Assignment-49 **A Job Ready Bootcamp in c++,DSA and IOT**

unordered_set

1. Count of distinct pair sums in a given Array arr[] of size N, the task is to find the total number of unique pair sums possible from the array elements.

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to count distinct pairs
// in array whose sum equal to K
int cntDisPairs(int arr[], int N, int K)
{
    // Stores count of distinct pairs
    // whose sum equal to K
    int cntPairs = 0;

    // Sort the array
    sort(arr, arr + N);

    // Stores index of
    // the left pointer
    int i = 0;

    // Stores index of
    // the right pointer
    int j = N - 1;

    // Calculate count of distinct
    // pairs whose sum equal to K
    while (i < j)
    {

        // If sum of current pair
        // is equal to K
        if (arr[i] + arr[j] == K)
        {

            // Remove consecutive duplicate
            // array elements
            while (i < j && arr[i] == arr[i + 1])
            {
                // Update i
                i++;
            }

            // Remove consecutive duplicate
            // array elements
            while (i < j && arr[j] == arr[j - 1])
            {
                // Update j
                j--;
            }

            // Update cntPairs
            cntPairs += 1;

            // Update i
            i++;

            // Update j
            j--;
        }
```

```
        // if sum of current pair
        // less than K
        else if (arr[i] + arr[j] < K)
        {
            // Update i
            i++;
        }
        else
        {
            // Update j
            j--;
        }
    }
    return cntPairs;
}

// Driver Code
int main()
{
    int arr[] = { 5, 6, 5, 7, 7, 8 };
    int N = sizeof(arr) / sizeof(arr[0]);
    int K = 13;
    cout << cntDisPairs(arr, N, K);
}
```

```
Output:
2
```

2. C++ Program to Print all triplets in sorted array that form AP(or Arithmetic Progression) Example..Input : arr[] = { 2, 6, 9, 12, 17, 22, 31, 32, 35, 42 };
   Output :
   6 9 12
   2 12 22
   12 17 22
   2 17 32
   12 22 32
   9 22 35
   2 22 42
   22 32 42

```
// C++ program to print all triplets in given
// array that form Arithmetic Progression
// C++ program to print all triplets in given
// array that form Arithmetic Progression
#include <bits/stdc++.h>
using namespace std;

// Function to print all triplets in
// given sorted array that forms AP
void printAllAPTriplets(int arr[], int n)
{
    unordered_set<int> s;
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            // Use hash to find if there is
            // a previous element with difference
            // equal to arr[j] - arr[i]
            int diff = arr[j] - arr[i];
            if (s.find(arr[i] - diff) != s.end())
```

```
                cout << arr[i] - diff << " " << arr[i]
                    << " " << arr[j] << endl;
        }
        s.insert(arr[i]);
    }
}

// Driver code
int main()
{
    int arr[] = { 2, 6, 9, 12, 17, 22, 31, 32, 35, 42 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printAllAPTriplets(arr, n);
    return 0;
}
```

```
Output:
6 9 12
2 12 22
12 17 22
2 17 32
12 22 32
9 22 35
2 22 42
22 32 42
```

3. C++ Program for Number of unique triplets whose XOR is zero.
   Input : a[] = {1, 3, 5, 10, 14, 15};
   Output : 2
   Explanation : {1, 14, 15} and {5, 10, 15} are the
   unique triplets whose XOR is 0.
   {1, 14, 15} and all other combinations of
   1, 14, 15 are considered as 1 only.
   Input : a[] = {4, 7, 5, 8, 3, 9};
   Output : 1
   Explanation : {4, 7, 3} is the only triplet whose XOR is 0

```
// CPP program to count the number of
// unique triplets whose XOR is 0
#include <bits/stdc++.h>
using namespace std;

// function to count the number of
// unique triplets whose xor is 0
int countTriplets(int a[], int n)
{
    // To store values that are present
    unordered_set<int> s;
    for (int i = 0; i < n; i++)
        s.insert(a[i]);

    // stores the count of unique triplets
    int count = 0;

    // traverse for all i, j pairs such that j>i
    for (int i = 0; i < n-1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            // xor of a[i] and a[j]
            int xr = a[i] ^ a[j];
```

```
            // if xr of two numbers is present,
            // then increase the count
            if (s.find(xr) != s.end() && xr != a[i] && xr != a[j])
                count++;
        }
    }
    // returns answer
    return count / 3;
}

// Driver code to test above function
int main()
{
    int a[] = {1, 3, 5, 10, 14, 15};
    int n = sizeof(a) / sizeof(a[0]);
    cout << countTriplets(a, n);
    return 0;
}
```

```
Output:
2
```

4. C++ Program to give two arrays with size n, maximise the first array by using the elements from the second array such that the new array formed contains n greatest but unique elements of both the arrays giving the second array priority (All elements of second array appear before first array). The order of appearance of elements is kept the same in output as in input.

   Examples:

   Input : arr1[] = {2, 4, 3}
   arr2[] = {5, 6, 1}
   Output : 5 6 4
   As 5, 6 and 4 are maximum elements from two arrays giving the second array higher priority. Order of elements is the same in output as in input.

   Input : arr1[] = {7, 4, 8, 0, 1}
   arr2[] = {9, 7, 2, 3, 6}
   Output : 9 7 6 4 8

```
// C++ program to print the maximum elements
// giving second array higher priority
#include <bits/stdc++.h>
using namespace std;

// Compare function used to sort array
// in decreasing order
bool compare(int a, int b)
{
    return a > b;
}

// Function to maximize array elements
void maximizeArray(int arr1[], int arr2[],
                                int n)
{
    // auxiliary array arr3 to store
    // elements of arr1 & arr2
    int arr3[2*n], k = 0;
    for (int i = 0; i < n; i++)
        arr3[k++] = arr1[i];
    for (int i = 0; i < n; i++)
        arr3[k++] = arr2[i];
```

```
    // hash table to store n largest
    // unique elements
    unordered_set<int> hash;

    // sorting arr3 in decreasing order
    sort(arr3, arr3 + 2 * n, compare);

    // finding n largest unique elements
    // from arr3 and storing in hash
    int i = 0;
    while (hash.size() != n)
    {

        // if arr3 element not present in hash,
        // then store this element in hash
        if (hash.find(arr3[i]) == hash.end())
            hash.insert(arr3[i]);

        i++;
    }

    // store that elements of arr2 in arr3
    // that are present in hash
    k = 0;
    for (int i = 0; i < n; i++)
    {
        // if arr2 element is present in hash,
        // store it in arr3
        if (hash.find(arr2[i]) != hash.end())
        {
            arr3[k++] = arr2[i];
            hash.erase(arr2[i]);
        }
    }

    // store that elements of arr1 in arr3
    // that are present in hash
    for (int i = 0; i < n; i++)
    {

        // if arr1 element is present in hash,
        // store it in arr3
        if (hash.find(arr1[i]) != hash.end())
        {
            arr3[k++] = arr1[i];
            hash.erase(arr1[i]);
        }
    }

    // copying 1st n elements of arr3 to arr1
    for (int i = 0; i < n; i++)
        arr1[i] = arr3[i];
}

// Function to print array elements
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; i++)
        cout << arr[i] << " ";
    cout << endl;
}

// Driver Code
```

```
int main()
{
    int array1[] = { 2, 4, 3};
    int array2[] = { 5, 6, 1};
    int size = sizeof(array1) / sizeof(array1[0]);
    maximizeArray(array1, array2, size);
    printArray(array1, size);
}
================================================================================
Output:
5 6 4
```

5. C++ Program to given an array of positive and negative numbers, find if there is a subarray (of size at-least one) with 0 sum.
   Examples :
   Input: {4, 2, -3, 1, 6}
   Output: true
   Explanation:

   There is a subarray with zero sum from index 1 to 3.
   Input: {4, 2, 0, 1, 6}
   Output: true
   Explanation:
   There is a subarray with zero sum from index 2 to 2.

```
// C++ program to find if
// there is a zero sum subarray

#include <bits/stdc++.h>
using namespace std;

bool subArrayExists(int arr[], int N)
{
    unordered_set<int> sumSet;

    // Traverse through array
    // and store prefix sums
    int sum = 0;
    for (int i = 0; i < N; i++)
    {
        sum += arr[i];

        // If prefix sum is 0 or
        // it is already present
        if (sum == 0 || sumSet.find(sum) != sumSet.end())
            return true;

        sumSet.insert(sum);
    }
    return false;
}

// Driver's code
int main()
{
    int arr[] = {4, 2, 0, 1, 6};
    int N = sizeof(arr) / sizeof(arr[0]);

    // Function call
    if (subArrayExists(arr, N))
        cout << "Found a subarray with 0 sum";
```

```
    else
        cout << "No Such Sub Array Exists!";
    return 0;
}
```

```
Output:
Found a subarray with 0 sum
```

6. Given an array arr[] consisting of N positive integers, the task is to find the number of pairs such that the Greatest Common Divisor(GCD) of the pairs is not a prime number. The pair (i, j) and (j, i) are considered the same.
   Examples:
   Input: arr[] ={ 2, 3, 9}
   Output: 10
   Explanation:
   Following are the possible pairs whose GCD is not prime:
   (0, 1): The GCD of arr[0](= 2) and arr[1](= 3) is 1.
   (0, 2): The GCD of arr[0](= 2) and arr[2](= 9) is 1.
   Therefore, the total count of pairs is 2.
   Input: arr[] = {3, 5, 2, 10}
   Output: 4

```
// C++ program for the above approach

#include <bits/stdc++.h>
using namespace std;

// Function to find the prime numbers
void primeSieve(bool* p)
{
    for (int i = 2; i * i <= 1000000; i++)
    {

        // If p[i] is not changed,
        // then it is a prime
        if (p[i] == true)
        {
            // Update all multiples of i
            // as non prime
            for (int j = i * 2; j <= 1000000; j += i)
            {
                p[j] = false;
            }
        }
    }
}

// Function to find GCD of two integers
// a and b
int gcd(int a, int b)
{
    // Base Case
    if (b == 0)
        return a;

    // Find GCD Recursively
    return gcd(b, a % b);
}
```

```
// Function to count the number of
// pairs whose GCD is non prime
int countPairs(int arr[], int n, unordered_set<int> s)
{
    // Stores the count of valid pairs
    int count = 0;

    // Traverse over the array arr[]
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = i + 1; j < n; j++)
        {
            // Calculate the GCD
            int x = gcd(arr[i], arr[j]);

            // Update the count
            if (s.find(x) == s.end())
                count++;
        }
    }

    // Return count
    return count;
}

// Utility Function to find all the prime
// numbers and find all the pairs
void countPairsUtil(int arr[], int n)
{
    // Stores all the prime numbers
    unordered_set<int> s;
    bool p[1000005];
    memset(p, true, sizeof(p));

    // Find all the prime numbers
    primeSieve(p);

    s.insert(2);

    // Insert prime numbers in the
    // unordered set
    for (int i = 3; i <= 1000000; i += 2)
        if (p[i])
            s.insert(i);

    // Find the count of valid pairs
    cout << countPairs(arr, n, s);
}

// Driver Code
int main()
{
    int arr[] = { 3, 5, 2, 10 };
    int N = sizeof(arr) / sizeof(arr[0]);
    countPairsUtil(arr, N);

    return 0;
}
```

==========

Output:

```
4
```

7. Given an array of strings arr[] of size N, the task is to print all the distinct strings present in the given array.
   Examples:
   Input: arr[] = { “Good”, “God”, “Good”, “God”, “god” }
   Output: god Good God

```
#include <bits/stdc++.h>
using namespace std;

// Function to find the distinct strings
// from the given array
void findDisStr(vector<string>& arr, int N)
{
    // Stores distinct strings
    // from the given array
    unordered_set<string> DistString;

    // Traverse the array
    for (int i = 0; i < N; i++)
    {
        // If current string not
        // present into the set
        if (!DistString.count(arr[i]))
        {

            // Insert current string
            // into the set
            DistString.insert(arr[i]);
        }
    }

    // Traverse the set DistString
    for (auto String : DistString)
    {
        // Print distinct string
        cout << String << " ";
    }
}

// Driver Code
int main()
{
    vector<string> arr = { "Akhtar", "For", "Akhtar", "Code", "Coder" };

    // Stores length of the array
    int N = arr.size();

    findDisStr(arr, N);
    return 0;
}
================================================================================
Output:
Coder Code Akhtar For
```

8. Find all matrix elements which are minimum in their row and maximum in their column

```
#include <bits/stdc++.h>
using namespace std;

// Functionto find all the matrix elements
// which are minimum in its row and maximum
// in its column
vector<int> minmaxNumbers(vector<vector<int> >& matrix, vector<int>& res)
{
```

```
    // Initialize unordered set
    unordered_set<int> set;

    // Traverse the matrix
    for (int i = 0; i < matrix.size(); i++)
    {
        int minr = INT_MAX;
        for (int j = 0; j < matrix[i].size(); j++)
        {
            // Update the minimum
            // element of current row
            minr = min(minr, matrix[i][j]);
        }

        // Insert the minimum
        // element of the row
        set.insert(minr);
    }

    for (int j = 0; j < matrix[0].size(); j++)
    {
        int maxc = INT_MIN;

        for (int i = 0; i < matrix.size(); i++)
        {

            // Update the maximum
            // element of current column
            maxc = max(maxc, matrix[i][j]);
        }

        // Checking if it is already present
        // in the unordered_set or not
        if (set.find(maxc) != set.end())
        {
            res.push_back(maxc);
        }
    }

    return res;
}

// Driver Code
int main()
{
    vector<vector<int> > mat = { { 1, 10, 4 }, { 9, 3, 8 }, { 15, 16, 17 } };

    vector<int> ans;

    // Function call
    minmaxNumbers(mat, ans);

    // If no such matrix
    // element is found
    if (ans.size() == 0)
        cout << "-1" << endl;

    for (int i = 0; i < ans.size(); i++)
        cout << ans[i] << endl;

    return 0;
}
```

==========

```
Output:
15
```

9. Given N strings of equal lengths. The strings contain only digits (1 to 9). The task is to count the number of strings that have an index position such that the digit at this index position is greater than the digits at the same index position of all the other strings.
   Examples:
   Input: arr[] = {“223”, “232”, “112”}
   Output: 2
   First digit of the 1st and 2nd strings are the largest.
   Second digit of the string 2nd is the largest.
   Third digit of the string 1st is the largest.
   Input: arr[] = {“999”, “122”, “111”}
   Output: 1

```cpp
#include <bits/stdc++.h>
using namespace std;

// Function to return the count of valid strings
int countStrings(int n, int m, string s[])
{

    // Set to store indices of valid strings
    unordered_set<int> ind;
    for (int j = 0; j < m; j++)
    {
        int mx = 0;

        // Find the maximum digit for current position
        for (int i = 0; i < n; i++)
            mx = max(mx, (int)s[i][j] - '0');

        // Add indices of all the strings in the set
        // that contain maximal digit
        for (int i = 0; i < n; i++)
            if (s[i][j] - '0' == mx)
                ind.insert(i);
    }

    // Return number of strings in the set
    return ind.size();
}

// Driver code
int main()
{
    string s[] = { "223", "232", "112" };
    int m = s[0].length();
    int n = sizeof(s) / sizeof(s[0]);
    cout << countStrings(n, m, s);
}
```

```
Output:
2
```

10. Unordered_set operators in C++ STL(== and !=)

```cpp
#include <iostream>
#include <unordered_set>
using namespace std;

int main()
```

```
{
    // Initialize three unordered sets
    unordered_set<int>  sample1 = { 10, 20, 30, 40, 50 };
    unordered_set<int>  sample2 = { 10, 30, 50, 40, 20 };
    unordered_set<int>  sample3 = { 10, 20, 30, 50, 60 };

    // Compare sample1 and sample2
    if (sample1 == sample2)
        cout << "sample1 and "  << "sample2 are equal." << endl;
    else
        cout << "sample1 and " << "sample2 are not equal." << endl;

    // Compare sample2 and sample3
    if (sample2 == sample3)
        cout << "sample2 and " << "sample3 are equal." << endl;
    else
        cout << "sample2 and " << "sample3 are not equal." << endl;

    return 0;
}
```

```
Output:
sample1 and sample2 are equal.
sample2 and sample3 are not equal.
```