Hashing 2

Question: Given array of elements and k, check if there exists a pair (i, j) such that arr[i] + arr[j] = k and i != j

0	1	2	3	4	5	6	7	8
8	9	1	-2	4	5	11	-6	4

arr = [8, 9, 1, -2, 4, 5, 11, -6, 4]

k = 6 - True

k = 22 - False

k = 8 - True

Brute Force Approach:

- Iterate over all pairs
- Check if a pair exists whose sum = k.

T.C = $O(N^2)$ S.C = $O(1)$

Idea: Using Hashset.

0	1	2	3	4	5	6	7	8
8	9	1	-2	4	5	11	-6	4

arr = [8, 9, 1, -2, 4, 5, 11, -6, 4]

k = 8

arr[i]	arr[j]
8	0
9	-1
1	7
-2	10
4	4 → True

HS = {8, 9, 1, -2, 4, 5, 11, -6}

T.C = O(N) S.C = O(N)

Edge Case: arr = [5, 7, 9, 2, 3]

k = 4

arr[i]	arr[j]
5	-1
7	-3
9	-5
2	2 → True

With the fix:

arr = [5, 7, 9, 2, 3, -1]

k = 4

arr[i]	arr[j]
5	-1
7	-3
9	-5
2	2 → False
3	1
-1	5 → True

HS = {5, 7, 9, 2, 3}

First check if element is present in HS and then add it to get the correct result.

Using Hashmap:

arr = [5, 7, 9, 2, 5]

k = 4

HM = { 5 : 2,
7 : 1,
9 : 1,
2 : 1, }

arr[i]	arr[j]
5	-1
7	-3
9	-5
2	2 → False
5	-1

```
func checkPairSum(int[] arr, int k) {
    hm = {}
    for (int i = 0; i < arr.length; i++) {
        co_pair = k - arr[i];
        if (hm.search(co_pair)) return true;
        if (hm.search(arr[i])) hm[arr[i]]++;
        else hm.insert(arr[i], 1);
    }
    return false;
}
```

T.C = O(N) S.C = O(N)

Question: Given an array of elements, count the number of pairs such that arr[i] + arr[j] = k and i != j

```
          0   1   2   3   4   5
arr = [ 3 , 5 , 1 , 2 , 1 , 2 ]    k = 3
```

O/P = (2, 3) (3, 4) (4, 5) (2, 5) i < j

= 4

arr[i]	Co-pair	Ans
3	0	
5	-2	
1	2	
2	1	+1
1	2	+1
2	1	+2

HM = { 3 : 1 ,
5 : 1 ,
1 : ~~1~~ , 2
2 : 1 }

```
func countPairs (int[] arr , int k){
    hm = {};
    ans = 0;
    for (int i=0; i<arr.length ; i++){
        co_pair = k - arr[i];
        if (hm.search (co_pair))  ans+=hm[co_pair];
        if (hm.search (arr[i]))  hm[arr[i]]++;
        else  hm.insert (arr[i] , 1);
    }
    return ans;   // For i<j , else ans*2;
}
```

arr[i]	co-pair	ans
3	0	0
5	-2	0
1	2	0
2	1	0+1
1	2	1+1
2	1	2+2

hm = { 3 : 1,
5 : 1,
1 : 2,
2 : 2

T.C = O(N)

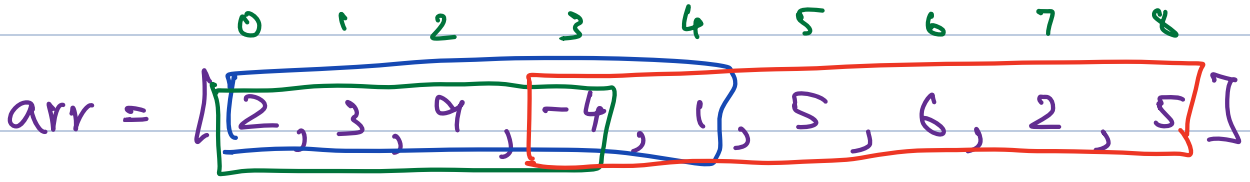
S.C = O(N)

0	1	2	3	4	5
3	5	1	2	1	2

arr = [3, 5, 1, 2, 1, 2]

k = 3

Question: Given an array of elements, check if there exists a subarray with sum = k.



k = 11 → True

k = 10 → True

k = 15 → True

Brute force Approach:

- Go over each subarray sum
- Check if sum == k, return true.

T.C = $O(N^2)$

S.C = $O(1)$

Quiz: A = [5, 10, 20, 100, 105] k = 110

false

$PF[i] - PF[j] = K \Rightarrow PF[j] = PF[i] - K$

↳ Co-pair

Optimised:

	0	1	2	3	4	5	6	7	8
arr =	2	3	9	-4	1	5	6	2	5

PFSum = [2, 5, 14, 10, 11, 16, 22, 24, 29]

PF[i]	Co-pair → PF[i] - K	K = 15	Bank Balance
2	-13	HM = {2:1,	1st Jan → 5 L
5	-10	5:1,	1st April → 8 L
14	-1	14:1,	
. . .			
29	14 → True		

```
func subarraySumK(int[] arr, int k){
    hm = {}
    PF[N]  // TODO
    for(int i=0; i<PF.length; i++){
        co_pair = PF[i] - k;
        if(hm.search(co_pair)) return true;
        if(hm.search(arr[i])) hm[arr[i]]++;
        else hm.insert(arr[i], 1);
    }
    return false;
}
```

T.C = O(N)

S.C = O(N)

Edge case: arr = [2, 3, 9, -4, 1] k = 11

PFSum = [2, 5, 14, 10, 11]

↳ Co_pair = 0

↓

Not existing in the HM.

↓

False

To fix: Add 0 initially into HM

Alternatively: Check if PF[i] == k while creating PF itself

```
func subarraySumK (int[] arr, int k){
    hm = {0:1}
    PF[N]  // TODO
    for (int i=0; i<PF.length; i++){
        co_pair = PF[i] - k;
        if (hm.search (co_pair)) return true;
        if (hm.search (arr[i])) hm[arr[i]]++;
        else hm.insert (arr[i], 1);
    }
    return false;
}
```

T.C = O(N)

S.C = O(N)

To get count of subarrays with sum = k

⇒ Add freq to ans (Same as previous problem)

To get the actual subarray:

```
func subarraySumK (int[] arr, int k){
    hm = {0:-1}
    PF[N]  // TODO
    for (int i=0; i<PF.length; i++){
        co_pair = PF[i] - k;
        if (hm.search (co_pair))
            return [hm[PF[i]]+1, i];
        if (hm.search (arr[i])) hm[PF[i]] = i;
        else hm.insert (arr[i], i);
    }
    return -1;
}
```

T.C = O(N)

S.C = O(N)

Question: Distinct elements in every window of size k.

$N = 7$

$k = 4$

0	1	2	3	4	5	6
1	2	1	3	4	2	3

arr = [1, 2, 1, 3, 4, 2, 3]

O/P = [3, 4, 4, 3]

Idea 1:

- for every subarray of size k, add elements to hashset
- Return the size of hashset.

```
func distinctK(int[] arr, int k){
    ans = [];
    for(int i=0; i<=N-k; i++){
        hs = {}
        for(int j=0; j<k; j++)
            hs.insert(arr[i+j]);
        ans.append(hm.size());
    }
    return ans;
}
```

$T.C = O((N-k+1)*k) = O(N*k)$

$S.C = O(k)$

Optimised Idea:

```
          0  1  2  3  4  5  6
arr = [ 1, 2, 1, 3, 4, 2, 3 ]
```

↳ hs = {1, 2, 3} +4 -1

= {2, 3, 4} → 3

Instead, store frequency map.

```
          0  1  2  3  4  5  6
arr = [ 1, 2, 1, 3, 4, 2, 3 ]
```

↳ hm = {1:2, 2:1, 3:1} +4 -1

= {1:1, 2:1, 3:1, 4:1}

+2 -2

= {1:1, 2:1, 3:1, 4:1}

+3 -1

= {1:0, 2:1, 3:2, 4:1}

↳ Size = 4

Additionally, if hm[i] == 0, then remove hm[i]

```
func distinctK(int[] arr, int k) {
    hm = {};
    // For the first subarray of size k.
    for(int i=0; i<k; i++){    // O(k)
        if(hm.search(arr[i]) hm[arr[i]]++;
        else hm.insert(arr[i], 1);
    }
    ans = [hm.size()];
    // Start from 2nd subarray
    for(int i=1; i<=N-k; i++){  // O(N-k)
        old_ele = arr[i-1];
        new_ele = arr[i+k-1];
        // Insert new_ele
        if(hm.search(new_ele)) hm[new_ele]++;
        else hm.insert(new_ele, 1);
        // Remove old_ele
        hm[old_ele]--;
        if(hm[old_ele]==0) hm.remove(old_ele);
        // Insert size of hm in ans
        ans.append(hm.size());
    } return ans; }
```

$T.C = O(N - k + k) \simeq O(N)$

$S.C = O(k)$

Doubts

* Given Array of element, calculate sum of all A[i] % A[j]

	A[0]	A[1]	A[2]
A[0]	11%11	11%17	11%100
A[1]	17%11	17%17	17%100
A[2]	100%11	100%17	100%100

A = [11, 17, 100]

N <= 10^5

A[i] <= 1000

```
hm = {Arr[i]}
sum = 0
for (int i=1 ; i<=1000; i++){
    for (int j=1 ; j<=1000; j++){
        if( i in hm && j in hm ){
            if(i!=j){
                count_i = hm[i]
                count_j = hm[j]
                sum = count_i * count_j * (i%j)
            }
        }
    }
}
```