Two Pointers

Good Evening -

Easiest class of Advance batch

Today's Content

01. Check Pair sum = K
02. Count pair sum = K
03. Pair difference = K
04. Subarr with sum = K
05. Container with most water

Search for one point
- unorganized — TC: O(N)
- Organized — TC: O(log n)

Search for two points
- unorganized — TC: O(n²)
- Organized → 2 pointers

01. Check pair sum

Given a sorted integer array & an integer K.
Check if there is any pair (i, j) such that

A[i] + A[j] == K && i < j

A = { 1 3 5 10 20 23 30 }
0 1 2 3 4 5 6

K = 23 → True

K = 30 → true

Brute force →

01. Check for every pair if a[i]+ar[j]==K

TC: O(n²)

SC: O(1)

02 Binary search A[i] + A[j] = K

A[j] = K − A[i]

Search for (K - A[i]) for each & every A[i]

TC: O(n log n)

SC: O(1)

* 2 pointer technique
  - ↳ where to start
    - diff corners
    - same corners
  - ↳ How to update

A = { 1 3 5 10 20 23 30 } K = 23

0 1 2 3 4 5 6

i → 1

j → 30

A[i] + A[j]		K	
31	>	23	→ decrease the sum j--;
24	>	23	→ decrease the sum j--;
1 + 20	<	23	→ increase the sum, i++;
3 + 20	=	23	return true

```
i = 0        j = N-1

while (i < j) {

    if (A[i] + A[j] == K) return true

    if (A[i] + A[j] > K) j--;

    else i++;

}

return false;
```

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

02. **Count pair sum**

Given a **sorted** integer array & an integer K.
Count all the pair (i, j) such that
A[i] + A[j] == K && **i < j**

A = { 1 3 5 10 20 23 30 }
0 1 2 3 4 5 6

K = 30 (1)
10 + 20

K = 33 (2)
3 + 30
10 + 23

A = { 1 3 3 10 20 23 30 }
0 1 2 3 4 5 6

K = 33 → 3 + 30
3 + 30
10 + 23

Ans = 3

```
i = 0 , j = N-1 , count = 0

while (i < j) {
    if (A[i] + A[j] == K) count++;
    if (A[i] + A[j] > K) j--;
    else i++;
}
return count;
```

unique elements

A = { 1 3 3 10 20 23 30 } K = 33 → 3

A = { 1 3 10 20 23 30 30 } K = 33 → 3

A = { 1 3 3 10 20 23 30 30 } K = 33 → 8

2 * 2 = 4

A = { 10 10 10 10 30 } K = 20

(i → 10, j → 10)

Ans = 6 pairs = $\frac{n * (n-1)}{2}$ // nC_2

10 + 30 > 20

40 > 20

10 + 10 == 20

```
i=0 , j=n-1 , count=0

while (i<j)
    if (A[i]+A[j]==k)
        if (A[i]==A[j]) {
            count = j-i+1
            ans += count * (count-1)/2
            break;
        }
        counti=0
        for (x=i; x<j; x++){
            if (A[x]==A[i]) counti++;
            else break;
        }
        countj=0
        for (x=j; x>i; x--){
            if (A[x]==A[j]) countj++;
            else break;
        }
        ans= counti * countj
        i = i + counti ;
        j = j - countj ;
    }
    if (A[i]+A[j]>k) j--;
    else i++;
```

TC: O(n)

SC: O(1)

3

return ans

03. **Pair difference**

Given a sorted integer array & an integer K. Check if there is any pair (i, j) such that $A[j] - A[i] = K$, $(i \neq j)$ $K > 0$

$j > i$

$A = \{-2, 0, 1, 3, 10, 20, 23\}$

indices: 0, 1, 2, 3, 4, 5, 6

$K = 9$ → true

$K = 15$ → False

Brute force

01. Check for every pair TC: $O(n^2)$

02. Binary Search → TC: $O(n \log n)$

03. Two pointer

A = {	-2	0	1	3	10	20	23 }
	0	1	2	3	4	5	6

K = 9

i j

A[j] - A[i] = 23 - (-2) = 25 > 9 dec the diff

moving j ⟶ 20 - (-2) = 22

moving i ⟶ 23 - 0 = 23

Note

* Keeping two pointers at two distinct corners fail

02 You can either start both pointers from front or both from last

A = {	-2	0	1	3	10	20	23 }
	0	1	2	3	4	5	6

K = 9

j

i

A[j] - A[i] k

increase j → diff increas

increase i → diff decreases

0 - (-2) < 9 j++

1 - (-2) < 9 j++

3 - (-2) < 9 j++

10 - (-2) > 9 i++;

10 - 0 > 9 i++;

10 - 1 == 9 True

```
i=0  j=1
while ( j<n )
|  if (A[j] - A[i] == k) return true.
|
|  if ( A[j] - A[i] < k) {
|  |  j++;
|  }
|  else {
|  |  i++;
|  |  if (i==j) j=i+1;
|  }
}
```

TC: O(n)

SC: O(1)

04. **Subarray with sum = K**

Given an array with **+ve elements** & an integer K. Find any **subarray** with sum = K.
If not possible return {-1, -1}

{L R}

A = { 1 3 10 5 23 3 }
0 1 2 3 4 5

K = 18 = {1, 3}

K = 20 = {-1, -1}

Bruteforce → find sum of all subarrays

TC : O(n³) SC : O(1)

↓ prefixsum approach

TC = O(n²)

SC = O(n)

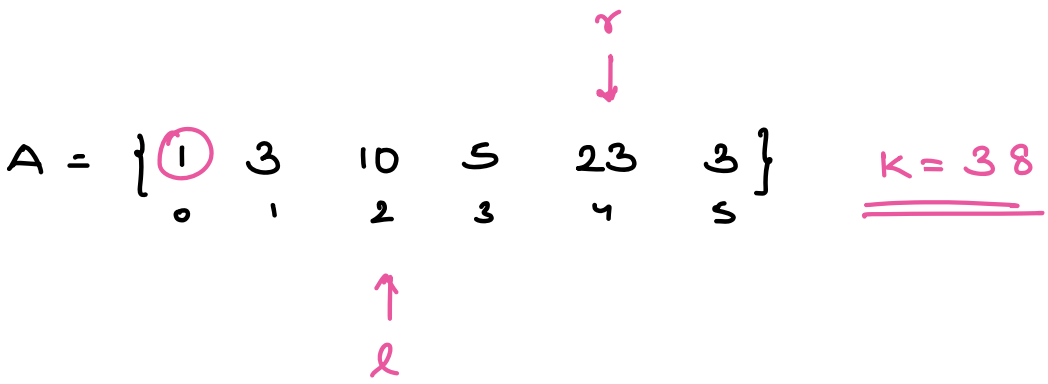
carry forward

TC : O(n²)

SC : O(1)

subarr sum → increase sum → add elements

↳ decrease sum → remove elements



Sum = 0 + 1 < 38

1 + 3 < 38

1 + 3 + 10 < 38

19 < 38

// add elements

r++;

42 > 38

41 > 38

// remove elements

l++;

38 == 38 return true

```
l = 0    r = 0    sum = A[0]

while (r < n) {
    if (sum == k) {
        return {L, R}
    }
    else if (sum > k) {
        sum -= A[l]
        l++
    }
    else {
        r++;
        if (r == n) break;
        sum += A[r];
    }
}
return {-1, -1};
```

TC : O(n)

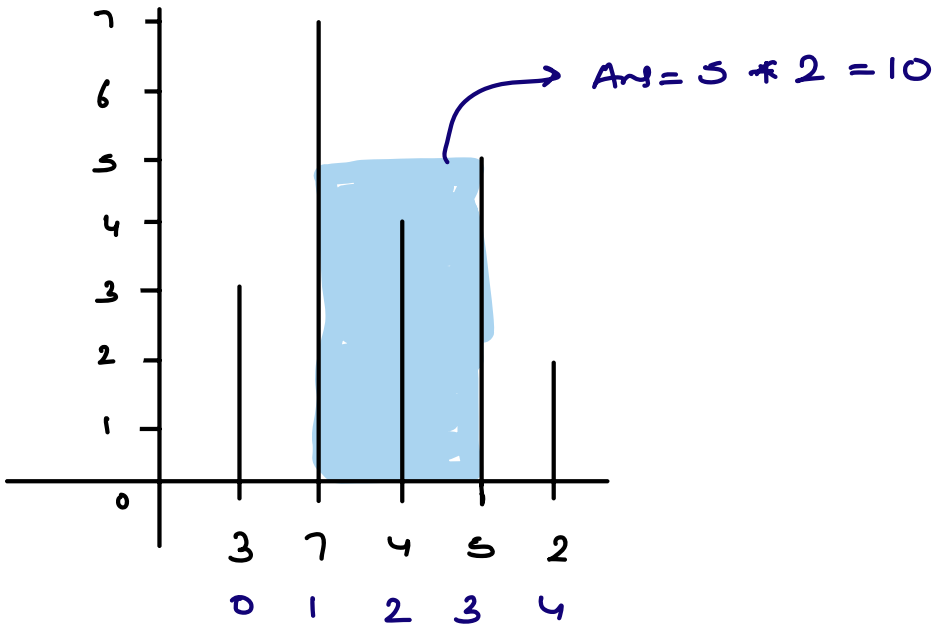
SC : O(1)

05 **Container with most water**

Given ar[N] elements, ar[i] represents height of each wall. Find **max water accumulated** between any two walls

Note :- Between two walls, 1 unit distance is present

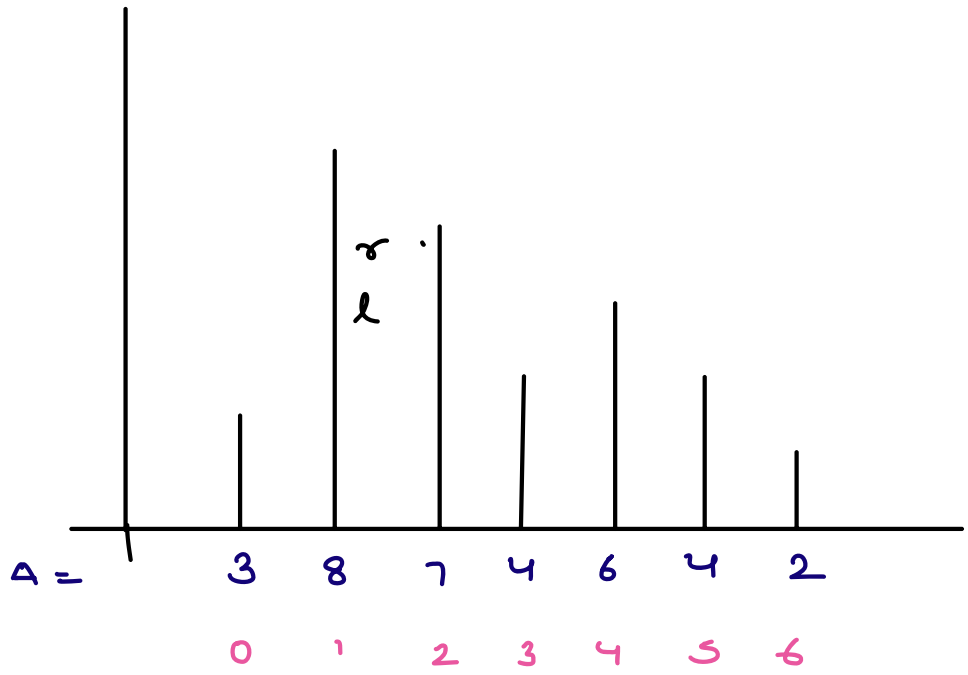
Eg:- ar[5] = { 3 7 4 5 2 }
0 1 2 3 4



ht = min of two walls

wd = j − i

Brute force = Consider all the pairs &
for each pair calculate the
area & update the max:



Max area = W * h

(R - l) min(A[l], A[r]) Area

6 - 0 = 6 min (3, 2) = 2 → 12

5 - 0 = 5 min (3, 4) = 3 → 15

5 - 1 = 4 min (8, 4) = 4 → 16

4 - 1 = 3 min (8, 6) = 6 → 18

3 - 1 = 2 min (8, 4) = 4 → 8

2 - 1 = 1 min (8, 7) = 7 → 7

1 - 1 = 0 → Stop

```
l = 0 , r = n-1 , ans = 0

while (l < r) {
    int ht = min(A[L], A[R])
    int wd = R - L
    area = ht * wd
    ans = Math.max(ans, area):
    if (A[L] < A[R]) l++ :
    else R--;
}

return ans
```

TC: O(n)

SC: O(1)