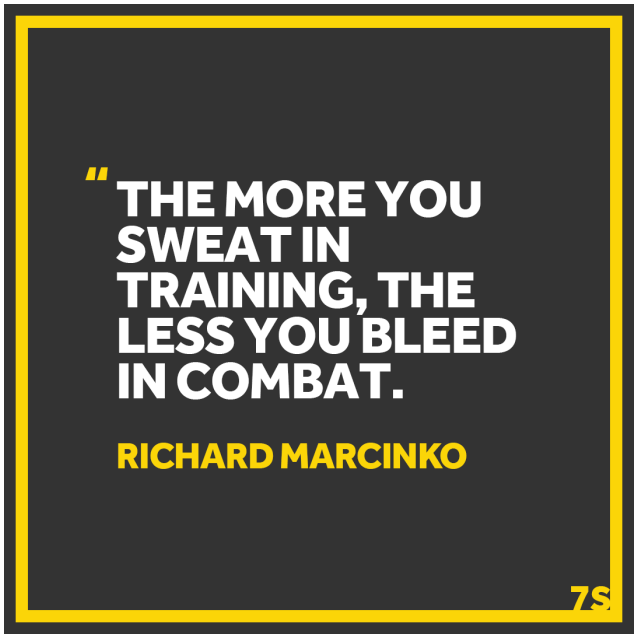
# Two Pointers




Good Evening.

## 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

# Two pointers Technique

Search for a single ele
- Unorganised — TC: O(n)
- Organised — TC: O(log n)

Search for two points
- Unorganised — TC: $O(n^2)$
- Organised — : 2 pointer TC: O(n)

## 01. Check pair sum

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

$A[i] + A[j] == K \quad \&\& \quad i < j$

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

K = 23 → True
K = 40 → True
K = 3 → False

Brute force → Check for every possible pair & see if

$A[i] + A[j] == K$

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

02 Use hashing

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

03 Use binary Search

$A[i] + A[j] = k$

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

For every A[i], search for k-A[i] in the remaing array using Binary Search

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

* Idea 4 → Two pointer technique
  - ↳ where to start
    - some corner
    - diff corners
  - ↳ how to update ?

→ Do not jump directly to Two pointer approach. Disrull all of them in interview.

A[] = { 1 3 5 10 20 23 30 } K = 23
(0) (1) (2) (3) (4) (5) (6)

i ↑ j ↑

| A[i] + A[j] | | K | |
|---|---|---|---|
| 1 + 30 = 31 | > | 23 | → decrease the sum : j--; |
| 1 + 23 = 24 | > | 23 | → decrease the sum : j--; |
| 1 + 20 = 21 | < | 23 | → increase the sum : i++; |
| 3 + 20 = 23 | = = | 23 | → True |

```
i = 0 , j = n-1
while ( i < j ){
    sum = A[i] + A[j]
    if ( sum == k) return true;
    if ( sum > 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 pairs (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 = 33

Ans = 2 { 1, 6 }
{ 3, 5 }

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

k = 33

Ans = 3
(1, 6)
(2, 6)
(3, 5)

```
i = 0 , j = n-1 , count = 0
while ( i < j ){
    sum = A[i] + A[j]
    if ( sum == k)  count ++;
    if ( sum > k ) j--;
    else { i++; }
}
return count;
```

Unique elements

A[ ] = { 1 3 3 10 23 30 }  k = 33 → Ans = 3

A[ ] = { 1 3 3 10 23 30 30 }  k = 33

i → 3 3 , j → 30 30

2 * 2 = 4 pairs

cnt = ~~1~~ ~~2~~ 3

A[ ] = { 10 10 10 10 30 }  k = 20

i → 10 10 10 10 ← j

$$Ans = {}^4C_2 = \frac{4 * (4-1)}{2} = 6$$

// ${}^nC_2 = \frac{n*(n-1)}{2}$

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

while (i<j){

    sum= A[i]+A[j]

    if (sum == k){
        if (A[i]==A[j]){
            cnt = j-i+1
            ans = ans + cnt *(cnt-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 = ans + counti * countj

        i = i + counti
        j = j - countj
    }
    else if (sum > k) j--;

    else {i++j;
}
return ans;
```

TC : O(n)

SC : O(1)

03. **Pair difference**

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

$j > i$

$A = \{ -2, 0, 1, 3, 10, 20, 23 \}$
indices: 0, 1, 2, 3, 4, 5, 6

K = 9
True
(2, 4)

K = 15
False

$A[] = \{ 1, 2, 4, 5, 6, 12 \}$
indices: 0, 1, 2, 3, 4, 5

K = 10
True (1, 5)

Ideas →

01. Check for every pair — TC: $O(n^2)$ — SC: $O(1)$
02 Hashset — TC: $O(n)$ — SC: $O(n)$
03. Binary Search — TC: $O(n \log n)$ — SC: $O(1)$
04. Two pointer Technique

$A[] = \{ -2, 0, 1, 3, 10, 20, 30 \}$ K = 9
indices: 0, 1, 2, 3, 4, 5, 6
i, j

A[j] - A[i] K

30 - (-2) = 32 > 9 ⇒ decrease the diff

20 - (-2) = 22 smaller 32 j--

30 - 0 = 30 smaller 32 i++.

} Ambiguity

* Start from font or start from last

A[ ] = { -2 0 1 3 10 20 30 } K = 9

0 1 2 3 4 5 6

i j

A[j] - A[i] K

0 - (-2) = 2 < 9 : increase diff → j++;

1 - (-2) = 3 < 9 : increase diff → j++;

3 - (-2) = 5 < 9 : increase diff → j++;

10 - (-2) = 12 > 9 : decrease diff → i++;

10 - 0 = 10 > 9 : decrease diff → i++;

10 - 1 = 9 == 9 → True

A[ ] = { -2 0 2 3 10 20 30 } K = 2

0 1 2 3 4 5 6

i j

3 - (-2) = 5 > 2

3 - 0 = 3 > 2

3 - 2 = 1 < 2

```
i=0 , j=1

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

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

10:23 pm → 10:33 pm

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}

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

K=18

subarr [3, 10, 5]

A[] = { 1 2 5 4 3 }

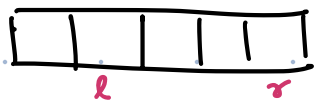
K=9

subarr [5, 4]

Brute force → Consider all subarr & find sum, check if sum == k

TC : O(n³)
SC : O(1)

prefix array

Pf[] = 


Pf[r] - Pf[l-1] == k

Pf[y] - Pf[x] == k ⟵ Diff in pref arrgy = k

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

Use 2 pointer technique in this prefix arrgy to check if (diff == k)

when not allowed to modify in given arr.

* Dynamic sliding window technique

sum = A[0] = 1

A[] = { 1 3 10 5 23 3 }  k = 38

i ↑ j ↑

1 + 3 = 4 < 38
4 + 10 = 14 < 38
14 + 5 = 19 < 38

increase subarr sum ⇓ expanding window → j++

$19 + 23 = 42 > 38$

$42 - 1 = 41 > 38$

$41 - 3 = 38 == 38$

↓

subarr [i j]

decrease subarr sum

⇓

drop ele → i+1

i = 0 , j = 0 , sum = A[0]

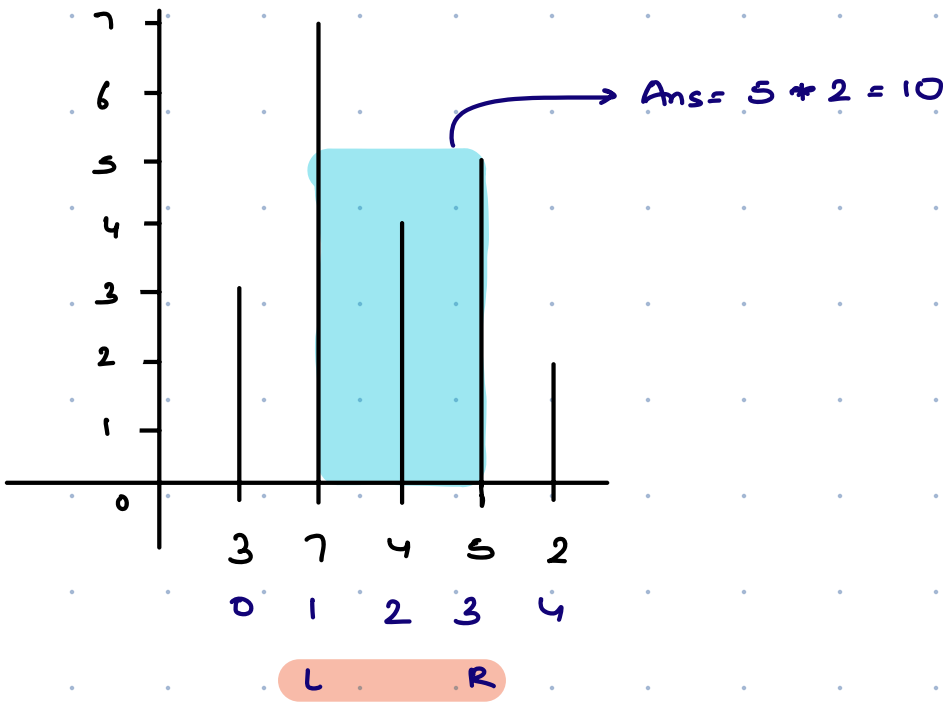
```
while (j<n){
    if (sum == k){
        AL<I> ans = new AL<>();
        for (x=i; x<=j; x++){
            ans.add(A[x]);
        }
        return ans;
    }
    else if (sum > k){
        sum = sum - A[i];
        i++;
    }
    else {
        j++;
        if (j<n) sum = sum + A[j];
    }
}
return {-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

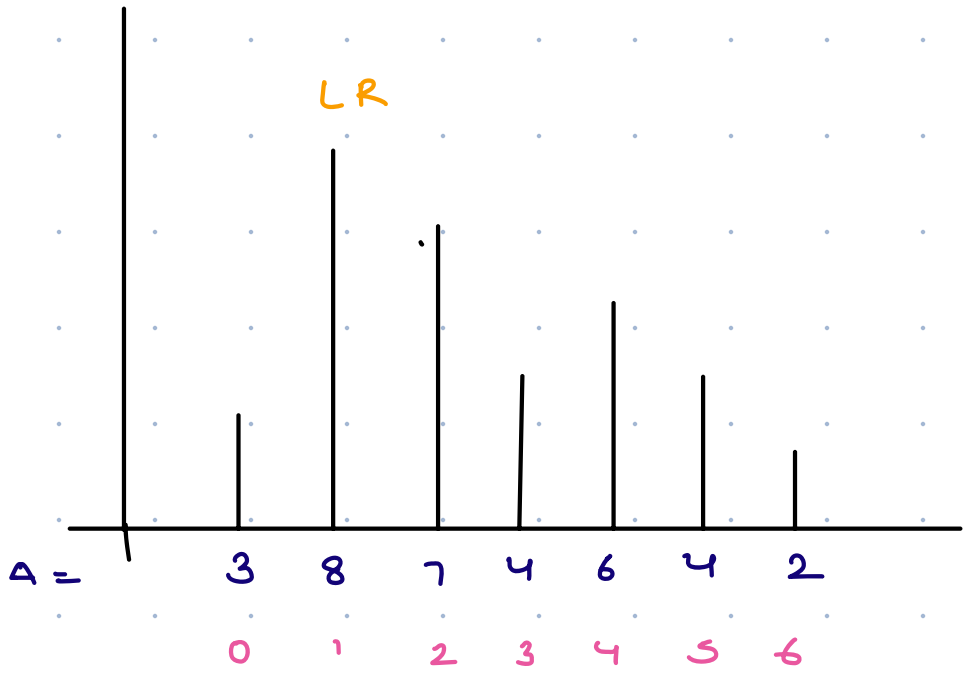



$$Water = \underbrace{\min(A[L], A[R])}_{ht} * \underbrace{(R-L)}_{width}$$

Brute force → Consider all the pairs & calculate water stored with formula = min ht * width, maintain the max ans.

TC : $O(n^2)$

SC : $O(1)$




Max Area = minht * width

| min( A[l], A[r]) | (r - l) | Area | Max Area = 0 |
|---|---|---|---|
| min (3, 2) = 2 | (6 - 0) = 6 | 2 * 6 = 12 | 12 |
| min (3, 4) = 3 | (5 - 0) = 5 | 3 * 15 = 15 | 15 |
| min (8, 4) = 4 | (5 - 1) = 4 | 4 * 4 = 16 | 16 |
| min (8, 6) = 6 | (4 - 1) = 3 | 6 * 3 = 18 | 18 |
| min (8, 4) = 4 | (3 - 1) = 2 | 4 * 2 = 8 | 18 |
| min (8, 7) = 7 | (2 - 1) = 1 | 7 * 1 = 7 | 18 |

Observation → keeping the smaller pointer fixed & moving larger ht ptr will always give smaller area.

* Code

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

while (l < r){

    int ht = min(A[l], A[r]);

    int w = R - l

    ans = max(ans, w * ht);

    if (A[l] < A[r]) l++;

    else { r-- }

}

return ans;
```

TC : O(n)

SC : O(1)