HASHING SET 1:-

Q) Check if an array is subset of another array .

Logic:- Let A=[3,3,4,4,4,6,7,6] B=[3,3,3,6,7] is B a subset of A ? so the answer is NO because B has total three 3's but A has only two 3's.

So main logic here is to see if frequency of each element in B is lesser or equal to the frequency of same element in A.

Code Approach:-

```
string isSubset(int a1[], int a2[], int n, int m) {
    unordered_map<int,int> m1;
    for(int i=0;i<n;i++){
        m1[a1[i]]++;
    }

     unordered_map<int,int> m2;
    for(int i=0;i<m;i++){
        m2[a2[i]]++;
    }
    string s;
    int flag=1;
    for(int i=0;i<m;i++){
        if(m2[a2[i]] <=m1[a2[i]]){
            flag=1;
            continue;
        }
        else{
            flag=0;
            break;
        }
    }

    if(flag==0) s="No";
    else s="Yes";

    return s;
}
```

Q) Max distance between duplicate elements in an array.

Logic:- At the time of taking input of given array if x is not present in unordered map [condition of not found : um.find(x)==um.end()] then we just insert it's index as um[x]=i .

But if x is already present then,

we take max of (max , (present index of x (i) -first index of x (um[x]))) which is equal to maxDist between two duplicates.

```
// Function to find maximum distance between equal elements
int maxDistance(int arr[], int n)
{
    // Used to store element to first index mapping
    unordered_map<int, int> mp;

    // Traverse elements and find maximum distance between
    // same occurrences with the help of map.
    int max_dist = 0;
    for (int i=0; i<n; i++)
    {
        // If this is first occurrence of element, insert its
        // index in map
        if (mp.find(arr[i]) == mp.end())
            mp[arr[i]] = i;

        // Else update max distance
        else
            max_dist = max(max_dist, i - mp[arr[i]]);
    }

    return max_dist;
}
```

<u>Method 2:-(Not Hashing)</u> We will use two arrays -1) First occ & 2) Last occ

FirstOcc[5] represents the first index position where 5 appeared in given array.

LastOcc[5] represents the last index position where 5 appeared in given array.

When we have firstocc and last occ filled then just do lastocc[i]-firstocc[i] and store the max value of it in each loop .

How to fil firstOcc and lastOcc array?

For firstOcc travel from i=0 to end and whenever you see firstOcc[v[i]]==0 means its unvisited then mark it with its index i.e first position index.

For LastOcc travel from i=n-1 to 0 and whenever you see LastOcc[v[i]]==0 means its unvisited then mark it with its index i.e last position index.

```
class Solution{
    public:
    int firstOcc[1001];
    int lastOcc[1001];
    // your task is to complete this function
    int maxDistance(int arr[], int n)
    {

    for(int i=0;i<1001;i++){
        firstOcc[i]=-1;
        lastOcc[i]=-1;
    }

    for(int i=0;i<n;i++){
        if(firstOcc[arr[i]]==-1){
            firstOcc[arr[i]]=i;

        }
    }

    for(int j=n-1;j>=0;j--){
        if(lastOcc[arr[j]]==-1){
            lastOcc[arr[j]]=j;
        }
    }

    int maxDist=-1;
    for(int k=0;k<1001;k++){
        if(lastOcc[k]!=-1){
           int dist= lastOcc[k]-firstOcc[k];
           if(dist>maxDist){
               maxDist=dist;
           }
        }
    }

    return maxDist;

    }
};
```

Q) Minimum operations to make array equal .

Logic:- The element which will have highest frequency ..the rest of the elements will be converted into that element.

Let you have N elements and highest frequency of an element x is K ..then the rest (N-K) elements will be converted into x.

The min operations(convertions) needed= N- K=size-highest frequency

```
// function for min operation
int minOperation (int arr[], int n)
{
    // Insert all elements in hash.
    unordered_map<int, int> hash;
    for (int i=0; i<n; i++)
        hash[arr[i]]++;

    // find the max frequency
    int max_count = 0;
    for (auto i : hash)
        if (max_count < i.second)
            max_count = i.second;

    // return result
    return (n - max_count);
}
```

Q) Check if an array contains any duplicates within a distance K.

Method1) Here we just have to check whether any duplicate has in between dist <=k …so we will go by cheking nearest two elements…because if nearest elements have not dist <=k then others surely will not have .

Nearest two dupliactes i.e we have to store FirstOcc and SecondOcc then after filling them just check SeconfOcc[i]-firstOcc[i] <=k or not…… if yes then immediately break and make flag=1 else flag will remain 0.

Code Approach:- Make 3 arrays – 1) FirstOcc 2)SceondOcc and 3)Count

FirstOcc[7] = first poition index when 7 appeared in array 1st time

SecondOcc[7] = second poition index when 7 appeared in array 2nd time.

```
int firstOcc[1001];
int secondOcc[1001];

int main(){

int n,k;
cin>>n>>k;
vi v;
f(i,n){
    int x;
    cin>>x;
    v.pb(x);
}
f(i,1001){
    firstOcc[i]=-1;
    secondOcc[i]=-1;

}
f(i,n){
    if(firstOcc[v[i]]==-1){
        firstOcc[v[i]]=i;

    }
    else if(secondOcc[v[i]]==-1){
        secondOcc[v[i]]=i;

    }

}
int flag=0;
f(i,n){
    if(secondOcc[v[i]]!=-1){
        int diff=secondOcc[v[i]]-firstOcc[v[i]];
        if(diff<=k){
            flag=1;
            break;
        }
    }
}

if(flag==0) no();
else yes();

}
```

Q) Sum of all elemets with freq >= the element itself.

Logic:-

Input: arr[] = {2, 1, 1, 2, 1, 6}
Output: 3
The elements in the array are {2, 1, 6}
Where,
2 appear 2 times which is greater than equal to 2 itself.
1 appear 3 times which is greater than 1 itself.
But 6 appears 1 time which is not greater than or equals to 6.
So, sum = 2 + 1 = 3.

At the time of taking input from given array we store the elements in unordered map so we will have freq of each distict element.

After that for(auto it: um) see if it.se means freq is >= element means it.fi if its true then add the element into sum means sum=sum+it.fi

```
int main(){
  int n;
  cin>>n;

  vi v;
  unordered_map<int,int> um;

  f(i,n){
      int x;
      cin>>x;

      um[x]++;
  }

int sum=0;
for(auto it:um){
    if(it.se>=it.fi){
        sum=sum+it.fi;
    }
}

cout<<sum<<endl;


}
```

Q) Find the first non-repeating character in the string .

Logic:- First non-repeating character == First unique character in string.

SO we store the s[i] in unordered_map like um[s[i]]++;

After that loop from i=0 → s.size() if we get um[s[i]]==1 i.e it's the first unique character and immediately store the index and break it.

Later print the index.

```cpp
class Solution {
public:
    int firstUniqChar(string s) {
        unordered_map<char,int> um;

        for(int i=0;i<s.size();i++){
            um[s[i]]++;
        }
int flag=-1;
      for(int i=0;i<s.size();i++){
          if(um[s[i]]==1){
              flag=i;
              break;
          }
      }

      return flag;
    }
};
```

Q) Find Common Charecters in a string .

Example 1:

```
Input: words = ["bella","label","roller"]
Output: ["e","l","l"]
```

Example 2:

```
Input: words = ["cool","lock","cook"]
Output: ["c","o"]
```

Bella = b→1 time , e→1 time , l→2 time , a→1 time

Label= b→1 time e→1 time l→2 time a→1 time

Roller=b→0 time e→1 time l→2 time a→ 0 time r→1time o→1 time

e is common and its present atleast 1 time in all

l is common and its present atleast 2 time in all

So ans=[“e”,”l”,”l”] .

Logic:- First we will store freq of each char of first word in hash1 vector

Hash1(26,0) vector where hash[2] represents how many times b appeared in word 1….hash[i] = freq of ith alphabet in word1

Next we will go through the vector of vector of strings and for each string temporarily store freq of each char in hash2(26,0) vector and compare hash2[i] with hash1[i] and update hash1[i] with min(hash1[i] and hash2[i])

and make hash2[i]=0 for i=0 to 26 also…..because hash2 stores freq of next strings temporarily just compare it with hash1[i] to store the min in hash1[i]

Now we have current minimum of each alphabet in hash1.

In this way we go every string and store their char freq in hash2 and get the min of hash2[i] & hash1[i] in hash1[i] and make hash2[i]=0 for next string.

Now after all these we will have final hash1 ….suppose hash[3]=4 that means we have to push 3==C 4 times like C,C,C,C

```
    vector<string> ans;
    for(int i = 0 ; i < 26 ; i++)

        if(hash1[i] > 0)
        {
            int count = hash1[i];
            while(count--)
            {
                char x = i+ 'a';
                string s ;
                s = x;
                ans.push_back(s);

            }
        }

    return ans;
}
```

```cpp
vector<string> commonChars(vector<string>& arr) {
   vector<int> hash1(26, 0);
    vector<int> hash2(26, 0);

    for(auto ch : arr[0])
    {
        hash1[ch - 'a']++;
    }

    for(int i = 1; i < arr.size() ; i++)
    {
        for(auto ch : arr[i])
        {
            hash2[ch-'a']++;
        }

        for(int i = 0 ; i < 26 ; i++)
        {
            hash1[i] = min(hash1[i], hash2[i]);
            hash2[i] = 0;
        }

    }
```

Q) Count no of pairs in an array which gives target sum k.

Logic:- Suppose we have 1,5,7,-1 and k=6(target sum) …..

Remember um[k] means freqOf(k)

Algorithm is- For i=0 um[k-arr[i]]=um[6-1]=um[5]=1 so ct=ct+1

For i=1 um[6-arr[1]] =um[6-5]=um[1]=1 so ct=ct+1

For i=2 um[6-arr[2]]=um[6-7]=um[1]=1 so ct=ct+1

For i=3 um[6-arr[3]]=um[6- -1]=um[7]=1 so ct=ct+1

So Final ct=4 but here took (a,b) and (b,a) as different pairs so count has been doubled so return ct/2

But one special case:- if arr[i]=k-arr[i] then um[k-arr[i]] consider the element itself too but (arr[i] ,arr[i]) is not a valid pair

So do ct--;

```cpp
class Solution{
public:
    int getPairsCount(int arr[], int n, int k) {
        unordered_map<int,int> um;

        for(int i=0;i<n;i++){
            um[arr[i]]++;
        }

        int ct=0;
        for(int i=0;i<n;i++){
            ct=ct+ um[k-arr[i]];
            if(arr[i]==k-arr[i]){
                ct--;
            }
        }

        return ct/2;
    }
};
```