Todays Content

1. 2 Pointers Intro
2. a+b=k
3. a-b=k
4. Closest pair sum.

2 Pointer

1. variables storing index as value;
2. Can be extended:

   3 pointers

Q1 Given ar[N] distinct sorted elements, check if there exists a pair (i, j) such that ar[i]+ar[j] = k && i != j.

            0   1   2   3   4   5   6

Ex1: ar[7] = { 3   7   8   11   14   19   20 } k = 25. return True.

Ideas:

a) Generate all pairs, check if sum == k.

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

b) Iterate on ar[]

   For a = ar[i], b = k - a, binary search [i+1..N-1]

            0   1   2   3 | 4   5   6

   ar[7] = { ~~3~~   7   8   11 | 14   19   20 } k = 25

                           a | b

   a+b = 25

   3 22 search b in ar[1..6] *

   7 18 search b in ar[2..6] *

   8 17 search b in ar[3..6] *

   11 14 search b in ar[4..6] ✓

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

c) Solve it using HashSet/HashMap

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

Idea3: 2 Pointers

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 P_2 P_2 P_2

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

```
bool checkSum(vector<int> &arr, int k){ TC: O(N)
    int P1=0, P2 = arr.size()-1;
    while(P1<P2){
        if(ar[P1]+ar[P2]==k){return true}
        if(ar[P1]+ar[P2] < k){
            # Inc sum;
        } P1++;
        else{ #ar[P1]+ar[P2]>k #Dec sum
        } P2--;
    }
    return false;
}
```

#Note: At each iteration we skip an element, total N elements, at max it will take N iterations

#Note: logic will work only if arr() sorted.

Why?

$$
\begin{array}{lcccccccccc}
 & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & \\
ar[\,] = & \{ & \cancel{3} & 5 & 10 & 13 & 15 & 19 & 24 & \cancel{30} & \} \quad k = 28 \\
 & & P_1 & P_1 & & & & & P_2 & P_2 &
\end{array}
$$

P_1 P_2 $ar[P_1] + ar[P_2]$

0 7 $ar[0] + ar[7] = 33 > 28$; Dec sum; P_2-- ?

Reason: $ar[7]$ + smallest value > 28, $ar[7]$ + any value > 28

It's a gurantee with ar[7] we cannot get ans hence discard

0 6 $ar[0] + ar[6] = 27 < 28$; Inc sum; P_1++ ?

Reason: $ar[0]$ + greates value < 27 $ar[0]$ + anyvale < 28

It's a gurantee with ar[0] we cannot get ans hence discard

Q2: Given ar[N] sorted elements.

Check if there exists a pair (i,j) such that ar[j]-ar[i] = k

Note: i != j && k >= 0, #update k < 0 or k >= 0

ar[10] = { -3 0 1 3 6 8 11 14 21 25 } k = 5

Ideas: 1. Brute Force generate pairs TC: $O(N^2)$

2. Using BS TC: $O(N \log N)$

3. Using HM/HS TC: $O(N)$ SC: $O(N)$

2 Pointers:

	0	1	2	3	4	5	6	7	8	9	
ar[10] = {	-3	0	1	3	6	8	11	14	21	25 }	k = 5
			P_1		P_2						

Case : $P_1 = 0$ $P_2 = 1$

P_1	P_2	$ar[P_2] - ar[P_1]$	
0	1	$0-(-3) = 3 < 5$	Inc Diff : P_2++
0	2	$1-(-3) = 4 < 5$	Inc Diff : P_2++
0	3	$3-(-3) = 6 > 5$	Dec Diff: P_1++
1	3	$3 - 0 = 3 < 5$	Inc Diff : P_2++
1	4	$6 - 0 = 6 > 5$	Dec Diff: P_1++
2	4	$6 - 1 = 5 == 5$;	return True;

Case : $P_1 = 0$ $P_2 = 9$

P_1	P_2	$ar[P_2] - ar[P_1]$	
0	9	$25-(-3) = 28 > 5$	↓ Dec Diff

#Issue: Either update P_1 or P_2 diff dec, we cannot decide which to update.

#Con: It's a wrong way to Initialize

#2 Pointers:	0	1	2	3	4	5	6	7	8	9	
ar[10] = {	-3	0	1	3	6	8	11	14	21	25 }	k = 5
					P_1	P_2					

Case : $P_1 = N/2$ $P_2 = N/2 + 1$

P_1	P_2	ar[P_2] - ar[P_1]
4	5	8 - 6 = 2 < 5 : Inc Diff

#Issue: Either update P_1 or P_2 Inc Diff
we cannot decide which to update.

#Con: It's a wrong way to Initialize

#2 Pointers:	0	1	2	3	4	5	6	7	8	9	
ar[10] = {	-3	0	1	3	6	8	11	14	21	25 }	k = 5
								P_1		P_2	

Case : $P_1 = N-2$ $P_2 = N-1$

P_1	P_2	ar[P_2] - ar[P_1]
8	9	25 - 21 = 4 < 5 Inc Diff, P_1--
7	9	25 - 14 = 11 > 5 Dec Diff, P_2--

#Note: We initialize pointers in such a way that, there is no ambiguity while updating pointers

$\rightarrow$ P_1 iterations P_2 iterations.

TC: O(N+N)

```
bool check(vector<int> &arr, int k){
    int P1=0, P2=1, N= arr.size();
    k= abs(k);
    while( P2 < N ){
        if( ar[P2] - ar[P1] == k){
            return true;
        }
        else if( ar[P2]- ar[P1] > k){
            #Dec sum;
            P1++;
            if( P1==P2){ P2++;}
        }
        else{ # ar[P2]- ar[P1] < k
            Inc sum
            P2++;
        }
    }
    return false;
}
```

Edge Case

ar[3] = { 4 10 13 } P_2 k=10 (indices 0 1 2 3)

Case: P_1=0 P_2=1

P_1	P_2	ar[P_2]-ar[P_1]
0	1	10-4 = 6 < 10 Inc Diff P_2++
0	2	13-4 = 9 < 10 Inc Diff P_2++
0	3	stop process

ar[3] = { 4 10 13 13 } k=0 (indices 0 1 2 3)

Case: P_1=0 P_2=1

P_1	P_2	ar[P_2]-ar[P_1]
0	1	10-4 = 6 > 0; Dec Diff P_1++
1	1	if P_1==P_2; P_2++;
1	2	13-10=3 > 0 Dec Diff P_1++
2	2	if P_1==P_2; P_2++;
2	3	13-13 = 0 return True;

#Note: If there exists a pair with diff k, there will exists a diff with -k as well.

(a-b) = k (b-a) = -k.

#Con: If question k<0:

Check pair for it's absolute value;

38. Closest pair to x.

Given sorted arrays $A[N]$ & $B[M]$ & x

Return min value of expression $|A[i]+B[j]-x|$

Ex1:

	0	1	2	3	4	5
A[] = {	3	7	8	10	11	14 }
B[] = {	2	6	9	12	14	21 }

$x = 20$

Dry Run:

i	j	$\lvert A[i]+B[j]-x \rvert$
0	0	$\lvert 3+2-20 \rvert = \lvert -15 \rvert = 15$
2	3	$\lvert 8+12-20 \rvert = \lvert 20-20 \rvert = 0$

Idea1: Generate all pairs TC: $O(N^2)$

```
i=0; i<N; i++){          # (i,j) ≠ (j,i)
    j=0; j<N; j++){
        ans = min(ans, abs(A[i]+B[j]-x))
    }
}
```

Idea2: Using BS TODO .

Fix 1 ele:

For that ele, get min expression value.

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

	0	1	2	3	4	5	
A[] = {	3	7	8	10	11	14	}
B[] = {	2	6	9	12	14	21	}

$x = 20$

Case : P_1 P_2

P_1	P_2	$ar[P_1] + ar[P_2]$	$\|sum - x\|$	update
0	0	3 + 2 = 5	|5 - 20| = 15	if sum < x Inc sum

Inc P_1 or P_2 will Inc sum, ambiguity, Invalid Initialization.

	0	1	2	3	4	5	
A[] = {	3	7	8	10	11	14	}
B[] = {	2	6	9	12	14	21	}

$x = 20$

Case : P_1 P_2

P_1	P_2	$ar[P_1] + ar[P_2]$	$\|sum - x\|$	update
0	5	3 + 21 = 24	|24 - 20| = 4	if (sum > x) Dec sum P_2--
0	4	3 + 14 = 17	|17 - 20| = 3	if (sum < x) Inc sum P_1++;

Note:
```
while (P1 < N && P2 >= 0) {
    | Repeat above approach
}
```

```
int min_diff(vector<int> &A, vector<int> &B, int x){

    int N= A.size(), M = B.size(), ans=0;
    int i = 0, j = M-1;

    while( i< N && j>=0){
        ans = min( ans, abs(A[i]+B[j] - x));
        if(A[i]+B[j] > x){ j--;}
        else{ i++;}
    }
    return ans;
}
```

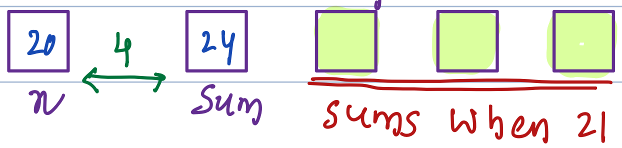
#Note: why should above logic work

	0	1	2	3	4	5	
A[] = {	3	7	8	10	11	14	}
B[] = {	2	6	9	12	14	21	}

arr[P_1] + arr[P_2]

P_1 P_2 Sum #obs:

0 5 24 here sum > x 21 paired with smallest element is 24.



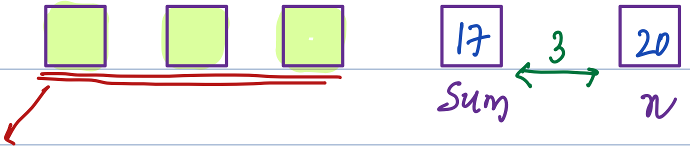
sums when 21 paired with other ele > 24.

That means diff > 4.

#Con: With 21 as element you cannot get a lesser differ than 4.

hence we can discard 21, update P_2.

0 4 17 here sum < x 3 paired with greatest ele is 17



Sums when 21 paired with other ele < 17, That means diff < 3.

#Con: With 17 as element you cannot get a lesser difference than 3, hence we discard 17, update P_1.