Q. Print repeating element.

It is opposite of previous question (print distinct element). In this question we have to print those element of array whose are ~~previously~~ already present in array.

ex. -

arr = {2, 5, 2, 7, 5, 2}

o/p - 2, 5, 2

approach 1 - $O(n^2)$ naive approach -

for each element we seach in left of it and if it is present previously we print it.

```
Void pprint repeating(int arr[], int n){
    for(int i=1; i<n; i++){
        bool p=false;
        for(int j=i-1; j>=0; j--){
            if(arr[i]==arr[j]){
                p=true;
                break;
            }
        }
        if(p==true)
            cout<<arr[i]<<" ";
    }
}
```

approach 2 - $O(n)$

approach 2 - O(n) using unordered_set.

check if given element is present in unordered_set, if present previously we print it. and if not present insert it.

```
Void print_repeating(int arr[], int n){
    unordered_set<int> s;
    for(int i=0; i<n; i++){
    auto it = s.find(arr[i]);
      if(it == s.end())
        s.insert(arr[i]);
      else
        cout<<*it<<" ";
    }
}
```

Q. Check pair with given sum.

We have given an array and a sum, and we have to check if there are two element in array whose sum is equal to Sum given.

arr[] = {2, 5, 7, 9, 10}

Sum = 14

o/p - true

approach 1:- O(n²)

for every element search in right side of it and try to find that number.

```
for(int i=0; i<n; i++){
    for(int j=i+1; j<n; j++){
        if(arr[i]==arr[j])
            return true;
    }
}
return false;
```

approach 2:- O(n log n)

We can sort the array or use set.

In both condition our element are in sorted condition.

now, make two variable

i=0 or i=s.begin()

and i=n-1 or i=s.end(); i--;

now, run loop

```
while(i != j){
    if(arr[i] + arr[j] > sum)
        j--;
    else if(arr[i] + arr[j] < sum)
        i++;
    else
        return true;
}
return false;
```

approach 3:- O(n)

we will use unordered_set.

we run a loop for whole array. for every element we check if sum - arr[i] is present in unordered set or not.

if not present insert arr[i] in unordered_set and if present return true.

```
bool pairSum(int arr[], int n, int sum){
    unordered_set<int> s;
    for(int i=0;i<n;i++){
        auto it=s.find(sum-arr[i]);
        if(it==s.end()) //not present.
            s.insert(arr[i]);
        else
            return true;
    }
    return false;
}
```

Q. Intersection of two unsorted array.

We have given two arrays with distinct elements and we have to print intersection of two arrays in order, in which they appear in array one.

Approach 1- $O(n^2)$

We run a loop for first array and inside it we run another loop for second array and check if arr1[i] is present in arr2[] or not if present print it.

```
for(int i=0; i<n1; i++){
    bool flag=false;
    for(int j=0; j<n2; j++){
        if(arr1[i]==arr2[j]){
            flag=true;
            break;
        }
    }
    if(flag==true)
        cout<<arr1[i];
}
```

Approach 2 - O(n)

We will use unordered set to store arr2 so that finding an element becomes O(1).

if arr1[i] present in set, print it and if not leave it.

```
void intersection(int arr1[], int arr2[], int n,
                  int m){
    unordered_set<int> s;
    for(int i=0; i<m; i++)
        s.insert(arr2[i]);
```

```
for(int i=0; i<m; i++)
    s.insert(arr2[i]);

for(int i=0; i<n; i++){
    auto it = s.find(arr1[i]);
    if(it != s.end()){
        cout<<arr1[i]<<" ";
        s.erase(arr1[i]);
    }
}
```

Q. Union of two unsorted array-

We have given two unsorted array having distinct element. We have to find union of both array.

We can print arr1[] and then traverse arr2[] and find element of arr2[] in arr1[] and print those element of arr2[] who are not in arr1[].

approach 1:- $O(n^2)$

We print arr1[] then run loop for arr2[] and inside this loop run another loop for arr1[] and find those elements of arr2[] whose are not in arr1[] and print them.

```
Void union(int arr1[], int arr2[], int n, int m){
    for(int i=0;i<n;i++)
        cout<<arr1[i]<<" ";
    for(int i=0; i<m; i++){ bool flag = false;
        for(int j=0; j<n; j++){
            if(arr2[i] == arr1[j])
                flag = true;
        }
        if(flag == false)
            cout<<arr2[i]<<" ";
    }
}
```

Approach 2- O(n)

we store arr1[] in unordered_set and print it. now search of arr2[] in set, if not present print it.

```
Void union(int arr1[], int arr2[], int n, int m){
    unordered_map<int> m(arr1, arr1+n);
    for(auto x: m)
        cout<<x<<" ";
    for(int i=0; i<m; i++)
        auto it = m.find(arr2[i]);
        if(it == m.end())
            cout<<arr2[i]<<" ";
}
}
```

no of in subarray

Q. Print maximum distinct element in ~~subarray of size k~~ when devided in 'k' equal sizes. (n is multiple of k)

ex. - arr = {2, 1, 1, 3, 2, 1}, n=6

k=2

O/P = 3

Since, k = 2, we can have 3 elements in both subarray we can arrange elements in these subarray in many ways but to get max distinct element in one subarray we will devide it as

{1, 1, 2}, {1, 2, 3}

so, no of distinct element in subarray is 2 in first one and 3 in 2nd one so ans is 3.

- arr[] = {1, 2, 1, 1, 2, 3} n=6

k=3

O/P - 2

here, 3 subarray will be formed with 2 elements each, and since we have 3 distinct elements in array as you can see, in any

arrangement max. no of distinct elemt in any subarray will be 2.

algorithm -

we first calculate no of distinct elemnt in array. We can use unordered_set ($O(n)$) or set ($O(n \log n)$).

then we calculate no of elemnt a subarray can hold by n/k.

now, if $d >= n/k$

It means there are more distinct elements in array then n/k so, we can put them inside one sub-array so, ans will be n/k.

else if $d < n/k$

It means no of distinct elemnt is less then n/k, in this case we put those distinct elemnt in one sub-array and fill rest space with other elements, so ans will 'd'.

code -

```
int distinct_max(int arr[], int n, int k){
    unordered_map<int> m(arr, arr+n);
    int d = m.size();
    if(d >= n/k)
        return n/k;
    else
        return d;
}
```

Q. Union of two sorted arrays and O/P should also be sorted. there can be dublicate element in arrays. T.C - O(n+m)

arr1[] = {2, 2, 3, 4, 5}
arr2[] = {1, 1, 2, 3, 4}

O/P - 1 2 3 4 5

approach -

- make a output vector v.
- make a unordered map S.
- run a loop for arr1[]

Check if element is present in 's' or not.
- if present make arr1[i] = INT_MAX
- else push it to set.

- Run another loop for arr2[]
check if element is present in 's' or not.
- if present make arr2[j] = INT_MAX
- else push it to set.

now, run a while loop (i<n && j<m)
- ignore INT_MAX element and increase i and j accordingly.

```
if(arr1[i] <= arr2[j])
    v.push_back(arr1[i++]);
else
    v.push_back(arr2[j++]);
```

while (i<n)
ignore INT_MAX and push rest of element of arr1[i] into v.

while (j<m)
ignore INT_MAX and push rest of element of arr2[j] in to v.

return v.

Code-

```
vector<int> findUnion(int arr1[], int arr2[], int n, int m)
{   vector<int> v;
    unordered_set<int> s;
    for(int i=0; i<n; i++){
     auto it=s.find(arr1[i]);
      if(it!=s.end())
        arr1[i]= INT_MAX;
      else
          s.insert(arr1[i]);
    }
    for(int i=0; i<m; i++){
     auto it=s.find(arr2[i]);
      if(it!=s.end())
         arr2[i]= INT_MAX;
      else
          s.insert(arr2[i]);
    }
    int i=0, j=0;
    while(i<n && j<m){
      if(arr1[i]==INT_MAX)
           i++;
      else if(arr2[j]==INT_MAX)
           j++;
      else if(arr1[i]<=arr2[j])
           v.push_back(arr1[i++]);
      else
          v.push_back(arr2[j++]);
    }
```

```
while(i<n){
    if(arr1[i]==INT_MAX){
        i++;
    else
        v.push_back(arr1[i++]);
}
while(j<m){
    if(arr2[j]==INT_MAX){
        j++;
    else
        v.push_back(arr2[j++]);
    }
return v;
}
```

Q. Longest Consecutive sequence.

We have given an array of integers, and we have to find length of longest sequence such the elements in sequence are consecutive integers, the consecutive numbers can be in any order.

ex.— arr[] = {2, 6, 1, 9, 4, 5, 3}

o/p = 6 (1, 2, 3, 4, 5 and 6 are in array)

arr[] = {1, 9, 3, 10, 4, 20, 2}

O/P = 4 (as 1, 2, 3 and 4 is longest sequence)

approach 1 - O(n log n);

We first sort the array.
make two variables cur and res;
cur = 1, res = 0;
run a loop from i = 1 to i < n.

```
if(arr[i] == arr[i-1]+1)
    cur++;
else
    res = max(res, cur);
    cur = 1;
```

return res;

Code:-

```
int long_consecutive(int arr[], int n){
    int cur = 1, res = 0;
    sort(arr, arr+n);
    for(int i = 0; i < n; i++){
        if(arr[i] == arr[i-1]+1) cur++;
        else{ res = max(res, cur); cur = 1; }
    }
return res;
}
```

approach 2 - O(n)

We will use unordered_set.
We first make two variables res=0,
cur=1;
and an unordered_set<int> s(arr, arr+n);

run a loop from i=0 to i<n

if check if unordered set contain arr[i]-1 or not

if contain do nothing
if not contain
make cur=1
while(s.find(arr[i]+cur)!=s.end())
cur++;
res=max(res, cur);

return res.

Code-

```
int longConsecutive(int arr[], int n){
    int res=0, cur=1;
    unordered_set<int> s(arr, arr+n);
```

```
for(int i=0; i<n; i++){
    if(s.find(arr[i]-1)==s.end()){
        cur=1;
        while(s.find(arr[i]+cur)!=s.end())
            cur++;
        res=max(res,cur);
    }
}
return res;
}
```

Q. Subarray with zero sum.

We have given given an array of integers and we have to check if there is a subarray (may be of size one or more) whose sum is zero.

ex--

arr[] = {3, 2, 1, -3}

o/p - true (2+1+(-3))

arr[] = {2, 1, 0, 2, 3}

o/p - true ({0})

approach -

We will calculate prefix sum of given array and store that in an unordered set,

but before pushing/inserting prefix sum one by one we check if cur prefix sum is already present in set or not if present means there is a subarray whose sum is 0 and if not there is not.

but there is a corner case, if first subarray whose initial index is '0' can't be in previous case, for this we have to explicitly check if at any point prefix sum is zero.

ex -

arr[] = {2, 3, 1, -4, 2}

Pre. sum = 2 5 6 2, 4

o/p - true

same

arr[] = {3, -3, -1, 4, 5}

Pre. sum = 3, 1, 0, 4, 9

o/p - true (corner condition)

Code-

```
bool is_sum(int arr[], int n){
    unordered_set<int> s;
    int pre_sum = 0;
    for(int i=0; i<n; i++){
        pre_sum += arr[i];
        if(s.find(pre_sum) != s.end())
            return true;
        else if(pre_sum == 0)
            return true;
        s.insert(pre_sum);
    }
    return false;
}
```

Q. Winner of election-

We have given an array of string in which each name represents each vote. we have to find winner. If no of votes for two persons is same then winner will be that person whose name is lexicographically small.

Code-

```
String tiebreak(string name[], int n){
    unordered_map<string, int> m;

    for(int i=0; i<n; i++)
        m[name[i]]++;

    string winner;
    int max_vote=0;

    for(auto x: m){
        if(x.second > max_vote){
            winner=x.first;
            max_vote=x.second;
        }
        else if(x.second == max_vote){
            winner=min(winner, x.first);
        }
    }
    return winner;
}
```