Todays Content:

2 Pointers : Solved Questions → { merge, Quick Sort → re-arrange, reverse }

↳ variable pointing indices

↳ ≈ 3/4.....

Qs) Given ar[N] distinct sorted elements, check if there exists a pair (i, j) such that ar[i] + ar[j] = k && i != j

Eg:

ar[5] = { 3 7 8 12 19 } k = 15 : return True

Ideas

a) Check all pairs : O(N^2) SC: O(1)

b) optimize

→ Using hashset ⇒ TC: O(N) SC: O(N)

→ Using Binary Search ⇒ TC: O(N log N) SC: O(1)

ar[5] = { 2 5 8 11 15 } k = 19

a + b = 19

2, search for 17 ✗

5, search for 14 ✗

8, search for 11 ✓

```
i = 0; i < N; i++){
    a = ar[i], b = k - ar[i]
    search b in sorted ar[]
}
```

TC: N log N SC: O(1)

TODO: Handle Edge Cas

Ideas:

	0	1	2	3	4	5	6	7	8	9	
ar[11] = {	-3	0	1	3	6	8	11	14	18	25 }	k = 17

P_1 moves →; P_2 moves ←

P_1	P_2	ar[P_1] + ar[P_2], k
0	9	22 > 17, dec sum P_2--
0	8	15 < 17, inc sum P_1++
1	8	18 > 17, dec sum P_2--
1	7	14 < 17, inc sum P_1++
2	7	15 < 17, inc sum P_1++
3	7	17 == 17 return True

```
bool checksum(int ar[N], int k){   TC: O(N)  SC: O(1)
    P1 = 0, P2 = N-1
    while(P1 < P2){
        if(ar[P1] + ar[P2] == k)
        {   return True     }
        if(ar[P1] + ar[P2] > k){
        |     P2-- // dec sum
        }
        else{
        |     P1++ // inc sum
        }
    }
    return false
}
```

Why?

ar[5] = { 3 9 10 14 18 } k = 19

P_1 at 3; P_2 at 18 → P_2 ⇔ P_1

P_1	P_2
3	18

3 + 18 > 19

$\begin{bmatrix} 9 \\ 10 \\ 14 \end{bmatrix}$ + 18 > 19

18 cannot be in your ans hence reduce P_2

P_1	P_2
3	14

3 + 14 < 19

3 + $\begin{bmatrix} 9 \\ 10 \end{bmatrix}$ < 19

3 cannot be in your ans hence increase P_1

2Q) Given ar[N] sorted distinct elements, check if there exists a pair (i, j) such that ar[j] - ar[i] = k && k > 0

	0	1	2	3	4	5	6	7	8	9	
ar[10] = {	-3	0	1	3	6	8	11	14	18	25 }	k=5
							↑ P_1	↑ P_2			

Case - I : {0, N-1} *

P_1 P_2 : ar[P_2] - ar[P_1]

0 9 : 28 > 5 : dec diff ⇒

: P_1++ & P_2-- diff will decrease

Case - II : {N/2 N/2+1} *

P_1 P_2 : ar[P_2] - ar[P_1]

4 5 : 2 < 5 : inc diff

: P_1-- & P_2++ diff will increase

Case - III : {0, 1} ✓

P_1 P_2 : ar[P_2] - ar[P_1]

0 1 : 3 < 5 : inc diff P_2++

0 2 : 4 < 5 : inc diff P_2++

0 3 : 6 > 5 : dec diff P_1++

1 3 : 3 < 5 : inc diff P_2++

1 4 : 6 > 5 : dec diff P_1++

2 4 : 5 == 5 : return True

Case - IV : {0, N/2} *

P_1 P_2 : ar[P_2] - ar[P_1]

0 4 : 9 > 5 : dec diff

P_1++ & P_2-- diff will decrease

Case - V : {N-2 N-1} ✓

P_1 P_2 : ar[P_2] - ar[P_1]

8 9 : 7 > 5 : dec diff P_2--

8 8 : Same P_1--

7 8 : 4 < 5 : inc diff P_1--

6 8 : 7 > 5 : dec diff P_2--

⋮

```
bool diff(int ar[N], int k){
    p1 = 0, p2 = 1
    k = abs(k) // code works for -ve
                  → needed
    while(p1<N && p2<N)
        if(ar[p2]-ar[p1]==k){return True}
        if(ar[p2]-ar[p1]>k){
            // dec sum p1++
        }
        else{// inc sum
            p2++
        }
    }
    return false
}
```

Note: k = -5

ar[j] - ar[i] = -5

ar[i] - ar[j] = 5

8:05 AM →

8:15 AM break

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

Note: Any 2 pointer

a) Where to initialize
b) How to update
c) While condition
d) We move from an index/pointer:

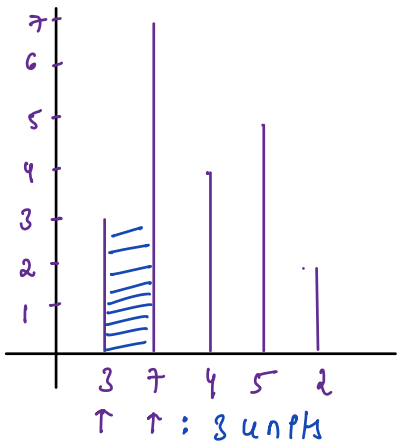
if we cannot get a better ans from that

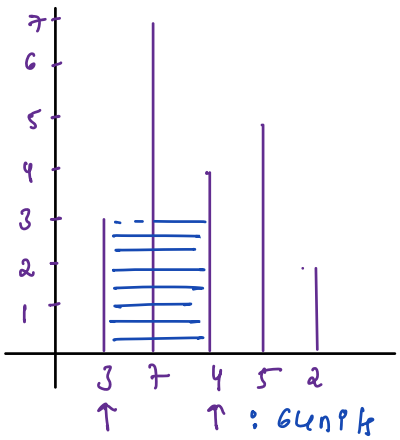
3Q) Water logging

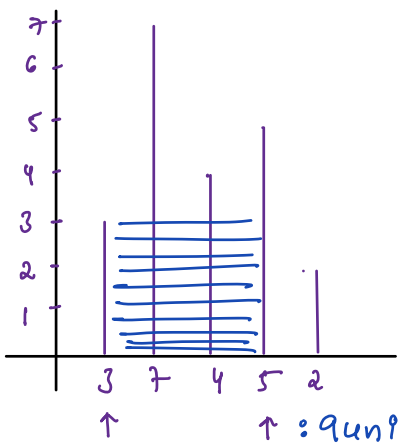
Given ar[N] ele, ar[i] represents height of each wall,

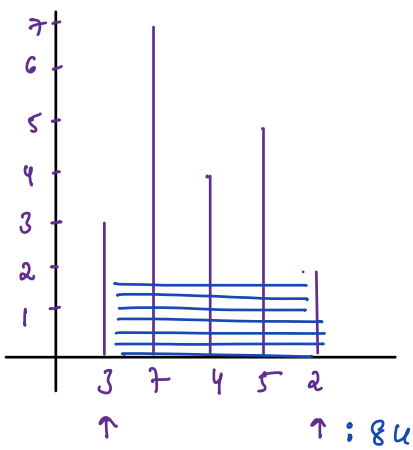
Pick any 2 walls such that, water acumulated between is max

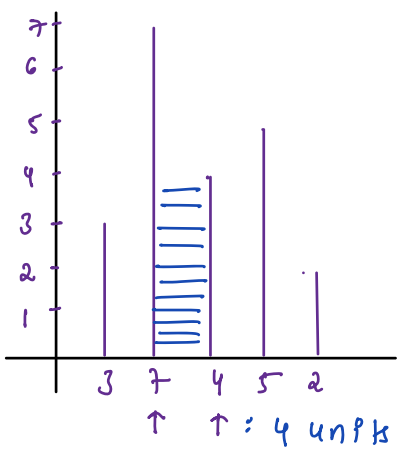
Eg: ar[5] = { 3 7 4 5 2 } : ans = 10

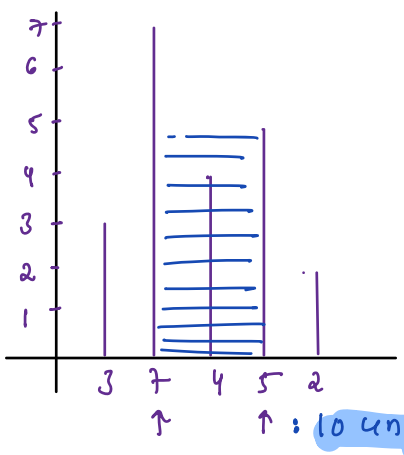


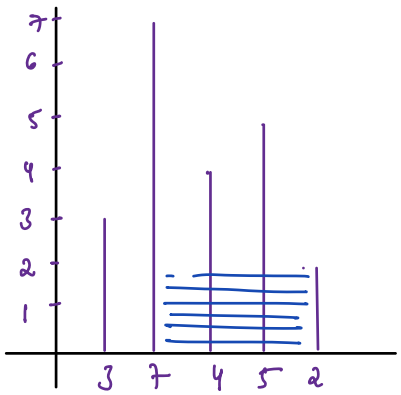




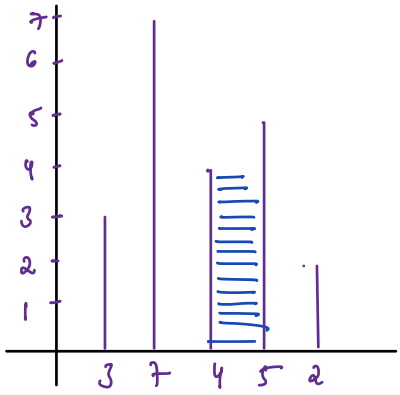




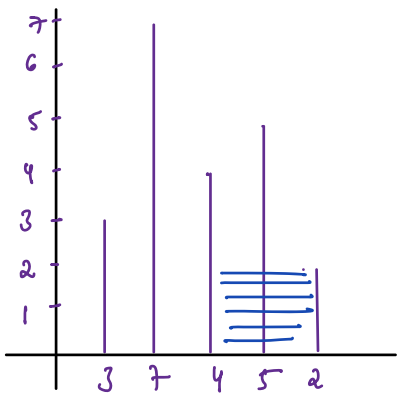




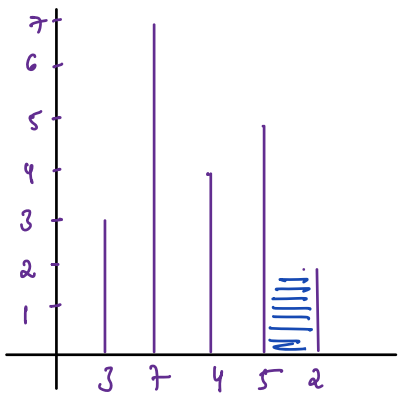
↑ : 6 units



↑ ↑ : 4 units



↑ ↑ 4 units



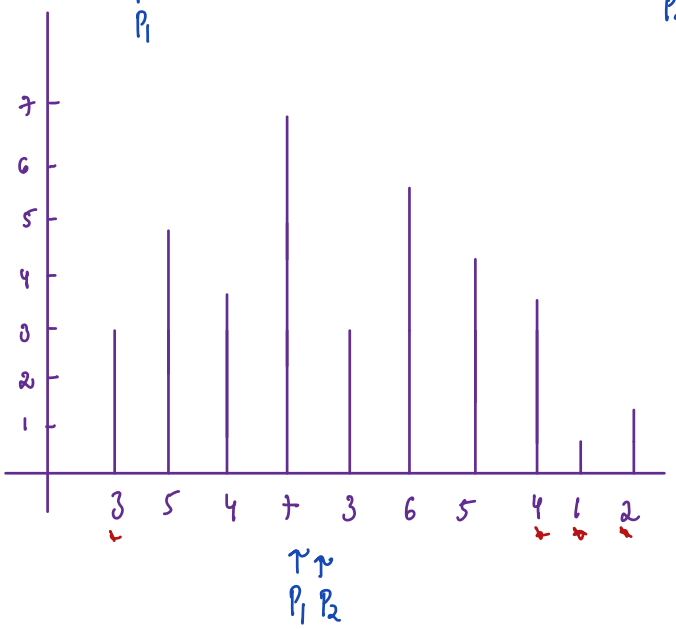
↑ ↑ : 2 units

Idea: Check all pairs TC: $O(N^2)$ SC: $O(1)$

```
ans = __
i=0; i<n; i++) {
    j=i+1; j<n; j++) {
        // 1st wall height = ar[i]
        // 2nd wall height = ar[j]
        H = min(ar[i], ar[j])
        L = j-i
        ans = max(ans, H*L)
    }
}
return ans;
```

Idea: $P_1 = 0$, $P_2 = N-1$ { move pointer with min height } TC: O(N) SC: O(1)

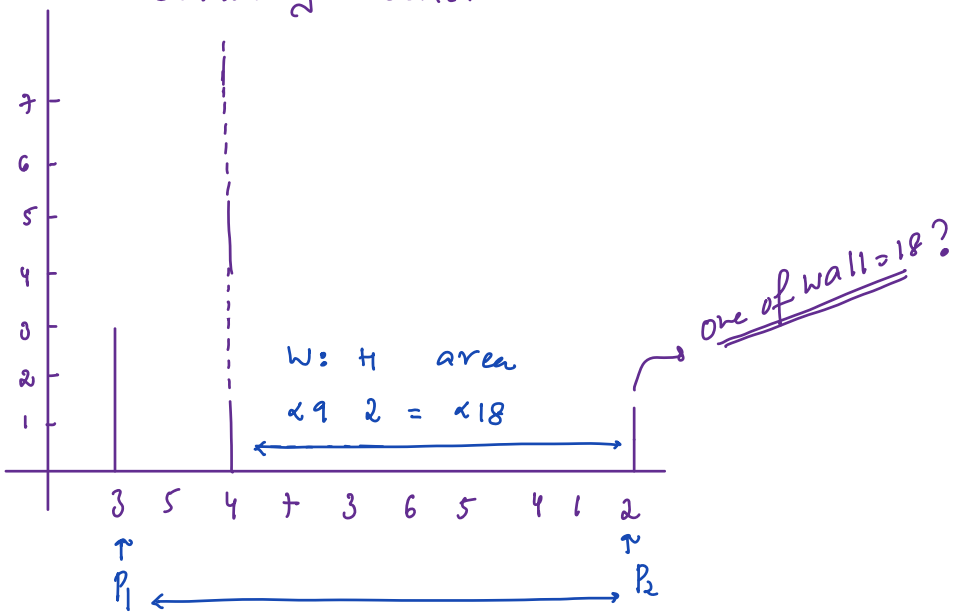
	0	1	2	3	4	5	6	7	8	9
ar[10] = {	3	5	4	7	3	6	5	4	1	2 }



P_1	P_2	$ht = \min(ar[P_1], ar[P_2])$	$L = P_2 - P_1$	area	: Change pointer with min height
0	9	2	9	18	P_2--
0	8	1	8	8	: P_2--
0	7	3	7	21	: P_1++
1	7	4	6	24	: P_2--
1	6	5	5	25	: P_1++
2	6	4	4	16	: P_1++
3	6	5	3	15	: P_2--
3	5	6	2	12	: P_2--
3	4	3	1	3	: P_2--
3	3 { break }				

return max area = 25

Discard? We can move pointer facing min height we cannot get better ans than that.



4Q) Given 3 sorted arrays A[] B[] C[] of size N

find i, j, k such that

max (A[i], B[j], C[k]) − min (A[i], B[j], C[k]) is minimized

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

i	j	k	max(ar[i] ar[j] ar[k]) − min(ar[i] ar[j] ar[k])	
0	0	1	15 − (−6) = 21	ans = 2
1	1	1	23 − 14 = 9	
5	4	4	40 − 35 = 5	
4	3	3	31 − 29 = 2	

Idea1: Check all triplets & get overall min diff TC: O(N^3) SC: O(1)

Idea2:

Idea 2: $P_1 = P_2 = P_3 = 0$ { min pointer with value }

	0	1	2	3	4	5	
A[ ] = {	3	14	16	20	29	40 }	P_1
B[ ] = {	-6	23	24	30	35	50 }	P_2
C[ ] = {	-15	15	26	31	39	42 }	P_3

P_1	P_2	P_3	ans
3	-6	-15	18
14	23		: max − min
16	24		: 23 − (-15)
20	30		: diff > 18
29	35		: With P_3 at -15 we cannot get any less than 18 hence discard
40	50		

P_1	P_2	P_3	max(arr[P_1] arr[P_2] arr[P_3]) − min(arr[P_1] arr[P_2] arr[P_3])	
0	0	0	3 − (-15) =	18
0	0	1	15 − (-6) =	21
0	1	1	23 − 3 =	20
1	1	1	23 − 14 =	9
2	1	1	23 − 15 =	8
2	1	2	26 − 16 =	10
3	1	2	26 − 20 =	6
4	1	2	29 − 23 =	6
4	2	2	29 − 24 =	5
4	3	2	30 − 26 =	4
4	3	3	31 − 29 =	2
5	3	3	40 − 30 =	10
5	4	3	40 − 31 =	9
5	4	4	40 − 35 =	5
5	5	4	50 − 39 =	11
5	5	5	50 − 40 =	10

6 5 5 : { Note P_1: goes out of bounds break }

ans = 2:

Note: any pointer exceeding array finder, you will break & return min ans

TC: O(3N) → O(N)

SC: O(1)

Set:3 : [Hashmap / Strings / TreeMap : 5] [In your language of choice]

[LinkedList : 3]

[Stacks / Queue / Deq : 4]

Set4 : [Trees, Heaps, Tries, Greedy]

Set5 : [Back Tracking, Dp, graphs]