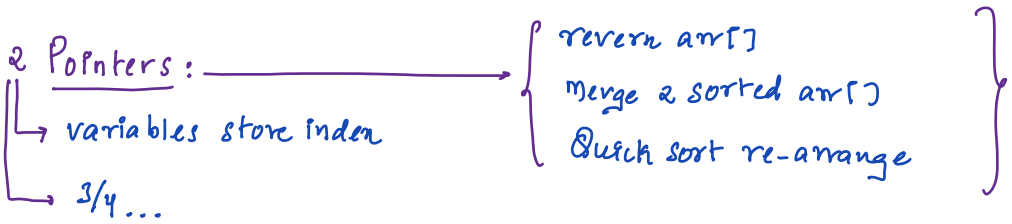
Todays Content:



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:

	0	1	2	3	4	
ar[5] = {	3	7	8	12	19 }	k = 15 : return true

Ideas

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

b) optimize :

a) Using hashmap TC: O(N) SC: O(N) * no extra space

b) Using binary search : fix 1 element search other in sorted arr[]

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

a + b = 19

2 search 17 in arr[] *

5 search 14 in arr[] *

8 search 11 in arr[] ✓

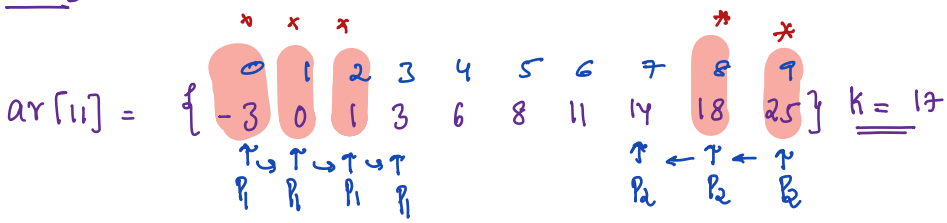
return True

// Pseudocode: TC: O(nlogn) SC: O(1)

```
bool pairsum(int ar[n], int k){

    i = 0; i < n; i++){
        // a = ar[i]
        // b = k - a
        Search for b in arr[]
        using binary search
    }
    Note: It has 1 edge case, handle
    them carefully
}
```

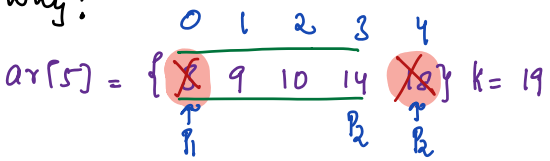
Idea3:



P_1	P_2	$ar[P_1]+ar[P_2]$, k
0	9	$-3+25=22>17$, dec sum, $P_2=P_2-1$
0	8	$-3+18=15<17$, inc sum, $P_1=P_1+1$
1	8	$0+18=18>17$, dec sum $P_2=P_2-1$
1	7	$0+14=14<17$, inc sum, $P_1=P_1+1$
2	7	$1+14=15<17$ inc sum $P_1=P_1+1$
3	7	$3+14=$ 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){
            //dec sum
            P2 = P2-1
        }
        else //inc sum
            P1 = P1+1
    }
    return false
}
```

Why?



P_1	P_2	$ar[P_1]+ar[P_2]$
3	18	21 > 19 : dec sum P_2--

9, 10, 14 + 18 > 19 — 18 cannot be a part of our ans hence we discard

P_1	P_2	$ar[P_1]+ar[P_2]$
3	14	17 < 19 : inc sum P_1++

3 + {10, 9} < 19 — 3 cannot be a part of our ans hence we discard

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

	0	1	2	3	4	5	6	7	8	9	
ar[10] = {	-3	0	1	3	6	8	11	14	21	25 }	k=5
								↑ P1	↑ P2		

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

P1 P2 : ar[P2] - ar[P1]

0 9 : 25 - (-3) = 28 > k

dec diff

P2-- / P1++

ambiguity, we cannot decide

hence above initialization wrong

Case-2: {mid, mid+1}

P1 P2 : ar[P2] - ar[P1]

4 5 : 8 - 6 = 2 < k

: inc diff

P1-- or P2++

ambiguity, we cannot decide

Case-3: {0, mid}

P1 P2 : ar[P2] - ar[P1]

0 4 : 6 - (-3) = 9 > 5

: dec diff

P1++ / P2--

ambiguity, we cannot answer

Case-4: {0, 1}

P1 P2 : ar[P2] - ar[P1]

0 1 : 0 - (-3) = 3 < 5

inc diff P2++

0 2 : 1 - (-3) = 4 < 5

inc diff P2++

0 3 : 3 - (-3) = 6 > 5

dec diff P1++

1 3 : 3 - 0 = 3 < 5 : P2++

1 4 : 6 - 0 = 6 > 5 : P1++

2 4 : 6 - 1 == 5 return True

Case-5: {n-2, n-1}

P1 P2 : ar[P2] - ar[P1]

8 9 : 25 - 21 = 4 < k :

inc dec P1--

7 9 : 25 - 14 = 11 > k

dec dec P2--

⋮

TODO continue tracing

```
bool diff(int ar[N], int k){   TC: O(2*N)  O(N)   SC: O(1)
```

// Case-4 Pseudocode:

```
k = abs(k) // we are making k>=0
p1 = 0, p2 = 1        -> is not needed
while(p1<n && p2<n){
    if(ar[p2]-ar[p1]==k){
        return True
    }
    if(ar[p2]-ar[p1]>k){ //dec diff
        p1++,
        if(p1==p2) p2++
    }
    else{ //inc diff
        p2++
    }
}
return false
```

If $k < 0$:

$ar[j] - ar[i] = k$

$ar[i] - ar[j] = -k$

Obs: If we have a pair with diff, k we will also have a pair with diff -k

Eg: ar[ ] = { 4 (0) 10 (1) 13 (2) } k = 0

p1 p2

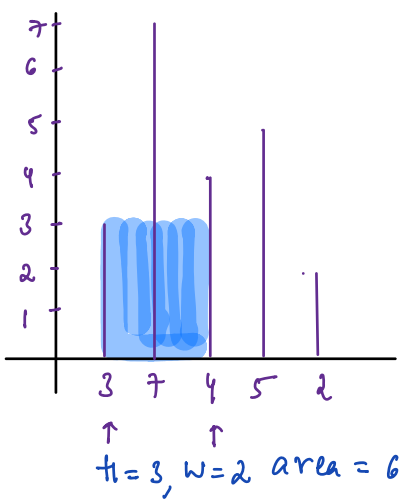
p1	p2	: ar[p2] - ar[p1]
0	1	: 10 - 4 = 6 > 0 : dec p1++
1	1	: p1 == p2, p2++
1	2	: 13 - 10 = 3 > 0 : dec p1++
2	2	: p1 == p2 p2++
2	3	break return false

3Q) Water logging

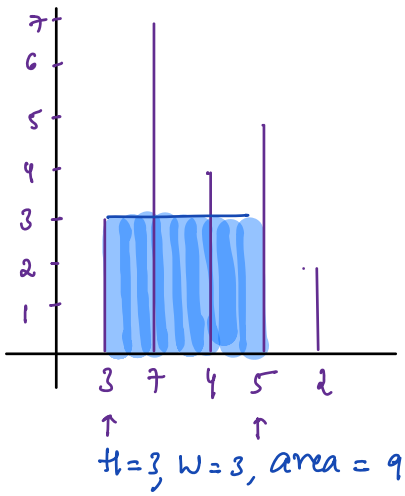
Given ar[N] ele, ar[i] represents height of each wall, find max water accumulated between any 2 walls?

Note: Between 2 walls 1 unit of distance is present

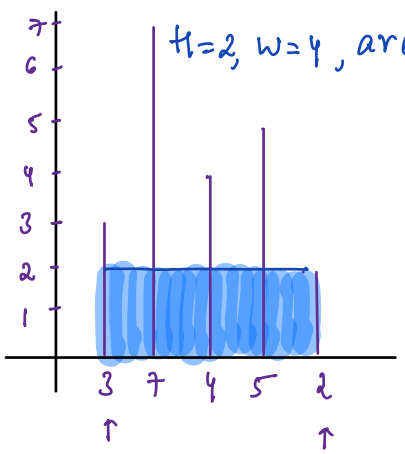
	0	1	2	3	4	
Eg: ar[5] = {	3	7	4	5	2	} ans = 10



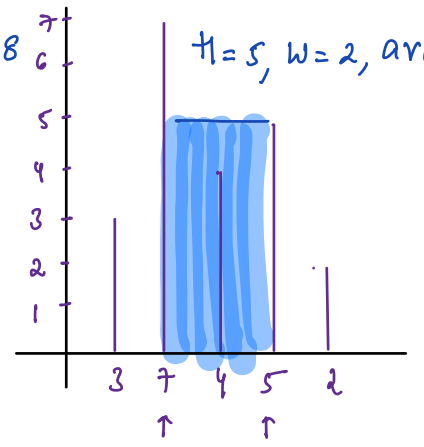
ht = 3, w = 2 area = 6



ht = 3, w = 3, area = 9



ht = 2, w = 4, area = 8



ht = 5, w = 2, area = 10

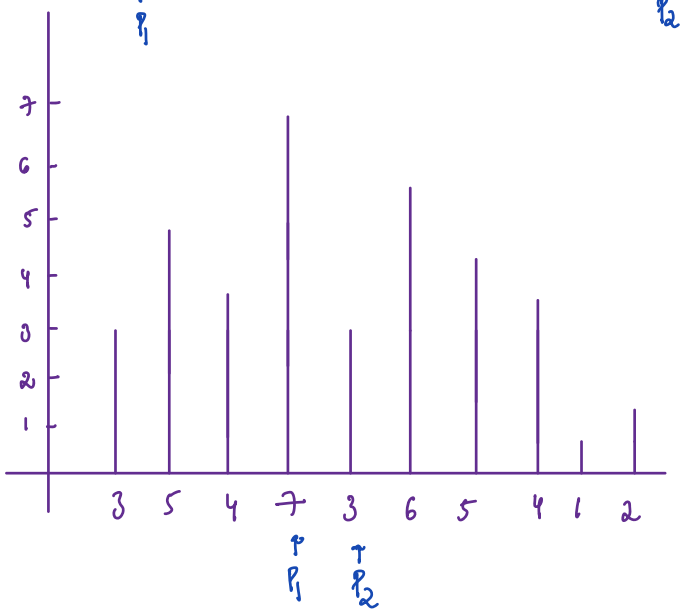
Idea: For every pair, calculate amount of water accumulated between them & get overall max

```
int water(int h[N])   TC: O(N^2)   SC: O(1)
{
    ans = 0
    for(i=0; i<n; i++){
        for(j=i+1, j<n; j++){
            // wall heights are h[i] & h[j]
            area = min(h[i], h[j]) * j - i
            ans = max(ans, area)
        }
    }
}
```

Idea: $P_1 = 0$, $P_2 = N-1$, move pointer with min height & get max area

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



P_1	P_2	$H = min(ar[P_1], ar[P_2])$	$W = P_2 - P_1$	area	update pointer with min value
0	9	$H = min(2, 3) = 2$	$W = 9$,	area = 18	: P_2--
0	8	$H = min(3, 1) = 1$	$W = 8$,	area = 8	: P_2--
0	7	$H = min(3, 4) = 3$	$W = 7$	area = 21	: P_1++
1	7	$H = min(5, 4) = 4$	$W = 6$	area = 24	: P_2--
1	6	$H = min(5, 5) = 5$	$W = 5$	area = 25	: P_1++

Note: If both are same, move any pointer

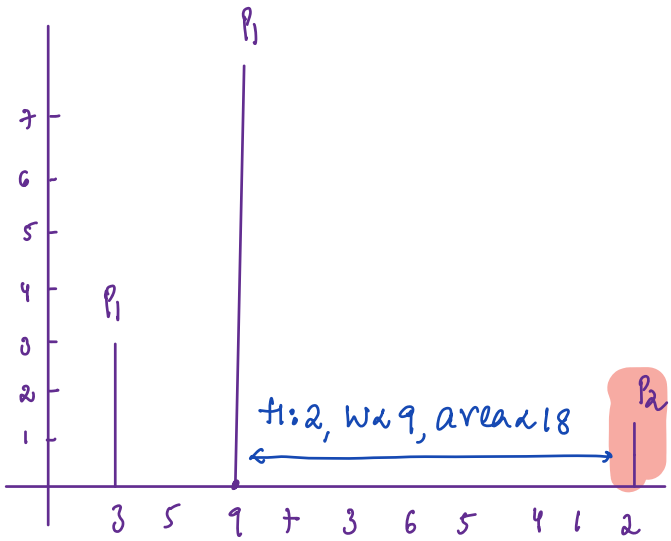
2	6	$H = min(4, 5) = 4$	$W = 4$	area = 16	: P_1++
3	6	$H = min(7, 5) = 5$	$W = 3$	area = 15	: P_2--
3	5	$H = min(7, 6) = 6$	$W = 2$	area = 12	: P_2--
3	4	$H = min(7, 3) = 3$	$W = 1$	area = 3	: P_2--

3 3 $P_1 == P_2$ break

return 25

Discard? We discard wall with min height?

: With that wall we cannot get a better ans, hence discard wall



h=2, w=9, area=18

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

find min { max(trip) − min(trip) }

	0	1	2	3
A[] = {	3	14	16	23 }
B[] = {	-6	23	24	30 }
C[] = {	-15	15	26	31 }

ans = 3

i	j	k	max(ar[i] ar[j] ar[k]) − min(ar[i] ar[j] ar[k])
0	0	1	15 − (−6) = 21
0	0	0	3 − (−15) = 18
3	3	3	31 − (23) = 8
3	1	2	26 − 23 = 3
3	2	2	26 − 23 = 3

Idea: Check all triplets

```
for(i=0; i<n; i++){          TC: O(N3) SC: O(1)
    for(j=0; j<n; j++){
        for(k=0; k<n; k++){
            ==========
        }
    }
}
```

Idea 2: $p_1 = p_2 = p_3 = 0$ move pointer with min val TC: O(N) SC: O(1)

A[] = { 3 14 16 23 } (4) (p_1)

B[] = { -6 23 24 30 } (p_2)

C[] = { -15 15 26 31 } (p_3)

p_1	p_2	p_3	max - min
3	-6	-15	3 - (-15) = 18
14	23		23 - (-15)
16	24		23 + 15 = 18
23	30		

With -15 as one of the triplet we cannot get better ans than 18, hence discard

p_1	p_2	p_3	max(ar[p_1] ar[p_2] ar[p_3]) - min(ar[p_1] ar[p_2] ar[p_3])	
0	0	0	3 - (-15) = 18	move pointer with smaller value p_{1++}
0	0	1	15 - (-6) = 21	p_{2++}
0	1	1	23 - 3 = 20	p_{1++}
1	1	1	23 - 14 = 9	p_{1++}
2	1	1	23 - 15 = 8	p_{3++}
2	1	2	26 - 16 = 10	p_{1++}
3	1	2	26 - 23 = 3	p_{1++}

Note: If 2 values are same move any pointer

3 2 2 26 - 23 = 3 p_{1++}

4 2 2 break? return 3

If one of pointer goes out of bounds we stop

TODO Initialize $p_1 = p_2 = p_3 = n-1$ check if we can discard or not?

```
int mindiff(int A[N], int B[N], int C[N])   TC:O(N) SC:O(1)

    P1 = 0, P2 = 0, P3 = 0
    ans = INT_MAX
    while( P1<n && P2<n && P3<n ){
        V1 = min(A[P1], min(B[P2], C[P3]))
        V2 = max(A[P1], max(B[P2], C[P3]))
        val = V2 - V1
        ans = min(ans, val)
        if( V1 == A[P1] ){ P1++ }
        else if( V1 == B[P2] ){ P2++ }
        else { P3++ }
    }
    return ans;
}
```

Note:

1) Intialization pointers
2) We can discard pointer or not
   : If we cannot get a better ans or ans with that pointer we discard
3) Update pointer
4) while condition

Advanced content: 5 parts →

P1: arrays / bits / maths : ✓

P2: recursion + sorting + binary search + 2 Pointers ✓

P3: Hashmap / Strings / LinkedList / Stacks

P4: Trees / Tries / Heaps // Greedy

P5: Back tracking / Dynamic Prog / Graphs