Hashing

1. **Count Frequency** - Use dictionary
2. **Find a pair with sum x** **in array**
3. Sort and use **2 pointers (O(nlogn))**
4. **Use dictionary (O(n))**

**-> Take d = {}**

**-> loop through array. I+j = x; j = x-i**

**-> Chck if x-i in d. If yes return the pair. Else add i to d.**

**d = {}**

**for i in arr:**

**If x-i in d:**

**return [i,x-i]**

**d[i] = 1**

**3) Find whether an array is subset of another**

Use dictionary

-> make frequency array of arr1 called map and then loop through arr2 and if element is in map then reduce map[i] -= 1 and delete element from arr2.

-> if in the end map elements have all 0 in frequency then arr1 is subset of arr2.

4) **Maximum distance between 2 same elements in array**

Input : arr[] = {3, 2, 1, 2, 1, 4, 5, 8, 6, 7, 4, 2}

Output: 10

// maximum distance for 2 is 11-1 = 10

// maximum distance for 1 is 4-2 = 2

// maximum distance for 4 is 10-5 = 5

Use dictionary

-> if element in dict then ans = max(ans, i-d[li[i]])

-> else add element to dict; d[li[i]] = i

Ans = -1

d = {}

for i in range(len(li)):

if li[i] in d:

ans = max(ans, i-d[li[i]])

d[li[i]] = i

print(ans)

5) **Min. operations to make all elements equal**

-> result is len(arr) - (highest freq. element)

def func(arr):

d = {}

for i in arr:

if i in d:

d[i] += 1

else:

d[i] = 1

return len(arr) - max(d.values())

6) Check if a given array contains duplicate elements within k distance from each other

Input: k = 3, arr[] = {1, 2, 3, 1, 4, 5}

Output: true

1 is repeated at distance 3.

def func(arr, k):

d = {}

for i in range(len(arr)):

if arr[i] in d:

if abs(i - d[arr[i]]) == k:

return True

d[arr[i]] = i

else:

d[arr[i]] = i

return False

7) **Sum of elements in an array with frequencies greater than or equal to that element**

def func(arr, k):

d = {}

for i in range(len(arr)):

if arr[i] in d:

d[arr[i]] += 1

else:

d[arr[i]] = 1

s = 0

for i in d.keys():

if d[i] >= i:

s += i

return s

**8) First unique character of a string**

-> Make a dict and store all indexes of elements and return the element with only single element and lowest index.

class Solution:

def firstUniqChar(self, s: str) -> int:

m = {}

for i in range(len(s)):

if s[i] in m:

m[s[i]].append(i)

else:

m[s[i]] = [i]

c = 2\*\*31-1

print(m)

for i in m.keys():

if len(m[i]) == 1:

c = min(c, m[i][0])

if c == 2\*\*31-1: return -1

return c

9) [**Count pairs with given sum**](https://www.geeksforgeeks.org/count-pairs-with-given-sum/)

-> Create freq map and loop through array.  
-> if k-i in d: count += d[k-i] as it can form as many as pair as count

def getPairsCount(self, arr, n, k):

d = Counter()

c = 0

for i in arr:

c += d.get(k-i, 0)

d[i] += 1

return c

10) **Find Common Characters**

-> Use dictionary

Input: words = ["bella","label","roller"]

Output: ["e","l","l"]

**11)** [Count Number of Pairs With Absolute Difference K](https://leetcode.com/problems/count-number-of-pairs-with-absolute-difference-k/)

-> Similar to question 9

**class Solution:**

**def countKDifference(self, nums: List[int], k: int) -> int:**

**d = {}**

**c = 0**

**for i in nums:**

**c += d.get(i-k, 0) + d.get(i+k, 0)**

**if i in d:**

**d[i] += 1**

**else:**

**d[i] = 1**

**return c**

**12)** [**Number of Pairs of Strings With Concatenation Equal to Target**](https://leetcode.com/problems/number-of-pairs-of-strings-with-concatenation-equal-to-target/)

Brute -

class Solution:

def numOfPairs(self, nums: List[str], target: str) -> int:

ans=0

for i in range(len(nums)):

for j in range(i+1,len(nums)):

if nums[i]+nums[j]==target:

ans+=1

if nums[j]+nums[i]==target:

ans+=1

return ans

Optimal - -> Make a freq dict.  
Loop through array and replace i from target

If i == target-i: cnt += d[target-i]-1 as we want to omit extra occurance

else: cnt += d[target-i]

class Solution:

def numOfPairs(self, nums: List[str], target: str) -> int:

ans=0

cnt = Counter(nums)

for i in nums:

x = target.replace(i, "", 1)

#print(x)

if x in cnt:

if x == i:

ans += cnt[x] - 1

else:

if i+x == target:

ans += cnt[x]

return ans

13) **Count maximum points on same line**

-> calc. Slope of each point and store in map. Elements with same slope are in same line.  
-> return the slope with maximum points.

class Solution:

def slopeMaxElements(self, points: List[int], k: int) -> int:

d = {}

for i in points:

x = i[0]

y = i[1]

slope = (y-0)/(x-0)

if slope in d:

d[slope] += 1

else:

d[slope] = 1

return max(d.values())

**14) Smallest Subarray with given sum**

def subArraylen(arr, n, K):

mp = defaultdict( )

currPrefixSum = 0

result = sys.maxsize

for i in range(n):

currPrefixSum += arr[i]

if(currPrefixSum == K):

currLen = i + 1

result = min(result, currLen)

requirePrefixSum = currPrefixSum - K

if(requirePrefixSum in mp.keys()):

foundIdx = mp[requirePrefixSum]

currIdx = i

result = min(result, currIdx - foundIdx)

mp[currPrefixSum] = i

return result

**15)** Triplet Sum in Array (Better with 2 pointers)

(2 Pointer Approach TC - O(n\*\*2))

**#User function Template for python3**

**class Solution:**

**#Function to find if there exists a triplet in the**

**#array A[] which sums up to X.**

**def find3Numbers(self,A, n, X):**

**# Your Code Here**

**A.sort()**

**for i in range(n):**

**l = i+1**

**r = n-1**

**while l<r:**

**s = A[i]+A[l]+A[r]**

**if s == X:**

**return 1**

**elif s > X:**

**r -= 1**

**else:**

**l += 1**

**return 0**

(Hashmap TC - (O(n\*n))

->Run loop in from i -> 0-n

->take a set and make curr\_sum = sum-A[i] so we just need to find a pair with sum as curr\_sum and again run a loop from i+1-n.

->see if curr\_sum-A[j] in set. If yes return true

->add A[j] to set.

def find3Numbers(A, arr\_size, sum):

for i in range(0, arr\_size-1):

# Find pair in subarray A[i + 1..n-1]

# with sum equal to sum - A[i]

s = set()

curr\_sum = sum - A[i]

for j in range(i + 1, arr\_size):

if (curr\_sum - A[j]) in s:

print("Triplet is", A[i],

", ", A[j], ", ", curr\_sum-A[j])

return True

s.add(A[j])

return False

**16) Subarray sum equals k**

-> take sums = 0 and loop through array

-> at every iteration add element to sums

If k != 0 then make sums = sums%k

If sums is there in dict return True if i-d[sums] > 1 as we want a subarray with length 2 or more

else make d[sums] = i

Example:

nums = [23,2,4], k = 6

Lets walk through the code with the example.

(i=0) : sums = 23 => 23%6 => (sums = 5)

(i=1) : sums = 5+2=7 => 7%6 => (sums = 1)

(i=2) : sums = 1+4=5 => 5%6 => (sums = 5)

We have encountered the same sums(remainder) again which means we have the subarray of sums%k = 0.

But, there's another aspect to this problem. The subarray must have a minimum size of 2.

That is why we check if (i - d[sums])>1.

In the above example, this if loop is executed when (i=2) and (d[sums]=1).

In other words, the same remainder(sums=5) has been encountered twice and then we check for the respective difference in indices.

Counter example to understand this. Lets take nums = [23,6], k = 6

(i=0) : sums = 23 => 23%6 => (sums = 5)

(i=1) : sums = 5+6=11 => 11%6 => (sums = 5)

class Solution:

def checkSubarraySum(self, nums: List[int], k: int) -> bool:

curr = 0

d = {0: -1}

for i in range(len(nums)):

curr += nums[i]

if k != 0:

curr = curr%k

if curr in d:

if i-d[curr] > 1:

return True

else:

d[curr] = i

return False

**17) Number of subarray with sum 0**

First of all, the basic idea behind this code is that, whenever sums has increased by a value of k, we've found a subarray of sums=k.

I'll also explain why we need to initialise 0 in the hashmap.

Example: Let's say our elements are [1,2,1,3] and k = 3.

and our corresponding running sums = [1,3,4,7]

Now, if you notice the running sums array, from 1->4, there is increase of k and from 4->7, there is an increase of k. So, we've found 2 subarrays of sums=k.

But, if you look at the original array, there are 3 subarrays of sums==k. Now, you'll understand why 0 comes in the picture.

In the above example, 4-1=k and 7-4=k. Hence, we concluded that there are 2 subarrays.

However, if sums==k, it should've been 3-0=k. But 0 is not present in the array. To account for this case, we include the 0.

Now the modified sums array will look like [0,1,3,4,7]. Now, try to see for the increase of k.

1. 0->3
2. 1->4
3. 4->7  
   Hence, 3 sub arrays of sums=k

This clarified some confusions I had while doing this problem.

class Solution(object):

def subarraySum(self, nums, k):

"""

:type nums: List[int]

:type k: int

:rtype: int

"""

count = 0

sums = 0

d = dict()

d[0] = 1

for i in range(len(nums)):

sums += nums[i]

count += d.get(sums-k,0)

d[sums] = d.get(sums,0) + 1

return(count)