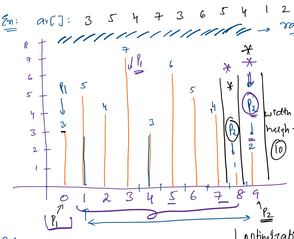
2 Pointers 2
Wednesday, 17 March 2021 4:50 PM

1Q) Given N array elements, where each ar[i] represents height of each building, calculate Area of water accumulated on a rainy day. find max amount of water between any 2 buildings

Ex: ar[]: 3 5 4 7 3 6 5 4 1 2 → rains



		Area of water: Height	width	Area
Ex: 1	4	3	3	9
2	7	4	5	20
3	6	min(7,5)	3	15
2	3	min(4,7)	1	4

// ans = 25, buildings = [1 6]

Sol:

1) To check all pairs

ans = 0
for i = 0 to N-1
  for j = i+1 → N-1
    // area[i] ar[i]
    // area = min(ar[i], ar[j]) * (j-i)
    ans = max(ans, area)
return ans;

TC: $O(N^2)$ SC: $O(1)$

2) Pseudo Code

int P1 = 0, int P2 = N-1
area = 0
while (P1 < P2) {
  ...
}

TC: $O(N)$ SC: $O(1)$

optimization → height: min(ar[P1], ar[P2]); width: P2 - P1; Area

P1	P2			
0	9	→ 2	→ 9	18
0	8	→ min(3, 1) 1	→ 8	8
0	7	→ min(3, 4) 3	→ 7	21
1	7	→ min(5, 4) 4	→ 6	24
1	6	→ min(5, 5) 5	→ 5	25
2	6	→ min(4, 5) 4	→ 4	16
3	6	→ min(7, 5) 5	→ 3	15
3	5	→ min(7, 4) 6	→ 2	12
3	4	→ min(7, 3) 3	→ 1	3

→ Idea: we move either P1 or P2 {we move a pointer because we cannot get a better ans with that particular pointer as fixed.}

→ In all 2-pointer logic, we move a pointer when we know it cannot give you better result

(a + b) = k
P2 - ...

2Q) Given 3 sorted arrays A[], B[], C[], find (i, j, k) such that max(A[i], B[j], C[k]) - min(A[i], B[j], C[k])
{Entire value is minimized.}

Ex:

	0	1	2	3	4	5	6
A[] :	3	14	16	20	29	40	55
B[] :	-6	23	24	30	35	50	
C[] :	15	15	26	29	31	39	42

P1 → Take any P1 } max ↑
P2 → Take any P2 } min ↓
P3 → P3 = -15 } min remain same

P1	P2	P3	ans
3	-6	-15	3 - (-15)

(max - min) ↑

Note: with -15 as P3 I cannot get a better ans, so move P3 to next data point

Ex: a	b	c	max	min	max-min
14	-6	-15	14	-15	29
3	-6	-15	3	-15	18
14	23	15	23	14	9
29	24	26	29	24	5
29	30	31	31	29	2
29	30	29	30	29	1

For above example min ans = 1

P1	P2	P3	Max	Min	Val
3	-6	-15	3	-15	18
3	-6	15	15	-6	21
3	23	15	23	3	20
14	23	15	23	14	9
16	23	15	23	15	8
16	23	26	26	16	10
20	23	26	26	20	6
29	23	28	29	23	6
29	24	26	29	24	5
29	30	26	30	26	4
29	30	29	30	29	1

→ Solutions

1) Run 3 loops [All triplets]
TC: $O(N^3)$ SC: $O(1)$

2) int P1 = 0, P2 = 0, P3 = 0;
while (P1 < N && P2 < N && P3 < N) {
  ...
}

[Repeat till one of P1, P2, P3 is reaching end, continue the process]

3Q) Given a sorted array check if there exists a triplet ar[i] + ar[j] + ar[k] = 0 & i ≠ j ≠ k

	0	1	2	3	4	5	6	7	8
ar[] :	-8	-7	-4	-1	2	3	5	7	9

Solutions

1) Check all triplets
TC: $O(N^3)$ SC: $O(1)$

a + b + c = 0
-8 b + c = 8 // we apply 2 pointer P1 = 1, P2 = N-1
-7 b + c = 7 P1 = 2 P2 = N-1

2) General ar[i] b + c = -ar[i] P1 = i+1, P2 = N-1

Pseudo Code

for (int i = 0; i < N; i++) {
  int sum = -ar[i]
  P1 = i+1, P2 = N-1
  // Now apply 2 pointer logic
}

TC: $O(N^2)$ SC: $O(1)$

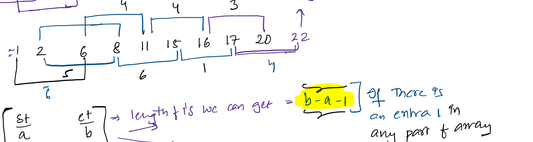
// Can we extend same logic

a + b + c = k → If we fix a = ar[i], b + c = k - ar[i] } 2 pointer sum idea P1 = i+1, P2 = N-1
a - b - c = k → If we fix a = ar[i], b + c = ar[i] - k } 2 pointer diff idea P1 = i+1, P2 = i+2
a + b - c = k → If we fix a = ar[i] b - c = k - ar[i]

4Q) Given a binary string of 0's, 1's, at max we can swap 1 position, find max no. of 1's we can get

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
ar[] :	1	1	0	1	1	1	0	1	0	1	1	0	1	1	1	0	0	0	1	0	1	

array size



Hint: I can make a entire part = 1 if it contains only a single 0

zero Index

st, ct → length of 1's we can get = b - a - 1 } If there is an extra 1 in any part of array

If there is no extra 1 = b - a - 2

between we have a single 0

Ex: 0 1 2 3 4 5 6
0 1 1 0 1 1 0 0 0 0 0
→ length = 6 - 0 - 1 = 5

Note: To check if extra 1 is are there or not get total cnt of 1's
cnt == (b - a - 2)
// That means do you have extra 1's Not really
else
// Means you have extra 1's
ans = max(ans, b - a - 1)

1 hr ⇒ 9 - 10 pm
[Basics of hashing]

8 - 9] optional session [No attendance]
→ Basics of hashmap/hashset
→ Treemap/Treeset
→ 1-2 problems
→
→ Assignments: 1/2 problems

How to use → hashmap/
→ When to use
→ When to use what kind of map
→

→ Friday

— Hashing Basics