Two Pointers

Topics To Cover
1) Check Pair sum = K
2) Count pair sum = K
3) Pair diff. = K
4) Subarr with sum = K
5) Containes with most waters.

song: Girlfriend - Avril Lavigne

Hi Everyone!!!

Search for 1 target
- Unorganised → Linear Search $\theta(N)$
- Organised → Binary Search $\theta(\lg N)$

Search for a pair
- Unorganised → linear nested search $\theta(N^2)$
- Organised → Two pointer $\theta(N)$

Q. Check pair sum
Given a sorted integer array & an int K. Check (T/F) 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 }

K = 23 ? True K = 30 ? True

HS
3 1
5 10

BF: 1) Check for all pairs using 2 nested for loops
TC: $\theta(N^2)$
SC: $\theta(1)$

2) Hashset, save all values and try finding K - Arr[i]
TC: $\theta(N)$
SC: $\theta(N)$

3) Binary Search: Pick each ele and look for K - Arr[i] using BS
TC: $\theta(N \log N)$
SC: $\theta(1)$

2 Pointers → where to put pointers (both @ start/end or corners)
→ How to move pointers

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

L R

① Put pointer at start and end.	L	R	Sum	
② if sum has to be decr than R--	1	30	31	> K → R-- (decr sum)
else L++	1	23	24	> K → R--
	1	20	21	< K → L++ (to incr sum)
	3	20	23	→ Ans = True

```
fn { find-pair (Ar[N], K)
    L = 0 ; R = N-1 ;
    while ( L < R )
    {
        if ( Ar[L] + Ar[R] == K )      return True ;
        else if ( Ar[L] + Ar[R] > K )  R-- ;
        else                           L++
    }
    return False ;
}
```

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

Q. Check pair sum
Given a sorted integers array & an int K. Count all the 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 = 30 ?
Ans: 1
{3, 4}

K = 33 ?
Ans = 2
{1, 6} {3, 5}

```
fn { find-pair (Ar[N], K)          // This only works for Non-duplicates.
    count = 0;
    L = 0 ; R = N-1;
    while (L < R)
    {
        if (Ar[L] + Ar[R] == K)      count++;
        if (Ar[L] + Ar[R] > K)       R--;
        else                         L++
    }
    return count;
}
```

A = { 1 3 3 10 20 23 30 } K = 33 (L, R) count = 111 = 3

A = { 1 3 10 10 20 30 30 } K = 33 (L, R) 1 = ✗

① A = { 1 3 3 10 20 30 30 } K = 33 (L = 10, R = 20)

2 2

count += 2 * 2 = 4

② Ar: [10 10 10 10 so] K = 20 (L, R)

$^4C_2 \Rightarrow \frac{4 \times 3}{2} = 6$ $\quad ^nC_2 = \frac{n(n-1)}{2}$

```
fn { count-sum (Ar[N], K)
    L = 0 ; R = N-1;
    Ans = 0;
    while (L < R)
    {
        if (Ar[L] + Ar[R] == K)
        {
            if (Ar[L] == Ar[R])
                cnt = (R - L + 1)
```

```
            Ans += cnt * (cnt-1) / 2        (cnt C2)
            return Ans;
        }

        count_L = 0;
        for(i=L ; i<R ; i++)
        {
            if (Ar[L] == Ar[i])
                count_L++;
            else
                break;
        }

        count_R = 0;
        for(i=R ; i>L ; i--)
        {
            if (Ar[R] == Ar[i])
                count_R++;
            else
                break;
        }
        Ans += count_L * count_R;

        R = R - count_R;
        L = L + count_L;
    }

    else if (Ar[L] + Ar[R] > K)  R--;

    else  L++;
  }
  return ans;
}
```

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

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

$A[j] - A[i] = K \quad \&\& \quad (i < j) \quad (K > 0)$

$A[j] > A[i]$

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

K = 9 ?
Ans = True
10 - 1 = 9

K = 15 ?
Ans = False

where to put the pointers

(1)

A = { -2 0 1 3 10 20 23 } K = 9

L → 0, R → 23

L	R	diff
-2	23	25
-2	20	22
0	23	23

observation: > K → R--; L++

Ambiguity

(2)

A = { -2 0 1 3 10 20 23 } K = 9

L → 1, R → 10

pointers @ start

L	R	diff	observation	
-2	0	2	< K	→ R++
-2	1	3	< K	→ R++
-2	3	5	< K	→ R++
-2	10	12	> K	→ L++
0	10	10	> K	→ L++
1	10	9	== K	TRUE

R++ → diff ↑

L++ → diff ↓

```
fn { pair-diff ( Ar[N] , K)
    L = 0 ; R = 1 ;
    while (R < N)
    {
        if (Ar[R] - Ar[L] == K)  return True ;
        else if (Ar[R] - Ar[L] > K)
        {
            L++ ;
            if ( R == L)
                R++ ;
        }
        else  R++ ;
    }
    return False ;
}
```

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

Break : 10:22 - 10:30

Q. Given an array with +ve no. & integer K. Find a subarray with sum == K. If not possible return {-1, -1}.

$K > 0$

Ar[i]

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

K = 18 ? → { 1, 3 }

K = 20 ? → {-1, -1}

Approach → BF → check all sub array
TC: $O(N^3)$

↓

Pf sum → create pf sum and check all pairs
TC: $O(N^2)$
SC: $O(N)$

↓

Binary Search → Pf sum (sorted by default)

TC: $O(N \log N)$
SC: $O(N)$


```
fn subar_sum (Ar[N], int K)
{
    L=0 ; R=0 ; Sum = Ar[0];

    while (R < N && L < N)
    {
        if (Sum == K)
            return {L, R};

        else if (Sum < K)
        {
            R++;
            if (R<N)
                Sum += Ar[R];
        }

        else
        {
            Sum -= Ar[L];
            L++;
        }
    }
    return {-1, -1}
}
```

TC: $\Theta(N)$
SC: $\Theta(1)$

A = { 19 3 10 5 23 3 }
0 1 2 3 4 5

R L

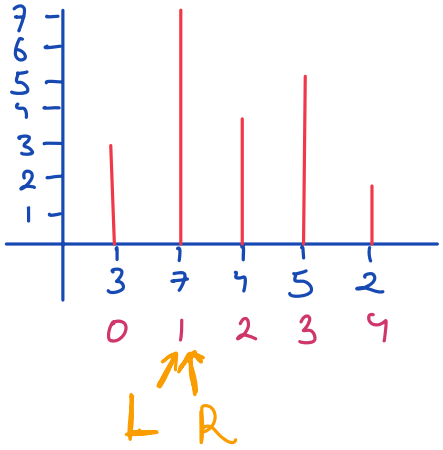
Sum = ~~19~~ ~~Ø~~ ~~8~~ ~~18~~ 18

K = 18

Q. Container with most water.

Given ar[N] elements, ar[i] represents height of a wall. Find max water accumulated b/w any two walls.

Ar: [3 7 4 5 2]
0 1 2 3 4



* move pointer based on pointer has the smaller height

select any 2 walls

water cap = min(Ar[L], Ar[R]) * R-L
(height)

Ans = 10

Index	L	R	Water Cap (min(Ar[L], Ar[R]) * R-L)
→	0	4	2 * 4 = 8
	0	3	3 * 3 = 9
	1	3	5 * 2 = 10 → Ans
	1	2	4 * 1 = 4
	1	1	→ exit

```
L = 0; R = N-1;   ans = 0;
while (L < R)
{
    h = min(Ar[L], Ar[R]);
    w = R-L;
    cap = h * w

    ans = max(ans, cap)

    if (Ar[L] < Ar[R])  L++;
```

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

```
        else  R--;
    }
return ans;
```