arr[] = { 5, 10, 3, 2, 50, 80 } B = 78

Brute force

```
for(int i=0; i<n; i++)
{
    for(int j=i+1; j<n; j++)
    {
        if (arr[i] - arr[j] == B)
            return true;
        if (arr[j] - arr[i] == B)
            return true;
    }
}
```

TC : O(N²)

SC : O(1)

arr[] = {5, 10, 3, 2, 50, 80} ~~B = 78~~

sorting ↓

arr[] : {2, 3, 5, 10, 50, 80} B = 45

ei si

if

```
if (arr[ei] - arr[si] > B)
        ei--;
else
        si++;
```

X

arr[] = { 5, 10, 3, 2, 50, 80 }

TC: O(N)

SC: O(N)

B = 78

(x, y)

let x = 80

y = 80 − 78

= 2

let 10 = x

y = 10 − 78

= − 68

let 10 = y

x = 78 + 10 = 88

$x - y = B$

(80, 2)

let 5 = x

y = 5 − 78

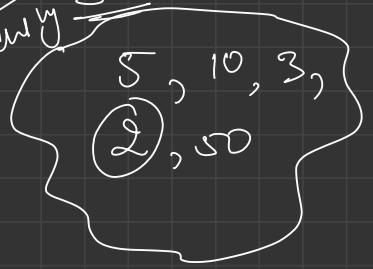
= − 73

$y = x - B$, $x = B + y$

O(1) searching

insertion O(1)

search & get



let 3 = x

y = 3 − 78

= − 75

let 3 = y

x = 78 + 3 = 81

let 5 = y

x = 73 + 5 = 78

let x = 50

y = − 28

let y = 50

x = 128

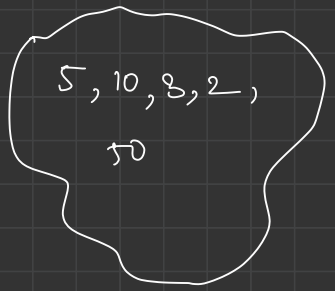
let 2 = x, y = − 76

2 = y, x = 80

```java
HashSet<Integer> mySet = new HashSet<>();

for (int ele : A) {
    // case 1
    int x = ele;
    int y = x - B;
    if (mySet.contains(y) == true) {
        return 1;
    }

    // case 2
    y = ele;
    x = B + y;
    if (mySet.contains(x) == true) {
        return 1;
    }

    // add myself for further reference of x, y
    mySet.add(ele);
}

return 0;
```

arr[] = { 5, 10, 3, 2, 50, 80 }

B = 78

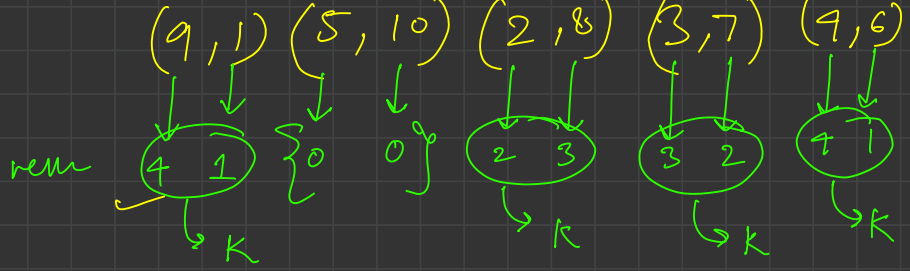


TC : O(N)

SC : O(N)

Array Pair divisible by k.

arr[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 } K = 5



(4)
(K-rem)

rem	freq
1	~~1~~ 2
2	~~1~~ 2
3	~~1~~ 2
4	~~1~~ 2
0	~~1~~ 2

{make pair}

neutralise each comp. rem.

```
for (int rem : remFreqMap.keySet()) {
    // edge case of rem = 0
    if (rem == 0) {
        // no need to neutralize
        continue;
    }

    int compRem = k - rem;
    if (remFreqMap.containsKey(compRem) == false) {
        return false;
    } else if (remFreqMap.get(rem) != remFreqMap.get(compRem)) {
        return false;
    }
}

return true;
```

rem	freq
0	4
2	2
1	4
3	2
4	4

k = 5

TC: O(N) + O(N) = O(N)

SC: O(K)

⇒ {remFreqMap}

Largest Subarray with zero Sum

arr[]: {15, -2, 2, -8, 1, 7, 10, 23}

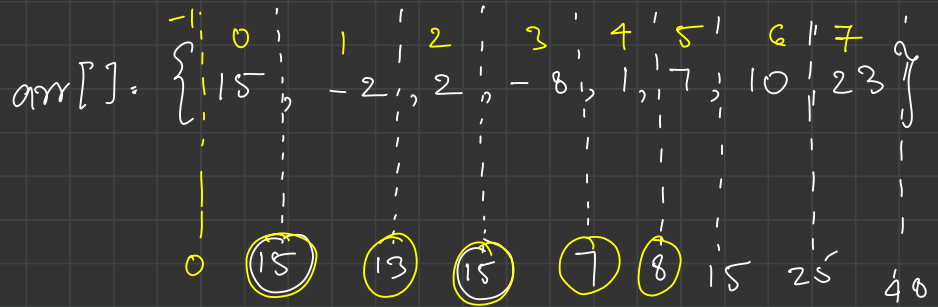
Brute force

↳ get each sub-array with sum

↳ max len

TC: $O(N^2)$

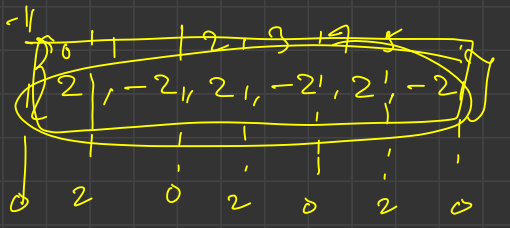
SC: $O(1)$



first occ. index

15 → 0
13 → 1
7 → 3
8 → 4

2 - 0 = 2
5 - 0 = 5
→ subarray with zero sum



5 - (-1) : 6

Equilibrium Index

arr[] = { 9, 3, 7, 6, 8, 1, 10 }

totalSum = 44

lsum = 9 | 12 | 19

lsum + val + rsum = totalSum

rsum = total − val − lsum

= 44 − 6 − 19

= 19

SC: O(1)

Subarray Sum equal to K

arr[]: {10, 2, -2, -20, 10} , K = 10

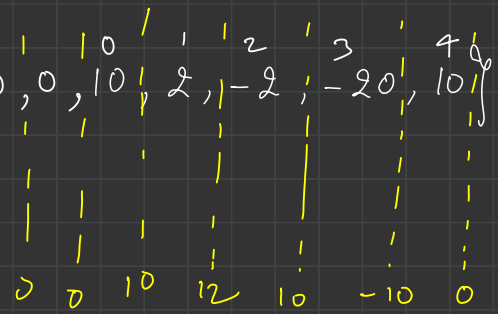
3 subarrays

Brute force

{ get all subarray and check sum == K;

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

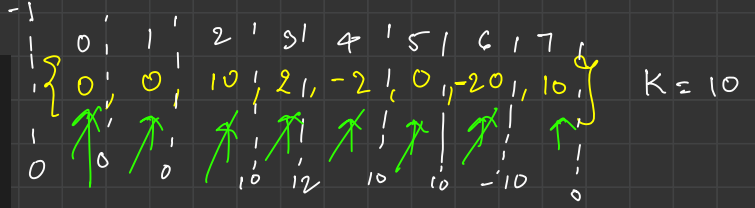
arr[] = { 0, 0, 10, 2, -2, -20, 10 } , K = 10



① Map (currsum, index)

② y = (currSum − K)

```java
class Solution {
    static int solve(int N, int[] Arr, int K) {
        // Write your code here
        int result = 0;

        // runningSum, freq
        HashMap<Integer, Integer> map = new HashMap<>();
        map.put(key: 0, value: 1);

        int runningSum = 0;
        for (int num : Arr) {
            runningSum += num;

            int x = runningSum;
            int y = runningSum - K;

            if (map.containsKey(y) == true) {
                result += map.get(y);
            }

            map.put(runningSum, map.getOrDefault(runningSum, defaultValue: 0) + 1);
        }

        return result;
    }
}
```



runningSum	freq
0	~~1~~ ~~2~~ ~~3~~ 4
10	~~1~~ ~~2~~ 3
12	1
-10	1

3 + 3 + 3 + 1

= 10

TC: O(N) &
SC: O(N)