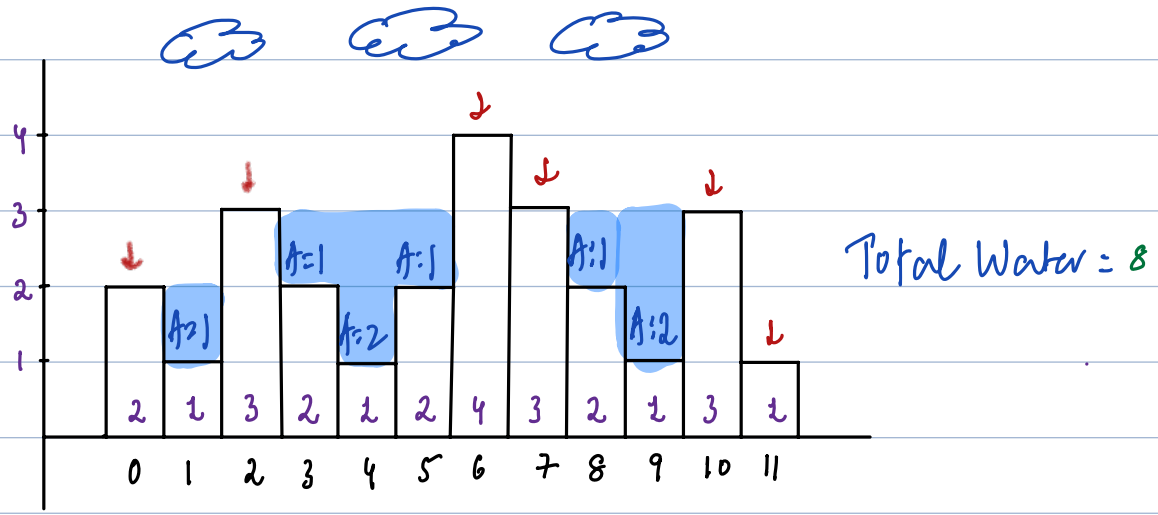
Todays Content

1. Water Filling
2. Product Except current element
3. Majority Element

Given an arr[n] where arr[i] represent height of the building
Return amount of water trapped in all buildings
Note: Width of each building is 1

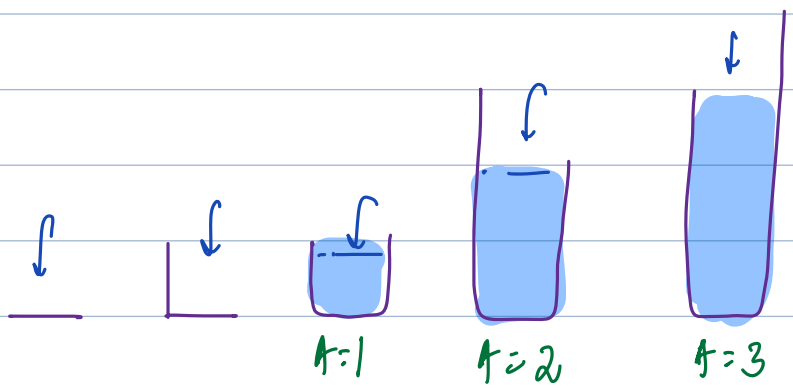
	0	1	2	3	4	5	6	7	8	9	10	11
arr[12] = {	2	1	3	2	1	2	4	3	2	1	3	1 }



Total Water = 8

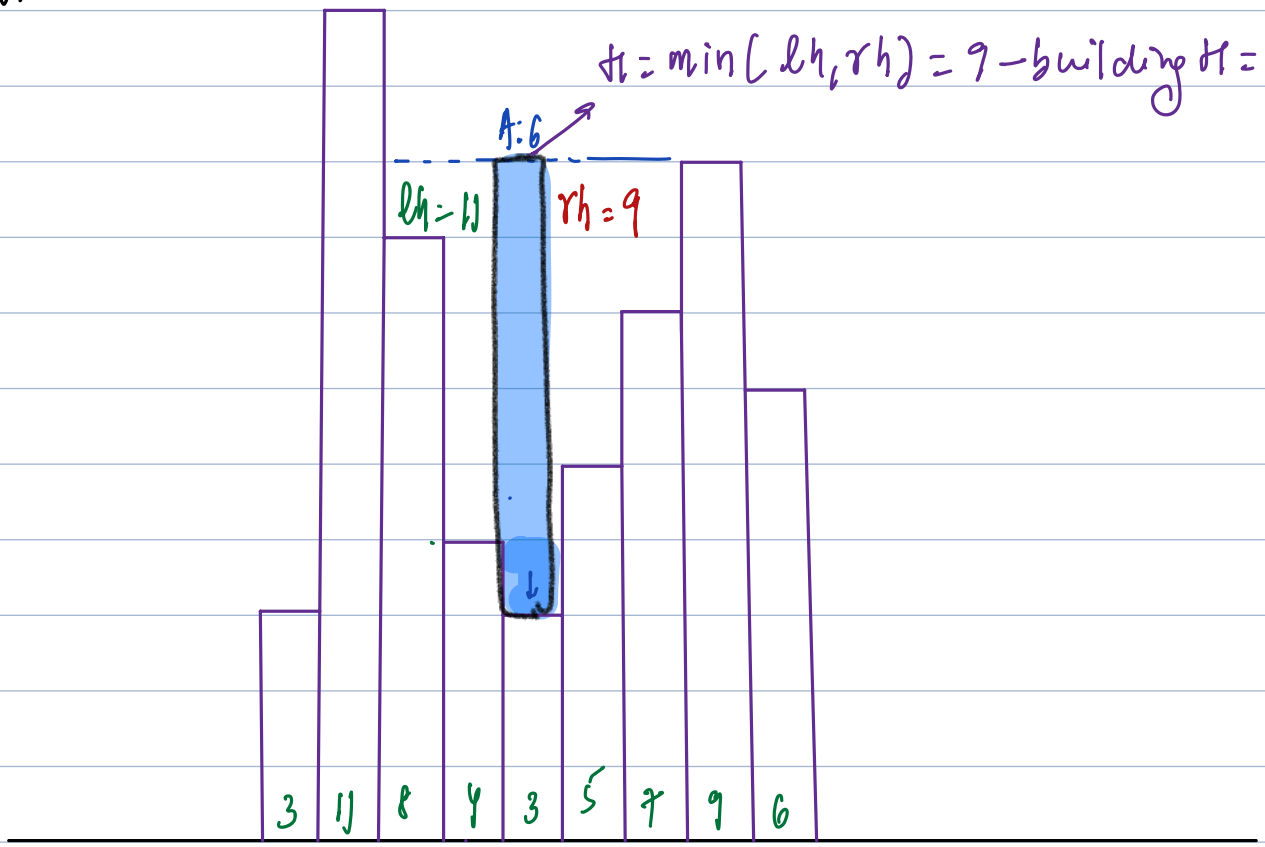
Idea: Add water accumulated on top of each building.

Dry Run 1:



Obs: Water on a building is minimum of (left, right) boundary.

Dry Run 2:



#Idea:

int ans = 0; TC: O(N²) SC: O(1)

For all buildings: Add water accumulated.

Water in i^{th} building

lh = Max height in leftSide [0..i-1]

rh = Max height in rightSide [i+1..n-1]

ht = min(rh, lh)

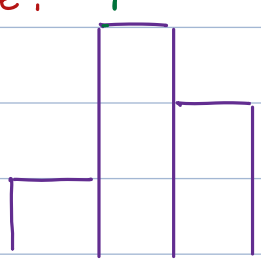
W = ht - height[i]

ans = ans + max(W, 0);

}

return ans;

#EdgeCase:



W = 1 - 3 = -2

If water acculate is -ve, skip it

Optimization: Using Precomputation.

For every element we want, max on left, max on right.

So **precompute**:

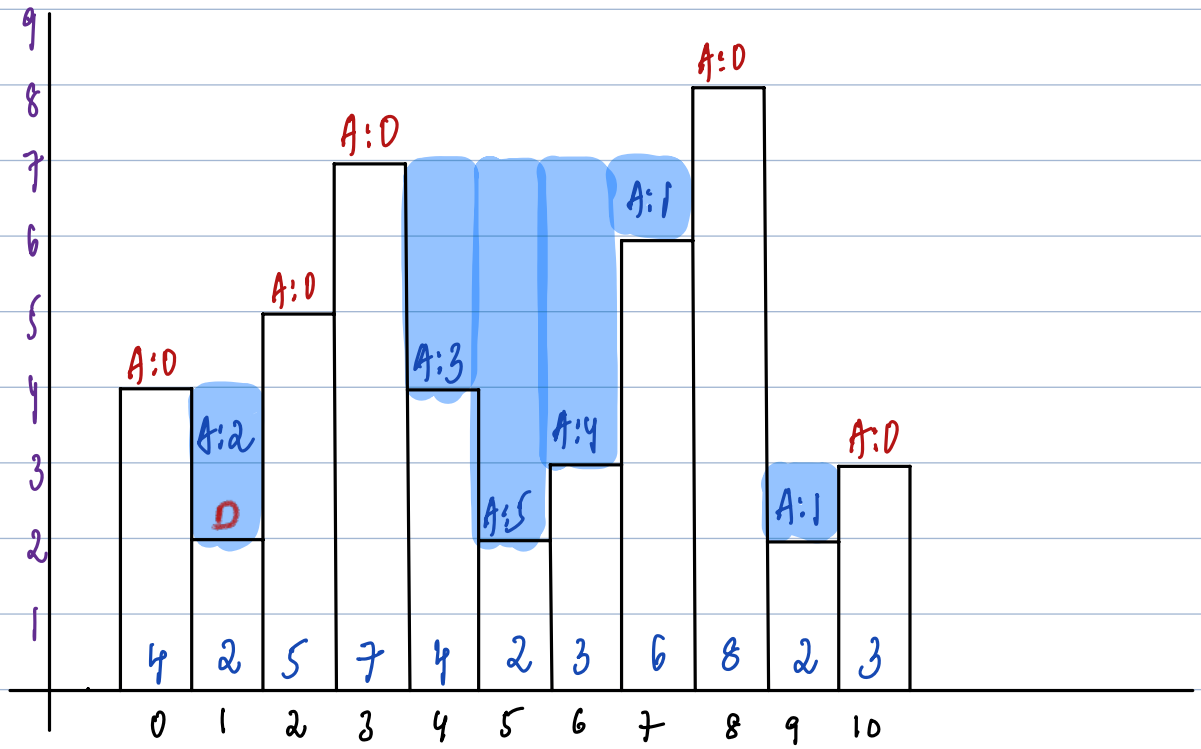
Max on left : PrefixMax

Pf[i] = Max of all elements [0..i]

Max on right: SuffixMax

sf[i] = Max of all elements [i..N-1]

DryRun2:



Pf[] = { 4 4 5 7 7 7 7 7 8 8 8 }

rh[] = { 8 8 8 8 8 8 8 8 8 3 3 }

#Water on i^{th} Building:

lh = max on left [0 i-1] = Pf[i-1]

rh = max on right [i+1..N-1] = sf[i+1]

h = min(lh, rh)

W = h - height[i]

ans = ans + max(W, 0).

Pf[i] = Max of all elements [0..i]

sf[i] = Max of all elements [i..N-1]

```
int water(vector<int> h){   TC: O(N+N+N) = O(N)   SC: O(N+N) = O(N)
    vector<int> leftm = pfmax(h);
    vector<int> rightm = sfmax(h);
    int N = h.size();
    # 0th & N-1th building will not have any water accumulated
    int ans = 0;
    for(int i=1; i<N-1; i++){
        #Water accumulated in h[i];
        int lh = leftm[i-1];     #max from [0..i-1]   # Edge Case i==0
        int rh = rightm[i+1];    #max from [i+1..N-1] # Edge Case i==N-1
        int bh = min(lh,rh);
        int w = bh - h[i];
        ans = ans + max(w,0);
    }
    return ans;
}
```

Product Array Puzzle:

Given a arr[] Construct a prod[]

Such that prod[i] = product of all arr[] except arr[i]

Note: Cannot use / operator.

	0	1	2	3	4
Ex1: arr[] = {	10	3	5	6	2 }
prod[] = {	180	600	360	300	900 }

	0	1	2	3	4	5
Ex2: arr[] = {	3	4	2	5	6	2 }
prod[] = {						}

Ideal

```
For every arr[i]:
    prod = product of all elements from [0..i-1]
    pror = product of all elements from [i+1..n-1]
    prod[i] = prod * pror;
}

return prod;
```

TC: O(N²) SC: O(1)

Optimization: Precomputation.

For every element we want, prod in left & prod in right

So **precompute**:

Prod in left: Prefix Prod

Pf[i] = Product of all elements [0..i]

Prod in right: Suffix Prod

Sf[i] = Product of all elements [i..N-1]

For i^{th} element:

pref = product of elements from [0..i-1] = Pf[i-1]

suff = product of elements from [i+1..N-1] = Sf[i+1]

prod[i] = Pf[i-1] * Sf[i+1];

```
vector<long> productArray(vector<long> ar){

    vector<long> pf = prefixProduct(ar);
    vector<long> sf = suffixProduct(ar);
    int N = ar.size();
    vector<long> prod(N, 0);

    if(N==1){ prod[0]=1; return}

    prod[0] = sf[1]; # product of elements [1..N-1]
    prod[N-1] = pf[N-2]; # product of elements [0..N-2]

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

    return prod;
}
```

ar[] = {10}
pf[] = {1}

Edge Cases: i = 0, i = N-1;