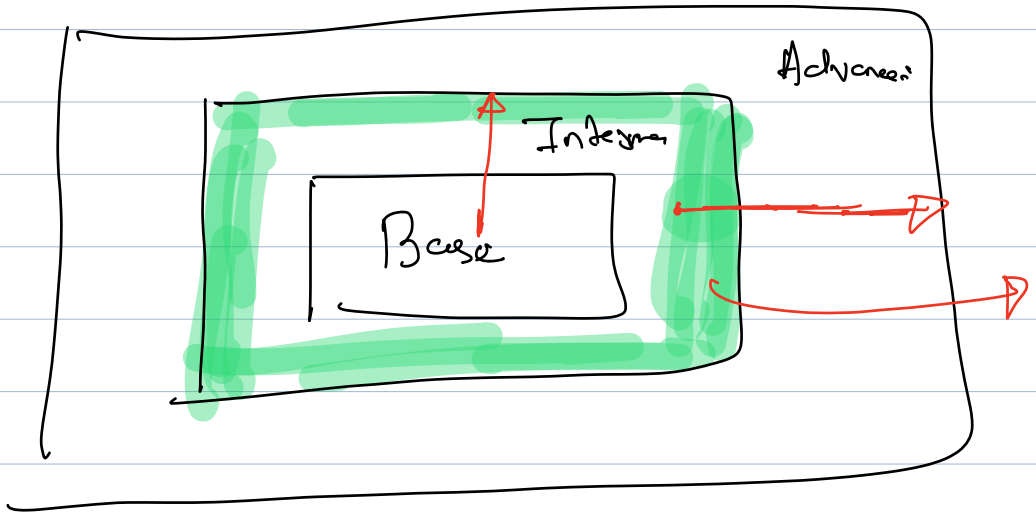
⇒ April intermediate + June Advanced.



Recursion, Bit manipulation.

⇒ Start fresh

⇒

Merge Intervals.

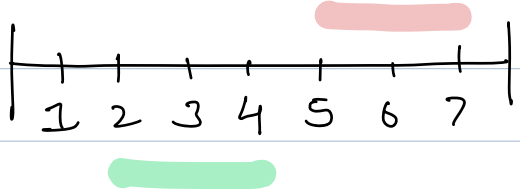
I_1	I_2		Merge Interval
(2, 6)	(3, 7)	⇒	(2, 7)
(2, 8)	(4, 6)	⇒	(2, 8)

(2, 7) → max(end1, end2)

(2, 8) → min(start1, start2)

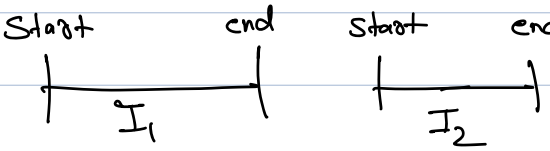
(2, 4) (5, 7) ⇒ can't merge

For merging, intervals should be overlapping!



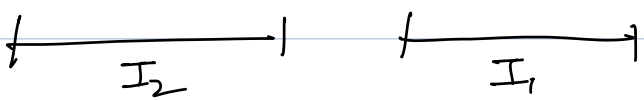
1) Two interval I_1 & I_2.

Case 1:



$I_2.start > I_1.end.$

Case 2:

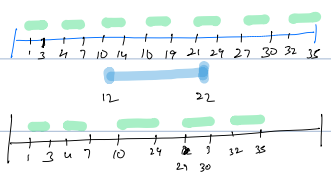


$I_1.start > I_2.end$

Q

N = 7, I =

			New overlapping Intervals.
[1, 3]			[1, 3]
[4, 7]			[4, 7]
[10, 14]	[12, 22]	⇒ [10, 22]	[10, 24]
[16, 19]	[10, 22]	⇒ [10, 22]	[27, 30]
[21, 24]	[10, 22]	⇒ [10, 24]	[32, 35]
[27, 30]	[10, 24]		
[32, 35]			

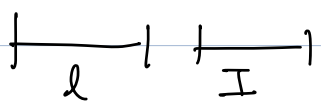


Pseudo Code !

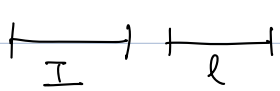
Input ⇒ 1) List<Interval> l | l.s | l.e |
2) Interval I.

List<Interval> ans;

for (int i = 0 ; i < n ; i++) {



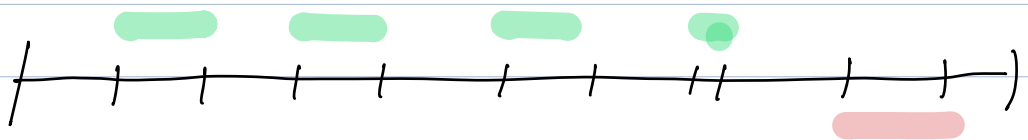
if (I.s > l[i].e) {
ans.push(l[i]); // Case 1
} else if (l[i].s > I.e) {
ans.push(I);



// Case 2 for(int j = i ; j < n ; j++) {
ans.push(l[j]);
}
return ans;
} else {
I.s ⇒ min(I.s, l[i].s); // Case 3
I.e ⇒ max(I.e, l[i].e);
}
}
ans.push(I);
return ans;

TC: O(n)
SC: O(n)

Edge Case



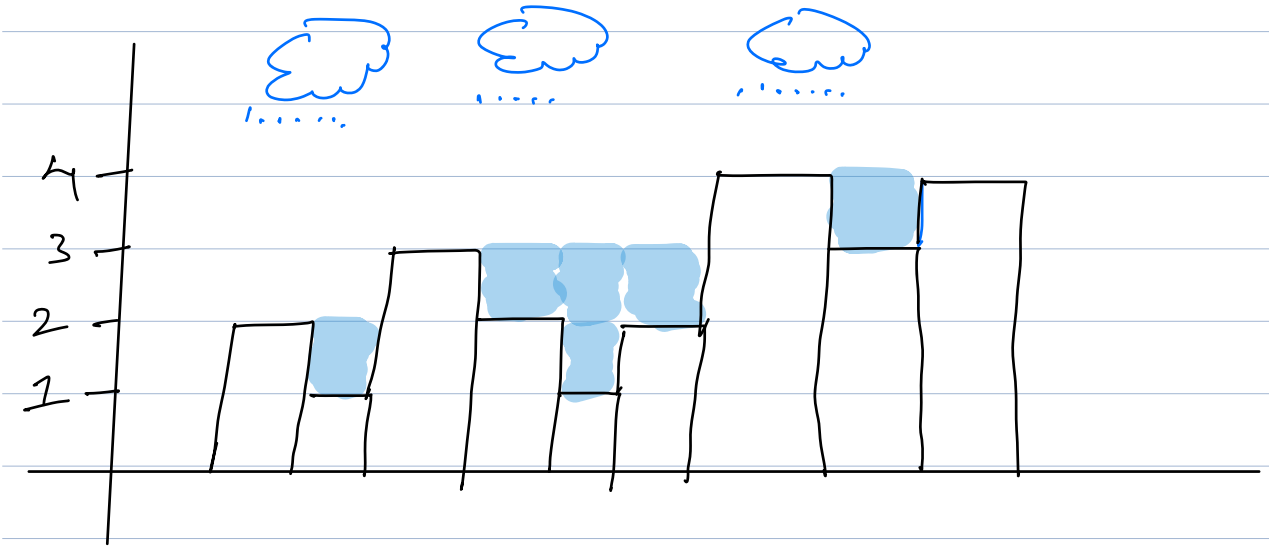
( ) ( ) ( ) ( ) ( )

(I)

Rain water Trapped

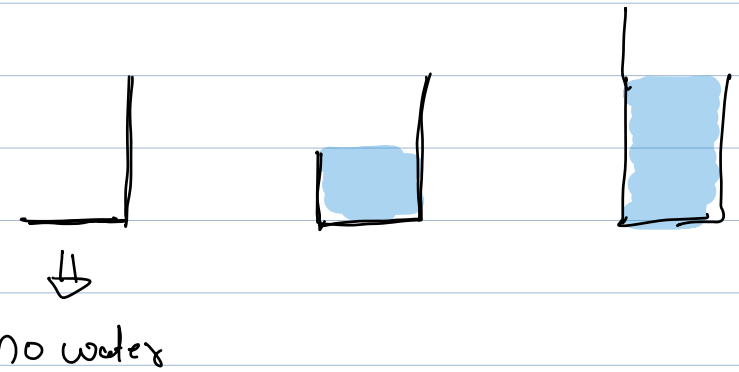
Given n array elements, where arr[i] represents the height of the building. Return the amount of water trapped. on the top of buildings.

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



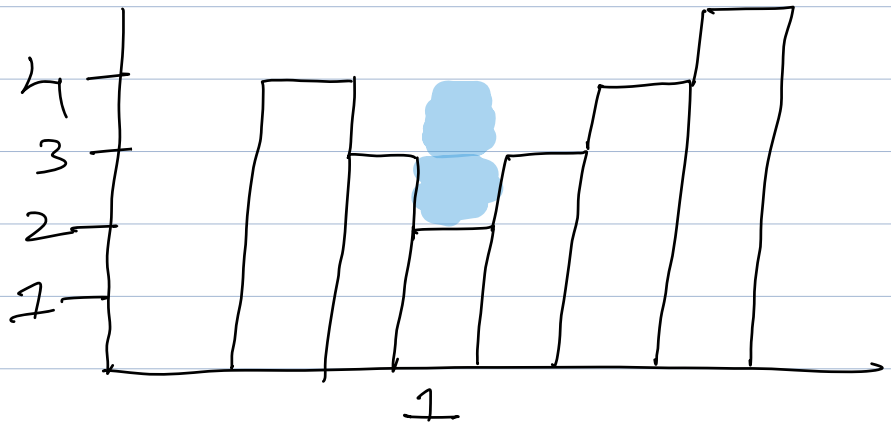
ans ⇒ 6

Observation



no water

#



max height on left ⇒ 4

max height on right ⇒ 5

water trapped ⇒ min (left_max, right_max) − height of the building

Q arr[n] = { -1, 4, 3, 5, 2, 7 }

leftmax[n] : leftmax[i] ⇒ maximum {0...i}

	0	1	2	3	4	5
leftmax[n]:	-1	4	4	5	5	7

⇒ O(n)

rightmax[n] : rightmax[i] ⇒ maximum {i...n-1}

	0	1	2	3	4	5
rightmax[n] =	7	7	7	7	7	7

⇒ O(n)

Pseudo Code

```
int water_trap = 0

for (int i = 1 ; i < (n-1) ; i++) {

    // left_max  => leftmax[i-1]
    // right_max => rightmax[i+1]

    int water => min(left_max, right_max)
                 - arr[i];

    if (water > 0)
        water_trapped += water;

}

return water_trapped;
```

TC: O(n)
SC: O(n)

Q3 Maximum Sum Subarray.

arr[] = { -1, 4, 2, 8, -2, 3 }
(indices: 0 1 2 3 4 5)

ans ⇒ 15

Brute Brute force : For every subarray find the sum. ⇒ $O(n^3)$

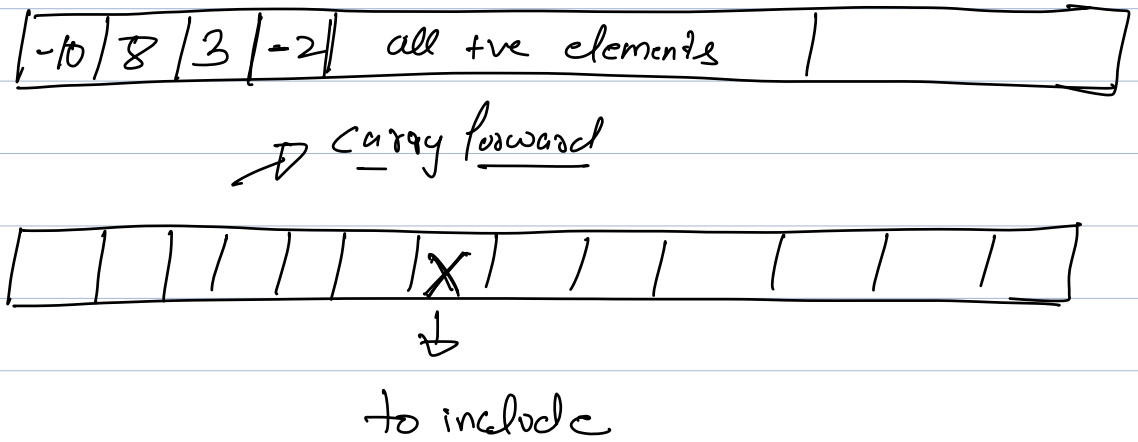
Brute force : Use prefix Sum to find Subarray sum ⇒ $O(n^2)$

Scenario 1 : | 2 | 4 | 6 | 8 |

ans ⇒ 20

Scenario 2 : | -1 | -2 | -3 | -4 |

ans ⇒ -1



arr[]	5	6	7	-3	2	-10	-12	8	12	21
Sum ⇒	5	11	18	15	17	7	-5	8	20	41
CF=0	5	11	18	15	17	7	0	8	20	41

max_sum ⇒ 41

⇒ Kadanes Algorithm.

-4 -3

Sum =

Cf = 0

max_sum ⇒ -Inf

⇒ -4

Pseudo Code!

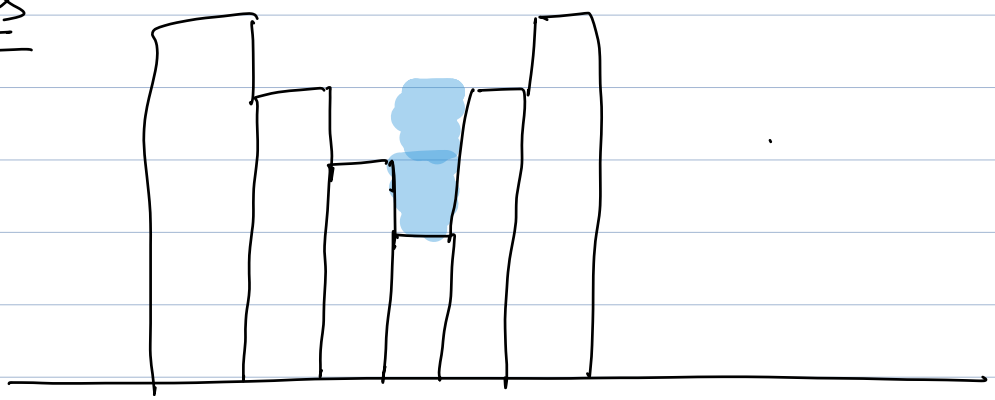
```
int max_sum ⇒ Integer.minimum;
int sum ⇒ 0;

for (int i=0; i<n ; i++) {
    sum ⇒ sum + arr[i];
    max_sum ⇒ max (max_sum, sum);
    if (sum <0)
        sum =0;
}
return max_sum;
```

TC : O(n)
SC : O(1)

-4	-3	-2	-1

max_sum = -1
sum ⇒ 0

Doubts



⇒ leftmax ⇒ 3

rightmax ⇒ 4

= min (3, 4)

⇒ (3)

3 — Height builds ⇒ 1