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

int divideIntoTwo(vector<int> v){
    int ans = -1;
    int totalSum = 0;
    for(auto i : v){
        totalSum += i;
    }
    int prefixSum = 0;
    for(int i=0; i<v.size(); i++){
        prefixSum += v[i];
        // if(totalSum-prefixSum = prefixSum)
        if(totalSum == 2*prefixSum){
            ans = i;
            return ans;
        }
    }
    return ans;
}

int LargestSum(vector<int> v){
    int maxi = INT_MIN;
    for(int i=0; i<v.size(); i++){
        int prefixSum = 0;
        for(int j=i; j<v.size(); j++){
            prefixSum += v[j];
            maxi = max(maxi,prefixSum);
        }
    }
    return maxi;
}

int LargestSumUsingKadanes(vector<int> v){
    int maxi = INT_MIN, prefixSum=0;
    for(int i=0; i<v.size(); i++){
        prefixSum += v[i];
        if(prefixSum<0){
            prefixSum = 0;
        }
        maxi = max(maxi,prefixSum);
    }
    return maxi;
}


vector<int> greaterThanThree(vector<int> v){
    unordered_map<int,int> mp;
    vector<int> ans;
    int minLimit = v.size()/3 + 1;
    for(auto ele : v){
        if(mp[ele]==minLimit){
            ans.push_back(ele);
```

```
            }
            mp[ele]++;
        }
        return ans;
    }

    vector<int> greaterThanThree1(vector<int> v){
        int minLimit = v.size()/3 + 1;
        int cnt1 = 0,cnt2 = 0, ele1 = INT_MIN, ele2 = INT_MIN;

        for(int i=0; i<v.size(); i++){
            if(cnt1==0 and ele2!=v[i]){
                ele1 = v[i];
                cnt1++;
            }
            else if(cnt2==0 and ele1!=v[i]){
                ele2 = v[i];
                cnt2++;
            }
            else if(v[i]==ele1){
                cnt1++;
            }
            else if(v[i]==ele2){
                cnt2++;
            }
            else{
                cnt1--;
                cnt2--;
            }
        }

        cnt1 = cnt2 = 0;
        for (auto ele : v) {
            if (ele == ele1) cnt1++;
            else if (ele == ele2) cnt2++;
        }
        vector<int> res;
        if (cnt1 > minLimit) res.push_back(ele1);
        if (cnt2 > minLimit) res.push_back(ele2);
        return res;
    }


    int trapWater(vector<int> h){
        int len = h.size();
        vector<int> leftMax(len,0);
        vector<int> rightMax(len,0);
        for(int i=1; i<len-1; i++){
            leftMax[i] = max(h[i-1],leftMax[i-1]);
        }
        for(int i=len-2; i>=0; i--){
            rightMax[i] = max(h[i+1],rightMax[i+1]);
        }
        int waterCount = 0;
```

```
    for(int i=0; i<len; i++){
        int minHeight = min(leftMax[i],rightMax[i]);
        if(minHeight>h[i]){
            waterCount += minHeight-h[i];
        }
    }
    return waterCount;
}


bool threeSum(vector<int> arr, int x){
    sort(arr.begin(), arr.end());
    for(int i=0; i<arr.size()-2; i++){
        int finalX = x-arr[i];
        int start=i+1, end=arr.size()-1;
        while (start < end) {
            int sum = arr[start] + arr[end];
            if (sum == finalX) {
                return true;
            }
            else if (sum > finalX) {
                end--;
            }
            else {
                start++;
            }
        }
    }
    return false;
}


int main(){
    //divide array in 2 subarray of equal sum
    vector<int> v = {3,4,-2,5,8,20,-10,8};
    int ans = divideIntoTwo(v);
    cout<<"Divide Array into Subarray: "<<ans<<endl;

    //largest sum subarray
    //mtd-1
    vector<int> vec = {3,4,-5,8,-12,7,6,-2};
    int res = LargestSum(vec);
    cout<<"Largest Sum Subarray: "<<res<<endl;

    //mtd-2 - kadane's algo
    vector<int> vec1 = {3,4,9,-5,8,-12,7,6,-2};
    int res1 = LargestSumUsingKadanes(vec1);
    cout<<"Largest Sum Subarray Using Kadane's: "<<res1<<endl;

    vector<int> arr = {2,2,1,2,2,1,2,1,1,1};
    //Element Appearing greater than n/3 times in the given array - Mtd-1
    vector<int> ans1 = greaterThanThree(arr); //Given : There might be multiple element
    for(auto ele : ans1){
        cout<<ele<<" ";
```

```
    }cout<<endl;

    //Element Appearing greater than n/3 times in the given array - Mtd-2
    vector<int> ans2 = greaterThanThree1(arr); //More Opitimized , Given : ans should
always return 2 ele
    for(auto ele : ans2){
        cout<<ele<<" ";
    }cout<<endl;

    //Trapping Rain Water
    vector<int> height = {4,2,0,5,2,6,2,3};
    cout<<trapWater(height)<<endl;

    //threeSum
    vector<int> arr1 = {1,4,45,6,10,8};
    int target = 13;
    cout<<threeSum(arr1,target)<<endl;

    return 0;
}


//Add 2 Number
int main(){
    string str1 = "999583";
    string str2 = "798";

    int carry = 0;
    int first = str1.size()-1, second = str2.size()-1;
    string ans = "";
    while(second>=0){
        int sum = str1[first--]-'0' + str2[second--]-'0' + carry;
        carry = sum/10;
        ans += (sum%10 + '0');
    }

    while(first>=0){
        int sum = str1[first--]-'0' + carry;
        carry = sum/10;
        ans += (sum%10 + '0');
    }
    if(carry){
        ans+="1";
    }
    reverse(ans.begin(),ans.end());
    cout<<ans;
    return 0;
}



//Longest Substring without repeatring char
int main(){
    string str = "abcdecbeadf";
```

```
    int length = INT_MIN, first = 0, second = 0;
    vector<bool> v(256,0); //to handle all ASCII characters
    while(second<str.size()){
        while(v[str[second]]){
            v[str[first++]] = 0;
        }
        v[str[second]] = 1;
        length = max(length,second-first+1);
        second++;
    }
    cout<<"Longest SubString: "<<length<<endl;
    return 0;
}


//Smallest Distinct Window
int main(){
    string str = "aabcbcdbca";
    int first = 0, second = 0, length = INT_MAX, diff = 0;
    vector<int> v(256,0);
    while(first<str.size()){
        if(v[str[first]]==0){
            diff++;
        }
        v[str[first]]++;
        first++;
    }
    first = 0;

    for(int i=0; i<256; i++){
        v[i] = 0;
    }

    while(second<str.size()){
        while(diff and second<str.size()){
            if(v[str[second]]==0){
                diff--;
            }
            v[str[second]]++;
            second++;
        }

        length = min(length,second-first);

        while(diff!=1){
            length = min(length, second-first);
            v[str[first]]--;
            if(v[str[first]]==0) diff++;
            first++;
        }
    }

    cout<<"Length: "<<length<<endl;
    return 0;
```

```
}
```

```cpp
// check whether string is anagram or not
#include<iostream>
#include<vector>
#include<algorithm>
using namespace std;
bool Anagram(string str1, string str2){
    if(str1.length()!=str2.length()){
        return false;
    }
    vector<int> v(26,0);
    for(int i=0; i<str1.length(); i++){
        v[str1[i]]++;
        v[str2[i]]--;
    }
    for(int i=0; i<26; i++){
        if(v[i]!=0){
            return false;
        }
    }
    return true;
}
int main(){
    string str1 = "anagram", str2 = "naagram";
    if(Anagram(str1,str2)){
        cout<<"yes";
        return 0;
    }
    cout<<"no";
    return 0;
}




//longest palindrome you can make out of the given string
int main(){
    string str = "aaababbcc"; //aabcacbaa
    vector<int> v(26,0);
    for(int i=0; i<str.size(); i++){
        v[str[i]-'a']++;
    }
    int longestPlaindromeLength = 0;
    int odd = 0;
    for(int i=0; i<26;i++){
        if(v[i]%2 == 0){ //present even no. of times
            longestPlaindromeLength += v[i];
        }
        else{
            longestPlaindromeLength += v[i]-1;
            odd = 1;
        }
    }
    cout<<"Longest Palindrome Length is: "<<(longestPlaindromeLength+odd);
```

```
    return 0;
}



//return the length of binary substring containing maximum number of 1's you can replace
atmost k 0's
//01001011001 => max length of substring containig 1's is 5
int main(){
    string str = "01001011001";
    int maxLen = 0;
    int zeroCount = 0;
    int k = 2;
    int start = 0, end = 0;
    for (int end = 0; end < str.size(); end++) {
        if (str[end] == '0') {
            zeroCount++;
        }

        while (zeroCount > k) {
            if (str[start] == '0') {
                zeroCount--;
            }
            start++;
        }

        maxLen = max(maxLen, end - start + 1);
        cout<<maxLen<<endl;
    }

    cout << "Maximum length of substring with at most " << k << " zero flips is: " <<
maxLen << endl;
    return 0;
}



//Add Strings num1="2603126" , num2="569" , ans="2603695"
int main(){
    string num1 = "2603126", num2 = "569";
    string ans = "";
    int carry = 0, index1 = num1.size()-1, index2 = num2.size()-1;
    while(index2>=0){
        int num = carry + num2[index2]-'0' + num1[index1]-'0';
        carry = num/10;
        ans += (num%10)+'0';
        index1--;
        index2--;
    }
    while(index1>=0){
        int num = carry + num1[index1]-'0';
        carry = num/10;
        ans += (num%10)+'0';
        index1--;
```

```
        }
        if(carry){
            ans += '1';
        }
        reverse(ans.begin(), ans.end());
        cout<<ans<<endl;
        return 0;
    }
```

```cpp
#include <iostream>
#include<vector>
#include<algorithm>
using namespace std;
//find target in rotated array
int findTarget(int arr[], int target)
{
    int start = 0, end = 8;
    while (end >= start)
    {
        int mid = (start + end) / 2;
        if (arr[mid] == target)
        {
            return mid;
            break;
        }
        // Check if the left part is sorted
        if (arr[mid] >= arr[start])
        {
            if (arr[start] <= target && target < arr[mid])
            {
                end = mid - 1; // Target is in the left part
            }
            else
            {
                start = mid + 1; // Discard the left part
            }
        }
        // Otherwise, the right part must be sorted
        else
        {
            if (arr[mid] < target && target <= arr[end])
            {
                start = mid + 1; // Target is in the right part
            }
            else
            {
                end = mid - 1; // Discard the right part
            }
        }
    }
    return -1;
}
int main()
{
    int arr[] = {7, 8, 9, 1, 2, 3, 4, 5, 6};
    int target = 5;
    int ans = findTarget(arr, target);
    cout << ans;
    return 0;
}
```

```cpp
//Book Allocation
int main(){
    int arr[] = {10,20,30,40};
    // int arr[] = {12,34,67,90};
    int student = 2;
    int size = sizeof(arr)/sizeof(arr[0]);
    int start = 0,end = 0, ans = -1;
    for(int i=0; i<size; i++){
        start = max(start,arr[i]);
        end += arr[i];
    }
    while(start<=end){
        int mid = (start+end)/2;
        int pages = 0, count=1;
        for(int i=0; i<size; i++){
            pages += arr[i];
            //if pages sixe exceed the max no. of pages that start assigning to next
student
            if(pages>mid){
                count++;
                pages = arr[i];
            }
        }
        //if we can assign mid no. of pages to the students , and we want to minimize
the max no. of pages assigned to a student
        //then we can try for a better solution in the left half of the array
        if(count<=student){
            ans = mid;
            end = mid-1;
        }
        //if we cannot assign mid no. of pages to the students, then we have to try for
a better solution in the right half of the array
        //eg if we have to assign max 20 pages to a student and require more than 2
students, then we have to increase the max no. of pages assigned to a student because if
we cannot able to assign max 20 pages than obviously we cannot assign less than 20 pages
also to a student
        else{
            start = mid+1;
        }
    }
    cout<<"ANS: "<<ans;
    return 0;
}



//aggresive cows
bool isPossible(vector<int> &arr, int mid, int cows){
    int count = 1;
    int lastPos = arr[0];
    for(int i=1; i<arr.size(); i++){
        if(arr[i]-lastPos>=mid){
            count++;
```

```
            lastPos = arr[i];
        }
        if(count >= cows){
            return true;
        }
    }
    return false;
}

int main(){
    vector<int> arr = {4,2,1,3,6};
    int cows = 2;
    int size = arr.size();
    int start = 0, end = 0, ans = -1;
    for(int i=0; i<size; i++){
        end = max(end,arr[i]);
    }
    sort(arr.begin(),arr.end());
    while(start<=end){
        int mid = (start+end)/2;
        if(isPossible(arr,mid,cows)){
            ans = mid;
            start = mid+1;
        }
        else{
            end = mid-1;
        }
    }
    cout<<"ANS: "<<ans;
    return 0;
}



//search element in row and column wise sorted matrix
//T.C = O(n+m) and S.C = O(1)
int main(){
    int arr[4][4] = {{10,20,30,40},
                     {15,25,35,45},
                     {27,29,37,48},
                     {32,33,39,50}};
    int target = 29;
    int row = 4;
    int col = 4;
    int i=0,j=col-1;
    while(i<row && j>=0){
        if(arr[i][j] == target){
            cout<<"Element found at: "<<i<<","<<j;
            return 0;
        }
        else if(arr[i][j]>target){
            j--;
        }
        else{
```

```
                i++;
            }
        }
        cout<<"Element not found";
        return 0;
    }
```

```cpp
#include<iostream>
#include<vector>
#include<unordered_map>
#include<algorithm>
using namespace std;

//check whether pair exist in an array whose sum = k
int main(){
    vector<int> v = {1,2,4,6,7,9};
    int k = 8;
    bool present = false;
    unordered_map<int,int> umap;
    for(auto ele : v){
        umap[ele]++;
    }
    for(auto ele:umap){
        int key = ele.first;
        int value = ele.second;
        int pair = k-key;
        if(umap.find(pair)!=umap.end()){
            if(pair==key){
                if(value>1){
                    present = true;
                    break;
                }
            }
            else{
                if(umap.find(pair)!=umap.end()){
                    present = true;
                    break;
                }
            }
        }
    }
    if(present){
        cout<<"yes";
    }
    else{
        cout<<"no";
    }
    return 0;
}



//sort the array ele in pairs of +ve and -ve
// {11,-4,4,-3,-9,4,-4,-2,1,2,-4,2} => {-2,2,-4,4,-4,4}
int main(){
    vector<int> v = {11,-4,4,-3,-9,4,-4,-2,1,2,-4,2};
    unordered_map<int,int> umap;
    vector<int> negativeEle;
    vector<int> ans;
    for(auto ele:v){
```

```
            if(ele>0){
                umap[ele]++;
            }
            else{
                negativeEle.push_back(ele);
            }
        }
        sort(negativeEle.begin(),negativeEle.end());
        for(int i=negativeEle.size()-1; i>=0; i--){
            int absValue = abs(negativeEle[i]);
            if(umap.find(absValue)!=umap.end()){
                if(umap[absValue]>0){
                    umap[absValue]--;
                    ans.push_back(negativeEle[i]);
                    ans.push_back(absValue);
                }
            }
        }
        for(auto ele: ans){
            cout<<ele<<" ";
        }
        return 0;
    }



    //isomorphic string
    bool isOneToOneMapping(string s1, string s2){
        unordered_map<char,char> umap;
        for(int i=0; i<s1.size(); i++){
            if(umap.find(s1[i])!=umap.end()){
                if(umap[s1[i]]!=s2[i]){
                    return false;
                }
            }
            else{
                umap[s1[i]]=s2[i];
            }
        }
        return true;
    }

    bool isIsomorphic(string s1, string s2){
        if(s1.size()!=s2.size()){
            return false;
        }
        bool res1 = isOneToOneMapping(s1,s2);
        bool res2 = isOneToOneMapping(s2,s1);
        return res1 && res2;
    }

    int main(){
        string str1="abcdeag",str2="hijklmn";
        isIsomorphic(str1,str2) ? cout<<"yes" : cout<<"no";
```

```
        return 0;
    }
```

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


//Number of subarray whose sum of elements equals to k
int sumK(vector<int> v, int k){
    unordered_map<int,int> mp;
    mp[0] = 1;
    int prefixSum = 0, total = 0;
    for(auto e : v){
        prefixSum += e;
        int ele = prefixSum - k; //eg arr = [3,5,2] we are at 2nd ele than (3+5)-8(k) = 0 ,0(inserted already) is present {0,1} so we can add 1 to total indicating that we have found a subarray with sum k which is [3,5] in this case
        if(mp.count(ele)){
            total += mp[ele];
            mp[prefixSum]++;
        }
        else{ //always put the element in the map if seen for the first time
            mp[prefixSum] = 1;
        }
    }
    return total;
}


//Subarray sum divisible by k
int sumKDivisible(vector<int> v, int k){
    unordered_map<int,int> mp;
    mp[0] = 1;
    int prefixSum = 0, total = 0, rem = 0;
    for(auto e : v){
        prefixSum += e;
        rem = prefixSum % k;
        if(rem < 0) rem += k; //to handle negative prefix sums
        if(mp.count(rem)){
            total += mp[rem];
            mp[rem]++;
        }
        else{ //always put the element in the map if seen for the first time
            mp[rem] = 1;
        }
    }
    return total;
}


//subarray product less than k
int sumKProductLessThanK(vector<int> v, int k){
    int n = v.size();
    int left = 0, right = 0, product = 1, total = 0;
    while(right < n){
        product *= v[right];
        while(left <= right && product >= k){
```

```
            product /= v[left];
            left++;
        }
        total += (right - left + 1);
        right++;
    }
    return total;
}


//Count of Subarray in which max Element appears at least k times : elelment present at
least k times
int maxEleKTimes(vector<int> v,int k){
    int maxEle = INT_MIN;
    for(auto e:v){
        maxEle = max(maxEle,e);
    }
    int start = 0, end = 0, total = 0 , n = v.size(), count = 0;
    while(end<n){
        if(v[end]==maxEle) count++;
        while(count==k){
            total += (n-end);
            if(v[start]==maxEle) count--;
            start++;
        }
        end++;
    }
    return total;
}


//Max Length of Subarry where particular ele is present atmost k times
int lenAtmostKTimes(vector<int> v, int k){
    int start = 0, end = 0, len = INT_MIN, n = v.size();
    unordered_map<int,int> mp;
    while(end<n){
        mp[v[end]]++;
        while(mp[v[end]] > k){
            mp[v[start]]--;
            start++;
        }
        len = max(len, end - start + 1);
        end++;
    }
    return len;
}


int main(){
    vector<int> v = {3,4,5,3,-6,4,-2,12};
    int target = 8;
    cout<<sumK(v,target);

    vector<int> v2 = {4,5,0,-2,-3,1};
```

```
    int k = 5;
    cout<<sumKDivisible(v2,k)<<endl;

    vector<int> v3 = {1,2,3};
    int k1=0;
    cout<<sumKProductLessThanK(v3,k1)<<endl;

    vector<int> v4 = {1,2,3,2,3,1,2,3,3,2};
    int k2 = 2;
    cout<<maxEleKTimes(v4,k2)<<endl;

    vector<int> v5 = {1,2,3,1,2,3,1,2};
    int k3 = 2;
    cout<<lenAtmostKTimes(v5,k3);
    return 0;
}
```

```cpp
//In priority queue it takes O(logn) time for deleting/inserting elements.
//to make priority queue from an array it takes O(n)-> if buit using step down mtd or O(nlogn)-> if built using stl
#include<bits/stdc++.h>
using namespace std;
class Node{
    public:
    int data;
    Node* next;
    Node(int val){
        data = val;
        next = NULL;
    }
};

void merge(int arr[], int s,int mid, int e){
    int a1 = mid-s+1;
    int b1 = e-mid;
    int a[a1], b[b1];
    for(int i=0; i<a1; i++){
        a[i] = arr[i+s];
    }
    for(int i=0; i<b1; i++){
        b[i] = arr[mid+1+i];
    }
    int i=0, j=0 ,k=s;
    while(i<a1 && j<b1){
        if(a[i]<=b[j]){
            arr[k++] = a[i++];
        }else{
            arr[k++] = b[j++];
        }
    }
    while(i<a1){
        arr[k++] = a[i++];
    }
    while(j<b1){
        arr[k++] = b[j++];
    }
    return;
}

void mergeSort(int arr[], int s, int e){
    if(s>=e){
        return;
    }
    int mid = (s+e)/2;
    mergeSort(arr,s,mid);
    mergeSort(arr,mid+1,e);
    merge(arr,s,mid,e);
}
```

```
Node* mergeLL(Node* head1, Node* head2){
    if(head1==NULL and head2==NULL){
        return NULL;
    }
    Node *dummy = new Node(0);
    Node *tail = dummy;
    while(head1 and head2){
        if(head1->data > head2->data){
            tail->next = head2;
            head2 = head2->next;
        }
        else{
            tail->next = head1;
            head1 = head1->next;
        }
        tail = tail->next;
    }
    if(head1){
        tail->next = head1;
    }
    if(head2){
        tail->next = head2;
    }
    Node* head = dummy->next;
    delete dummy;
    return head;
}


Node* mergeKLL(Node* arr[],int k){
    Node* temp = arr[0];
    for(int i=1; i<k; i++){
        temp = mergeLL(temp,arr[i]);
    }
    return temp;
}


Node* mergeKLL1(Node* arr[], int k){
    priority_queue<int, vector<int> , greater<int>> pq;
    for(int i=0; i<k; i++){
        Node* temp = arr[i];
        while(temp){
            pq.push(temp->data);
            temp = temp->next;
        }
    }
    Node* dummy = new Node(0);
    Node* temp = dummy;
    while(!pq.empty()){
        temp->next = new Node(pq.top());
        pq.pop();
        temp = temp->next;
    }
```

```
    Node* head = dummy->next;
    delete dummy;
    return head;
}

int main(){
    // //find sum of elements between k1^th and k2^th smallest element
    int arr[] = {7,10,4,3,20,15,8,5};
    int k1 = 3, k2 = 6, n = 8;
    priority_queue<int> pq1; //Max-Heap
    priority_queue<int> pq2; //Max-Heap
    for(int i=0; i<k1; i++){
        pq1.push(arr[i]);
    }
    for(int i=0; i<k2; i++){
        pq2.push(arr[i]);
    }

    for(int i=k1; i<n; i++){
        if(pq1.top()>arr[i]){
            pq1.pop();
            pq1.push(arr[i]);
        }
    }

    for(int i=k2; i<n; i++){
        if(pq2.top()>arr[i]){
            pq2.pop();
            pq2.push(arr[i]);
        }
    }
    int sum1 = 0, sum2 = 0;
    while(!pq1.empty()){
        sum1 += pq1.top();
        pq1.pop();
    }
    pq2.pop(); //to remove 1st ele from the second pq
    while(!pq2.empty()){
        sum2 += pq2.top();
        pq2.pop();
    }
    cout<<"Sum is: "<<sum2-sum1<<endl;


    //kth smallest element in a stream
    int arr[] = {1,2,3,4,5,6,7};
    int k = 4 , size = 7;
    vector<int> ans;
    priority_queue<int,vector<int>,greater<int>> pq; //Min-Heap
    for(int i=0; i<k-1; i++){
        ans.push_back(-1);
        pq.push(arr[i]);
    }
    for(int i=k-1; i<size; i++){
```

```
        pq.push(arr[i]);
        ans.push_back(pq.top());
        pq.pop();
    }
    for(auto ele: ans){
        cout<<ele<<" ";
    }


    //merge sort
    int arr[] = {1,5,2,7,100,56,4,7};
    int n = sizeof(arr) / sizeof(arr[0]);
    mergeSort(arr,0,n-1);
    for(int i=0; i<n; i++){
        cout<<arr[i]<<" ";
    }


    //merge 2 sorted linkedlist
    Node* head1 = new Node(1);
    head1->next = new Node(10);
    head1->next->next = new Node(11);
    head1->next->next->next = new Node(15);
    head1->next->next->next->next = new Node(20);

    Node* head2 = new Node(2);
    head2->next = new Node(3);
    head2->next->next = new Node(4);
    head2->next->next->next = new Node(5);
    head2->next->next->next->next = new Node(6);

    Node* temp = mergeLL(head1,head2);
    while(temp){
        cout<<temp->data<<"->";
        temp = temp->next;
    }
    cout<<"NULL"<<endl;


    //merge k sorted linked list
    //Method-1
    Node* ans = mergeKLL(Node* arr[], int size);

    //Method-2 : Using Heap(Priority Queue) O(nklognk)
    Node* ans = mergeKLL1(Node* arr[], int start, int end);

    //Method-3 : using merge Sort
    Node* ans = mergeKLL2(Node* arr[], int size)

    //Method-4 : Using Heap(Priority Queue) O(nklogk)
    Node* ans = mergeKLL3(Node* arr[], int start, int end);

    return 0;
}
```

```cpp
//Largest Rectangle in Histogram : This Problem will cover NGE and NSE(Next Smallest Ele) concept
#include<bits/stdc++.h>
using namespace std;
int main(){
    vector<int> height = {2,1,5,6,2,3};
    // {1,6,4,4,6,6}
    // {-1,-1,1,2,1,4}
    int n = height.size();
    vector<int> right(n,n);
    vector<int> left(n,-1);
    stack<int> st;
    int ans = -1;
    //find NSE in right
    for(int i=0; i<n; i++){
        while(!st.empty() and height[i]<height[st.top()]){
            right[st.top()] = i;
            st.pop();
        }
        st.push(i);
    }

    while(!st.empty()) st.pop();

    //find NSE in left
    for(int i=n-1; i>=0; i--){
        while(!st.empty() and height[i]<height[st.top()]){
            left[st.top()] = i;
            st.pop();
        }
        st.push(i);
    }
    //calculate largest rectangle
    for(int i=0; i<n; i++){
        ans = max(ans,height[i]*(right[i]-left[i]-1));
    }
    cout<<"Ans: "<<ans;
    return 0;
}
```

```cpp
//Evaluation of postfix evaluation
int eval(int v1,int v2, char op){
    if(op == '+'){
        return v1+v2;
    }
    else if(op == '-'){
        return v1-v2;
    }
    else if(op == '*'){
```

```
            return v1*v2;
        }
        else if(op == '^'){
            return v1^v2;
        }
        else{
            return v1/v2;
        }
    }

    int evaluatePostfix(string str){
        stack<int> st;
        for(int i=0; i<str.size(); i++){
            if(isdigit(str[i])){
                st.push(str[i]-'0');
            }
            else{
                int v2 = st.top();
                st.pop();
                int v1 = st.top();
                st.pop();
                st.push(eval(v1,v2,str[i]));
            }
        }
        return st.top();
    }

    int main(){
        string exp = "231*+9-";
        cout<<"Ans: "<<evaluatePostfix(exp);
        return 0;
    }




    //Evaluation of infix exp
    int precedence(char op){
        if(op=='^'){
            return 3;
        }
        if(op=='+' or op=='-'){
            return 1;
        }
        if(op=='*' or op=='/'){
            return 2;
        }
        return -1;
    }

    int evaluateInfix(string str){
        stack<int> digit;
        stack<char> ops;
```

```
        for(int i=0; i<str.size(); i++){
            if(isdigit(str[i])){
                digit.push(str[i]-'0');
            }
            else if(str[i]=='('){
                ops.push(str[i]);
            }
            else if(str[i]==')'){
                while(!ops.empty() and ops.top()!='('){
                    char op = ops.top();
                    ops.pop();

                    int v2 = digit.top();
                    digit.pop();
                    int v1 = digit.top();
                    digit.pop();
                    digit.push(eval(v1,v2,op));
                }
                ops.pop();
            }
            else{
                while(!ops.empty() and precedence(ops.top())>=precedence(str[i])){
                    char op = ops.top();
                    ops.pop();

                    int v2 = digit.top();
                    digit.pop();
                    int v1 = digit.top();
                    digit.pop();
                    digit.push(eval(v1,v2,op));
                }
                ops.push(str[i]);
            }
        }

        while(!ops.empty()){
            char op = ops.top();
            ops.pop();

            int v2 = digit.top();
            digit.pop();
            int v1 = digit.top();
            digit.pop();
            digit.push(eval(v1,v2,op));
        }

        return digit.top();
    }

    int main(){
        string str = "2*(5*(3+6))/5-2";
        cout<<"Ans: "<<evaluateInfix(str);
        return 0;
    }
```

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

//queue using stack
class Queue {
public:
    stack<int> st1; // push stack
    stack<int> st2; // pop stack

    bool empty() {
        return st1.empty() && st2.empty();
    }

    void push(int ele) {
        st1.push(ele);
    }

    int pop() {
        if (empty()) {
            throw runtime_error("Queue Underflow: Cannot pop from empty queue");
        }

        if (st2.empty()) {
            while (!st1.empty()) {
                st2.push(st1.top());
                st1.pop();
            }
        }

        int ele = st2.top();
        st2.pop();
        return ele;
    }

    int peek() {
        if (empty()) {
            throw runtime_error("Queue is empty");
        }

        if (st2.empty()) {
            while (!st1.empty()) {
                st2.push(st1.top());
                st1.pop();
            }
        }

        return st2.top();
    }
};

int main() {
    Queue q;
    q.push(5);
```

```
    q.push(10);
    q.push(20);
    cout << "Front: " << q.peek() << endl;
    cout << "Popped: " << q.pop() << endl;
    cout << "Front after pop: " << q.peek() << endl;
    cout << "Is empty? " << q.empty() << endl;
    q.pop();
    cout << "Is empty? " << q.empty() << endl;

    return 0;
}




// print first negative integer in every window of size k=3
// Method-1 (Less Optimized) - iterating all the elements(positive ele present before
the first -ve ele) for finding first -ve ele
void display(queue<int> q){
    while(!q.empty()){
        if(q.front() < 0){
            cout<<q.front()<<" ";
            return;
        }
        q.pop();
    }
    cout<<0<<" ";
}
int main(){
    vector<int> v = {2,-3,-4,-2,7,8,9,-10};
    queue<int> q;
    int k = 3;
    for(int i=0; i<k-1; i++){
        q.push(v[i]);
    }
    for(int i=k-1; i<v.size(); i++){
        q.push(v[i]);
        display(q);
        q.pop();
    };
    return 0;
}


//Method-2 (Optimized) - Only Storing -ve ele's , eliminating iterations
void display(queue<int> q, vector<int> v){
    if(q.empty()){
        cout<<0<<" ";
        return;
    }
    cout<<v[q.front()]<<" ";
}

int main(){
```

```
    vector<int> v = {2,-3,-4,-2,7,8,9,-10};
    queue<int> q;
    int k = 3;
    for(int i=0; i<k-1; i++){
        if(v[i]<0){
            q.push(i);
        }
    }
    for(int i=k-1; i<v.size(); i++){
        if(v[i]<0){
            q.push(i);
        }
        while(!q.empty() and q.front() <= i-k){
            q.pop();
        }
        display(q,v);
    }
    return 0;
}
```

```
//dequeue using array
#include <iostream>
using namespace std;

class Dequeue {
    int start, end, size, count;
    int *arr;

public:
    Dequeue(int n) {
        size = n;
        start = 0;
        end = n - 1;
        count = 0;
        arr = new int[n];
    }

    bool isFull() {
        return count == size;
    }

    bool isEmpty() {
        return count == 0;
    }

    void push_start(int x) {
        if (isFull()) {
            cout << "Dequeue Overflow\n";
            return;
        }
        start = (start - 1 + size) % size;
```

```
        arr[start] = x;
        count++;
    }

    void push_back(int x) {
        if (isFull()) {
            cout << "Dequeue Overflow\n";
            return;
        }
        end = (end + 1) % size;
        arr[end] = x;
        count++;
    }

    int front() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return -1;
        }
        return arr[start];
    }

    int back() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return -1;
        }
        return arr[end];
    }

    void pop_start() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return;
        }
        start = (start + 1) % size;
        count--;
    }

    void pop_back() {
        if (isEmpty()) {
            cout << "No Ele\n";
            return;
        }
        end = (end - 1 + size) % size;
        count--;
    }
};

int main() {
    Dequeue d(5);
    cout << d.front() << endl;      // No Ele
    d.push_back(3);
    d.push_start(2);
```

```
    cout << d.front() << endl;      // 2
    cout << d.back() << endl;       // 3
    d.pop_start();                  // remove 2
    cout << d.front() << endl;      // 3
    d.push_start(5);
    d.push_start(6);
    d.push_back(7);                 // Should fill it
    d.push_back(8);
    d.push_back(9);                 //should thrown an overflow error

    return 0;
}
```

```cpp
// Stream First Non-repeating
// Input: s = "aabc"
// Output: "a#bb"
class Solution {
  public:
    string FirstNonRepeating(string &s) {
        // Code here
        unordered_map<char,int> mp;
        queue<char> q;
        string ans = "";
        for(char e : s){
            q.push(e);
            mp[e]++;

            while(!q.empty() and mp[q.front()]>1){
                q.pop();
            }
            if(q.empty()){
                ans+="#";
            }
            else{
                ans+=q.front();
            }
        }
        return ans;
    }
};


// Sliding Window Maximum
// Input: nums = [1,3,-1,-3,5,3,6,7], k = 3
// Output: [3,3,5,5,6,7]
class Solution {
public:
    int display(queue<int> q){
        int ele = INT_MIN;
        while(!q.empty()){
            ele = max(ele,q.front());
            q.pop();
        }
        return ele;
    }
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        vector<int> ans;
        queue<int> q;
        for(int i=0; i<k-1; i++){
            q.push(nums[i]);
        }
        for(int i=k-1; i<nums.size(); i++){
            q.push(nums[i]);
            ans.push_back(display(q));
            q.pop();
        }
```

```
        return ans;
    }
};



// Optimized Approach
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        vector<int> ans;
        deque<int> q;
        for(int i=0; i<k-1; i++){
            if(q.empty()) q.push_back(i);
            else{
                while(!q.empty() and nums[i]>nums[q.back()]){
                    q.pop_back();
                }
                q.push_back(i);
            }
        }

        for(int i=k-1; i<nums.size(); i++){
            //enter the current element in the queue
            while(!q.empty() and nums[i]>nums[q.back()]){
                q.pop_back();
            }
            q.push_back(i);
            //check if first element in queue is valid or not
            if(q.front()<=i-k){
                q.pop_front();
            }
            ans.push_back(nums[q.front()]);
        }
        return ans;
    }
};



// Minimum Number of K Consecutive Bit Flips
// Input: nums = [0,1,0], k = 1
// Output: 2
class Solution {
public:
    int minKBitFlips(vector<int>& nums, int k) {
        int flipCount = 0;
        int n= nums.size();
        for(int i=0; i<n; i++){
            if(nums[i]==0){
                if(i+k-1>=n){
                    return -1;
                }
                flipCount++;
```

```
                for(int j=i; j<=i+k-1; j++){
                    nums[j] = !nums[j];
                }
            }
        }
        return flipCount;
    }
};
```

```cpp
// Longest Consecutive Sequence
// Input: nums = [100,4,200,1,3,2]
// Output: 4

class Solution
{
public:
    int longestConsecutive(vector<int> &nums)
    {
        if (nums.size() == 0)
            return 0;

        // mtd-1
        sort(nums.begin(), nums.end());
        int count = 0, longest = 1, lastSmaller = INT_MIN;
        for (int i = 0; i < nums.size(); i++)
        {
            if (nums[i] - 1 == lastSmaller)
            {
                count++;
                lastSmaller = nums[i];
            }
            else if (nums[i] != lastSmaller)
            {
                count = 1;
                lastSmaller = nums[i];
            }
            longest = max(count, longest);
        }
        return longest;

        // mtd-2
        int longest = 1, count = 0;
        unordered_set<int> st;
        for (auto e : nums)
        {
            st.insert(e);
        }
        for (auto e : st)
        {
            if (st.find(e - 1) == st.end())
            {
                // the current element is the starting ele of sequence
                count = 1;
                while (st.find(e + 1) != st.end())
                {
                    count++;
                    e = e + 1;
                }
            }
            longest = max(longest, count);
        }
        return longest;
```

```
        }
    };

    // Longest Substring Without Repeating Characters
    class Solution
    {
    public:
        int lengthOfLongestSubstring(string s)
        {
            if (s.size() == 0)
                return 0;
            int first = 0, second = 0, len = INT_MIN;
            vector<bool> allAsciiChar(256, 0);
            while (second < s.size())
            {
                while (allAsciiChar[s[second]])
                {
                    allAsciiChar[s[first++]] = 0;
                }
                len = max(len, second - first + 1);
                allAsciiChar[s[second]] = 1;
                second++;
            }
            return len;
        }
    };

    // Count Inversions in an array
    class Solution
    {
    public:
        // Function to count inversions in the array.
        int merge(vector<int> &arr, int left, int mid, int right)
        {
            int leftA = mid - left + 1;
            int rightA = right - mid;
            int count = 0;
            vector<int> arrLeft(leftA), arrRight(rightA);
            for (int i = 0; i < leftA; i++)
            {
                arrLeft[i] = arr[left + i];
            }
            for (int i = 0; i < rightA; i++)
            {
                arrRight[i] = arr[mid + 1 + i];
            }

            int i = 0, j = 0, k = left;
            while (i < leftA && j < rightA)
            {
                if (arrLeft[i] <= arrRight[j])
                {
                    arr[k++] = arrLeft[i++];
                }
```

```
            else
            {
                count += leftA - i;
                arr[k++] = arrRight[j++];
            }
        }
        while (i < leftA)
        {
            arr[k++] = arrLeft[i++];
        }
        while (j < rightA)
        {
            arr[k++] = arrRight[j++];
        }
        return count;
    }

    int mergeSort(vector<int> &arr, int left, int right)
    {
        int cnt = 0;
        if (left >= right)
            return 0;
        int mid = left + (right - left) / 2;
        cnt += mergeSort(arr, left, mid);
        cnt += mergeSort(arr, mid + 1, right);
        cnt += merge(arr, left, mid, right);
        return cnt;
    }
    int inversionCount(vector<int> &arr)
    {
        // Your Code Here
        return mergeSort(arr, 0, arr.size() - 1);
    }
};

// longest prefix suffix
// abababab => from starting "abab",  from ending "abab" therfore abab is the longest
prefix which is also a suffix
class Solution
{
public:
    string longestPrefix(string s)
    {
        int n = s.size();
        vector<int> lps(n, 0);
        int prefix = 0, suffix = 1;
        while (suffix < n)
        {
            if (s[prefix] == s[suffix])
            {
                lps[suffix] = prefix + 1;
                prefix++, suffix++;
            }
            else
```

```
            {
                if (prefix == 0)
                {
                    lps[suffix] = 0;
                    suffix++;
                }
                else
                {
                    prefix = lps[prefix - 1];
                }
            }
        }
        return s.substr(0, lps[n - 1]);
    }
};

// Input: intervals = [[1,3],[2,6],[8,10],[15,18]]
// Output: [[1,6],[8,10],[15,18]]
// Explanation: Since intervals [1,3] and [2,6] overlap, merge them into [1,6].
class Solution
{
public:
    vector<vector<int>> merge(vector<vector<int>> &intervals)
    {
        int n = intervals.size();
        vector<vector<int>> ans;
        sort(intervals.begin(), intervals.end());
        for (int i = 0; i < n; i++)
        {
            if (ans.empty() || ans.back()[1] < intervals[i][0])
            {
                ans.push_back(intervals[i]);
            }
            else
            {
                ans.back()[1] = max(ans.back()[1], intervals[i][1]);
            }
        }
        return ans;
    }
};

// Input: s = "aacecaaaa"
// Output: 2
// Explanation: Add 2 a's at front of above string to make it palindrome : "aaaacecaaaa"
class Solution
{
public:
    int lps(string str)
    {
        int n = str.size();
        vector<int> s(n, 0);
        int prefix = 0, suffix = 1;
        while (suffix < n)
```

```
        {
            if (str[prefix] == str[suffix])
            {
                s[suffix] = prefix + 1;
                prefix++, suffix++;
            }
            else
            {
                if (prefix == 0)
                {
                    s[suffix] = 0;
                    suffix++;
                }
                else
                {
                    prefix = s[prefix - 1];
                }
            }
        }
        return s[n - 1];
    }
    int minChar(string &s)
    {
        // Write your code here
        string temp = s;
        reverse(s.begin(), s.end());
        string final = temp + "$" + s;
        return s.size() - lps(final);
    }
};
```