Hashing:

Hashing is a technique used in computer science to map data of arbitrary size to fixed-size values. It involves applying a hash function to the input data, which generates a unique hash code or hash value. Hashing is commonly used for data storage, retrieval, and quick lookup operations.

- The primary purpose of hashing is to quickly determine the location or index of stored data in a data structure called a hash table or hash map.

Data Structures that use Hashing Technique

Set

A set is a collection of unique elements where the order of elements is not significant. It ensures that no duplicate elements are stored, providing efficient membership checks and eliminating redundancy.

In various programming languages, sets are typically implemented using data structures such as hash sets or tree sets.

Map

A map is a key-value pair data structure that uses hashing for efficient key-based operations. It allows for quick insertion, deletion, and retrieval of elements based on their associated keys.

Maps are also known as associative arrays, hash tables, or dictionaries in different programming languages.

Equivalents in Different Programming Languages:

Language	Set Equivalent	Map Equivalent
Java	HashSet	HashMap
C++	Unordered_Set	Unordered_Map
Python	Dictionary	Dictionary
JS/Ruby	Set	Map
C#	HashSet	Dictionary

Understanding hashing, sets, and hash maps, along with their equivalents in different programming languages, is essential for efficient data storage, retrieval, and manipulation in various software applications.

Problems :

Q- Count the number of distinct elements in an array.

Example:
Input: [2, 3, 2, 3, 2, 5, 7, 7]
Output: 4

Explanation: There are 4 different elements: **2, 3, 5 ,7**

Solution

```
function countDistinctElements(array):
      // Create an empty set
      set = new empty set

      // Traverse the array
      for element in array:
      // Add each element to the set
      set.add(element)

return set.size
```

Q. Given an array containing both positive and negative integers, we have to find the length of the longest subarray with the sum of all elements equal to zero.

Example

```
Input: [19, -4, 4, -2, 7, -5]
Output: 5
```

Explanation:

The following subarrays sum to zero:
{-4, 4} , {-2, 7, -5}, {-4, 4, -2, 7, -5}
So, the longest among the above three is the last one with length of 5.

Brute force :-

One way to solve this problem is to consider all possible subarrays and calculate their sum. If the sum of a subarray is zero, we update the maximum length. We repeat this process for all subarrays and return the maximum length found.

Optimal Approach using Hashmap:

We can optimize the solution using a hashmap to store the sum of elements encountered so far and their corresponding indices. We calculate the prefix sum by adding each element to the sum variable. If the current sum is zero or has been encountered before, it means the subarray from the previous occurrence of that sum to the current element has a sum of zero. We update the maximum length if necessary.

```
function findMaxLength(arr):
      n = length of arr
      maxLength = 0
      sum = 0
      hashMap = empty hashmap of mapping integer to integer

      for i = 0 to n-1:
            sum += arr[i]

            if sum == 0:
                  maxLength = i + 1
            if sum in hashMap:
                  maxLength = max(maxLength, i - hashMap[sum])
            else:
                  hashMap[sum] = i
return maxLength
```

Q. Given an array, find the length of the largest sub-sequence which can be re-arranged to form a sequence of consecutive numbers.

Example

Input: [100, 4, 100, 3, 1, 2]
Output: 4

Explanation:
The largest sub-sequence that can be rearranged to obtain a sequence of consecutive numbers is 4, 3, 1, 2 -> 1, 2, 3, 4.

Solution

Brute Force: The brute force algorithm for finding the length of the largest subsequence that can be rearranged to form a sequence of consecutive numbers does not employ any advanced techniques. It simply examines each number in the given array and tries to count as high as possible from that number, using only the numbers present in the array. Whenever it encounters a number that is not present in the array, it records the length of the current sequence if it is greater than the previously recorded maximum length. The algorithm exhaustively explores all possibilities, making it guaranteed to find the optimal solution.

TC: $O(n^3)$
SC: O(1)

Optimized Solution:
This involves using the concept of Hashing.

- Create a Set to store all the numbers from the input array.
- Iterate through the array and add each element to the Set.
- Iterate through the elements in the HashSet. For each element, check if its previous number (i-1) is not present in the HashSet. If the previous number is not present, it means the current number is the starting point of a consecutive subsequence.
- Start counting the length of the subsequence from the current number by incrementing a length and the current number by 1.
- Continue incrementing the currentNumber and counting the length until the consecutive subsequence ends (i.e., the next number is not present in the HashSet).

- Update the maxLen if the current length (len) is greater than the previous maximum.
- Repeat steps the steps for all elements in the HashSet.
- Return the maxLen as the length of the longest consecutive subsequence.

```
function longestConsecutive(nums):
      set = new set of integers
      for i in nums:
            set.add(i)

      maxLen = 0
      for i in set:
            if set does not contain (i-1):
                  len = 0
                  currentNumber = i
                  while set contains currentNumber:
                        len++
                        currentNumber++

                        maxLen = max(maxLen, len)

      return maxLen
```

Revision Video -

https://www.scaler.com/meetings/dsa-hashing-1-revision-15/j/a/pTY3NLSQ7$btw2Zh_8q1ucNlu0vpY2Bh