# **Hashing and Prefix Sum**

1. Check if there are any two equal numbers in an array at a distance less than or equal to k

* **Logic:** 
  1. Take a hashmap and run a loop on array
  2. First check for element is already present or not
  3. If present then simply check of distance as prev index – curr index is less than or equal to k then return true.
  4. Else add to map as arr[i] as key and value as index i
  5. If for loop doesn’t return true then simply return false
* **Code:**
* <https://onecompiler.com/java/42s2teaw4>

1. Count all the (i,j) pairs such that b[i] + b[j] == k or b[i] – b[j] == k (i<j)

* **Logic:** 
  1. Take a hashmap and run a loop on array
  2. First calculate complement as k – b[j] or k + b[j]
  3. Check complement is present in map or not if present then count++
  4. Add that element into map
* **Code:**
  1. <https://onecompiler.com/java/42s2z6nqu>

1. Count all the (i,j) pairs such that abs(b[i] - b[j]) = k

* **Logic:** 
  1. Take a hashmap and run a loop on array
  2. B[i] = b[j] – k and b[i] = b[j] + k (k!=0) add freq to count
* **Code:**

* 1. <https://onecompiler.com/java/42s32xy5x>

1. Find Sum of Range [l…..r] where (l<=r) using prefix sum

* **Logic:** 
  1. Create preSum array pre[i] = pre[i-1] + arr[i]
  2. Return pre[r] – pre[l-1] for sum of l to r (handle for l == 0)
* **Code:**
  1. <https://onecompiler.com/java/42s36qpt2>

1. Find count of subarray having sum == k

* **Logic:** 
  1. Use prefix sum and hashing (initially add [0, 1] in hashmap to handle edge cases)
  2. If curr\_sum – k is present in hashmap then count += freq[curr\_sum – k]
  3. Put curr\_sum in hashmap and increase freq if have already
* **Code:**
  1. [42sbdbdfd - Java - OneCompiler](https://onecompiler.com/java/42sbdbdfd)

1. Find largest/smallest length of subarray having sum == k

* **Logic:** 
  1. Use prefix sum and hashing
  2. Run a loop and calculate pre\_sum for curr index
  3. Check if pre\_sum == k then maxLen = i+1
  4. Else if pre\_sum – k is present in hashMap then calculate maxLen = max( i (curr\_len ) – mp.get(pres\_sum – k) (last\_len))
  5. And if preSum is already present in hashMap then don’t update it else put it with curr index
* **Code:**
  1. [42sbg6bnu - Java - OneCompiler](https://onecompiler.com/java/42sbg6bnu)

1. Valid Anagram Leetcode

* **Logic:** 
  1. Use array of size 26 as hashmap
  2. Run a loop on str1 and update the freq count of all char
  3. Run a another loop on str2 and decrease the freq count of all char in same hash array
  4. Run a loop on hash array check every value is 0 or not.
* **Code:**
  1. [Valid Anagram - LeetCode](https://leetcode.com/problems/valid-anagram/submissions/1391786396/)

1. Find two subarrays of maximum sum which are not overlapping each other (kadane’s algo)

* **Logic:** 
  1. First calculate prefixMaxSum array from front using kadane’s algo and after that calculate suffixMaxSum array from backward using kadane’s alg
  2. Take maxSum variable = 0
  3. Run a loop from 0 to n-1 and calculate maxSum = max( maxSum , prefixMaxSum[i] + suffixMaxSum[i] )
  4. Return maxSum
* **Code:**
  1. [42shqq6uv - Java - OneCompiler](https://onecompiler.com/java/42shqq6uv)