	-2	6	3	4
pf[4]:	-2	4	7	11

leftSum: Sum(0, i-1) = pf[i-1];

rightSum: Sum(i+1, N-1) = pf[N-1] - pf[i]
= pf[N-1] - pf[i]

//Pseudo code

```
int pivotIdx(int arr[N]){
    int pf[] = Prefix(arr);
    int ans = 0;

    for(int i=1; i<N-1; i++){
        int leftSum = pf[i-1];
        int rightSum = pf[N-1] - pf[i];
        if(leftSum == rightSum){ ans++; }
    }
```

T.C: O(N)+O(N)
O(2N) ≈ O(N)
S.C: O(N)

```
    //0th index Check
    if (0 == p[n-1] - p[0]){
        ans++;
    }

    //last index Check
    if (p[n-2] == 0) {ans++;}

    return ans;
}
```

Q) Product of Array Except itself

↳ Given arr[N], return an array answer such that answer[i] is equal to the product of all the elements of nums except nums[i].

Note: you can't use division operation.

Ex: arr[4]: { 1 2 3 4 }
(0 1 2 3)

ans[4]: 24 12 8 6

PrefixProd[i] = { arr[0] * arr[1] ... * arr[i] }

↳ arr[4]: { 1 2 3 4 }
(0 1 2 3) — i

PrefProd[4]: { ~~1~~ ~~2~~ ~~6~~ ~~24~~ }
24 12 8 6

Suffix = 1*4 = 4*3
= 12*2
= 24

PrefProd[i] = PrefProd[i-1] * Suffix

PrefProd[3] = PrefProd[2] * 1;
PrefProd[2] = PrefProd[1] * 4;
PrefProd[1] = PrefProd[0] * 12;

//Pseudo code

```
int[] ProductExceptItself (int arr[n]){
    int PrefProduct[n];
    PrefProduct[0] = arr[0];
    for (int i=1; i<n; i++){
        PrefProduct[i] = PrefProduct[i-1] * arr[i];
    }
    int Suffix = 1;
    for (int i=n-1; i>0; i--){
        PrefProduct[i] = PrefProduct[i-1] * Suffix;
        Suffix = Suffix * arr[i];
    }
    PrefProduct[0] = Suffix;
    return PrefProduct;
}
```

T.C: O(2n)

S.C: O(1)

Tracing

```
int PofProduct[n];
PofProduct[0] = arr[0];
for(int i=1; i<n; i++){
    PofProduct[i] = PofProduct[i-1] * arr[i];
}
int suffix = 1;
for(int i=n-1; i>0; i--){
    PofProduct[i] = PofProduct[i-1] * suffix;
    suffix = suffix * arr[i];
}
PofProduct[0] = suffix;
```

i: 0 1 2 3

arr[4]: { 1 2 3 4 }

PofProduct[4]: { ~~1~~ ~~2~~ ~~6~~ ~~24~~ }
24 12 8 6

suffix = 1 * 4 = 4 * 3 = 12 * 2 = 24

Prefixmax and Suffixmax

Prefixmax

Ex: arr[12]:

0	1	2	3	4	5	6	7	8	9	10	11
2	1	3	2	1	2	4	3	2	1	3	1

pmax[]:

2	2	3	3	3	3	4	4	4	4	4	4

Prefixmax[i] = max(0, ..., i)

Suffixmax

Suffixmax[i] = max(i, N-1)

arr[12]:

0	1	2	3	4	5	6	7	8	9	10	11
2	1	3	2	1	2	4	3	2	1	3	1

smax[12]:

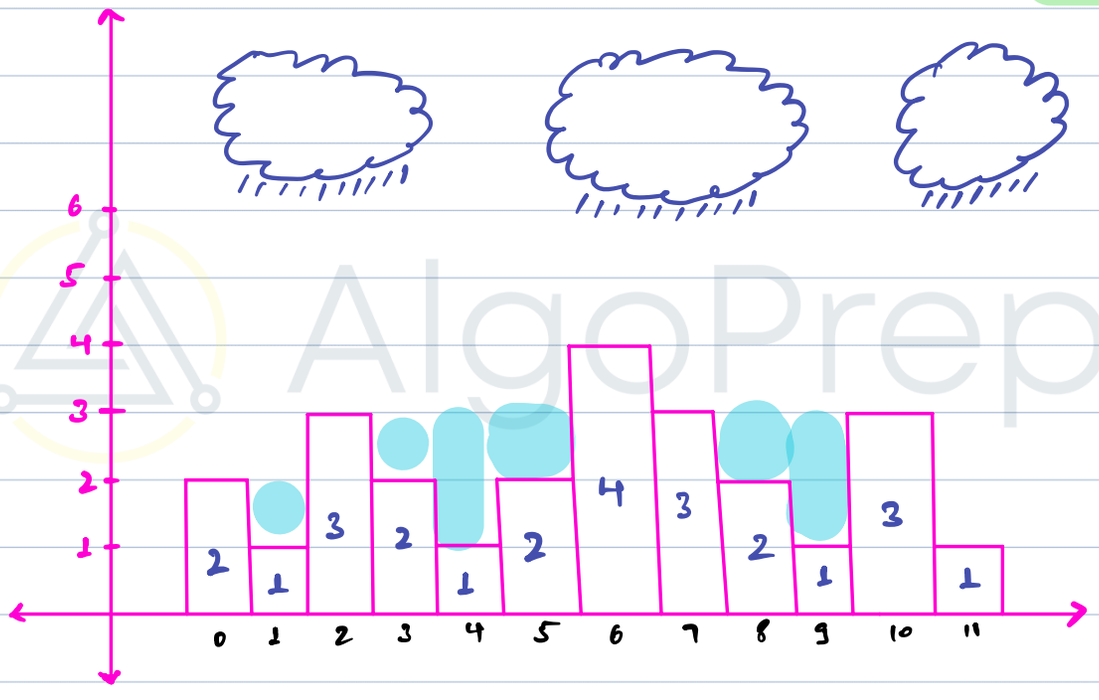
4	4	4	4	4	4	4	3	3	3	3	1

Q) Rain water trapping

↳ Given N non-negative integers representing an elevation map where the width of each bar is 1. Compute how much water it can trap after raining.

Eg: arr[12]: 2 1 3 2 1 2 4 3 2 1 3 1

↳ ans=8

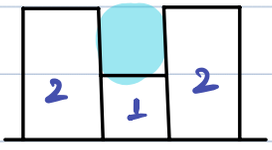


↳ Calculate amount at each wall and add them.

↓

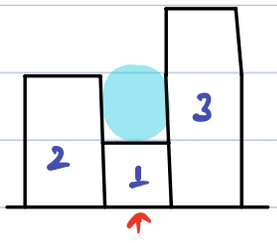
ans

Ex:



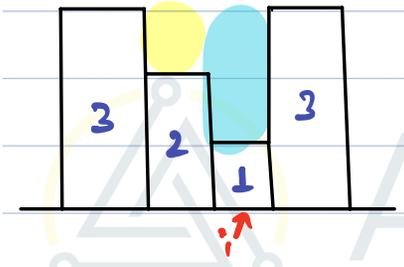
lb = 2
rb = 2

yb = 2
amount = 2 - 1 = 1



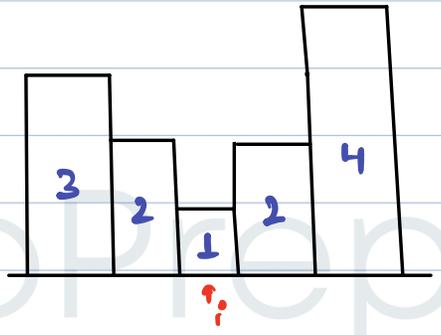
lb = 2
rb = 3

yb = min(lb, rb) = 2
amount = 2 - 1 = 1



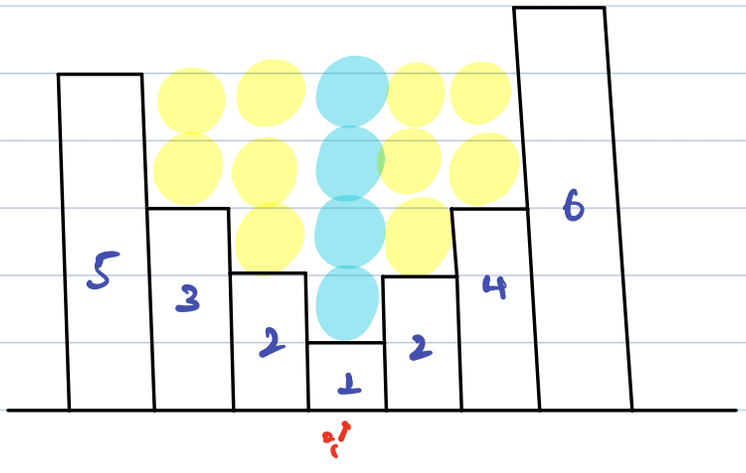
rb = 3
lb = 3

yb = min(lb, rb) = 3
amount = 3 - 1 = 2



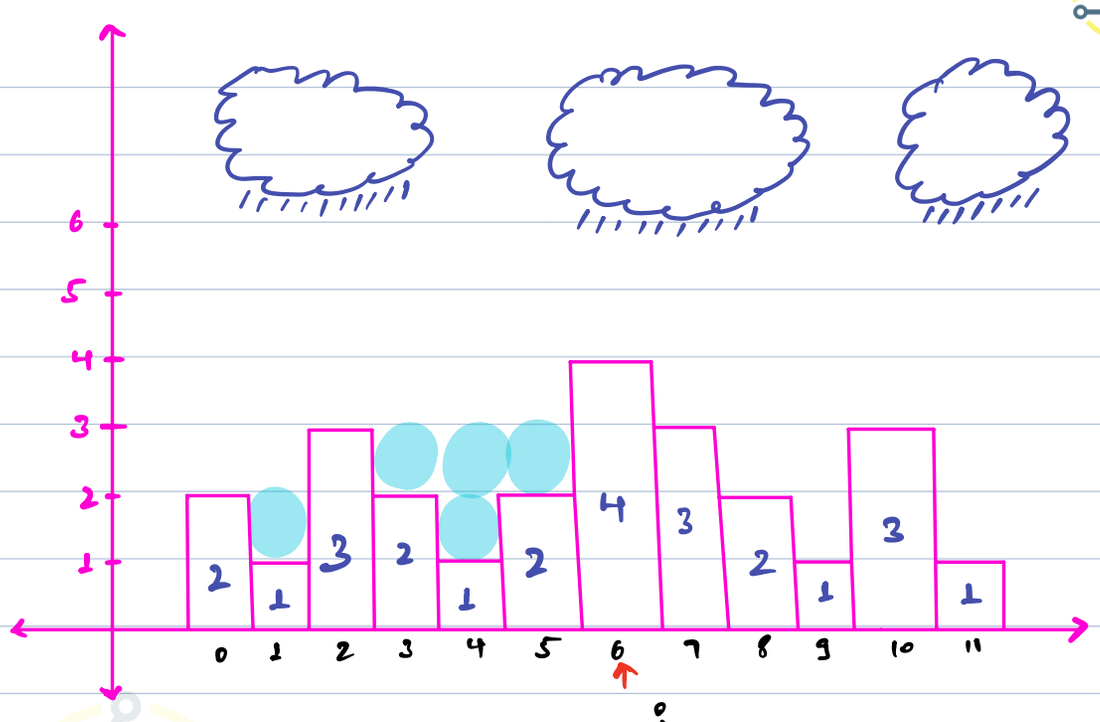
lb = 3
rb = 4

yb = min(lb, rb) = 3
amount = 3 - 1 = 2



lb = max height on left = 5
rb = max height on right = 6

yb = min(lb, rb) = min(5, 6) = 5
amount = 5 - 1 = 4



i

	0	1	2	3	4	5	6	7	8	9	10	11
arr[]:	2	1	3	2	1	2	4	3	2	1	3	1
Pmax[]:	2	2	3	3	3	3	4	4	4	4	4	4
Smax[]:	4	4	4	4	4	4	4	3	3	3	3	1

lb = Pmax[i-1] = 3

rb = Smax[i+1] = 4

yb = min(3,4) = 3

cont = 3 - 2 = 1

amount = 0+1 = 1+0 = 1+1 = 2+2 = 4+1 = 5

// Pseudo code

```
int Rainwater (int arr[N]) {

    int Pmax[] = func1(arr);   // HW
    int Smax[] = func2(arr);

    int amount = 0;
    for(int i = 1; i < N-1; i++) {
        int lb = Pmax[i-1];
        int rb = Smax[i+1];
        int yb = min(lb, rb);
        int conti = yb - arr[i];

        if (conti > 0) {
            amount = amount + conti;
        }
    }

    return amount;
}
```

T.C : O(3*N) : O(N)

S.C : O(2N) : O(N)

↓

O(1)

↓

Two Pointers

```
int[] func2(int arr[N]){

    int[] smax = new int[ N ];
    smax[N-1] = arr[N-1];

    for(int i=N-2 ; i>=0; i--){
        smax[i] = Math.max(smax[i+1],arr[i]);
    }

    return smax;

}
```