Sliding Window Maximum:

```
class Solution {
  public:
    vector<int> max_of_subarrays(vector<int> a, int n, int k) {
       vector<int> ans;
       deque<int> deq;
       for(int i = 0; i < n;++i){
           while(deq.size()>0 and a[i]>=a[deq.back()])
                deq.pop_back();
           if(deq.size()>0 and deq.front() == i-k)
                deq.pop_front();
           deq.push_back(i);
           if(i>=k-1)
                ans.push_back(a[deq.front()]);

       }
       return ans;
    }
};
```

Next Greater Element:

```
class Solution
{
    public:
    //Function to find the next greater element for each element of the array.

    vector<long long> nextLargerElement(vector<long long> a, int n){
        stack<long long> stk;
        vector<long long> ans(n);
        for(int i = 0; i < n; i++){
            while(stk.size()>0 and a[i]>a[stk.top()]){
                ans[stk.top()] = i;
                stk.pop();
            }
            stk.push(i);
        }
        for(int i = 0; i < n; i++){
            if(ans[i] == 0)
                ans[i] = -1;
            else
                ans[i] = a[ans[i]];
        }
        return ans;

    }
};
```

Find Peak Element in an Array:

```
class Solution
{
    public:
    int peakElement(int arr[], int n)
    {
       int left = 0, right = n-1;
       while(left<right){
           int mid = (left+right)/2;
           if(arr[mid]<arr[mid+1])
               left = mid+1;
            else
                right = mid;
       }
       return right;
    }
};
```

Minimum Number of Platforms to schedule all trains: (Arrival Priority):

```
class Solution{
    public:
    //Function to find the minimum number of platforms required at the
    //railway station such that no train waits.
    int findPlatform(int arr[], int dep[], int n)
    {

      vector<pair<int,int> > events;
      for(int i = 0; i < n; ++i){
          events.push_back({arr[i],0});
          events.push_back({dep[i],1});
      }
      sort(events.begin(),events.end());

      int makeplatform = 0;
      int ans = 0;
      for(auto &x: events){
          if(x.second == 0){
              makeplatform++;
          }
          else makeplatform--;
          ans = max(ans,makeplatform);
      }
      return ans;
    }
};
```

Minimum Number of Platforms to schedule all trains: (Deperture Priority):

```
class Solution{
    public:
    //Function to find the minimum number of platforms required at the
    //railway station such that no train waits.
    int findPlatform(int arr[], int dep[], int n)
    {

      vector<pair<int,int> > events;
      for(int i = 0; i < n; ++i){
          events.push_back({arr[i],1});
          events.push_back({dep[i],0});
      }
      sort(events.begin(),events.end());

      int freeplat = 0;
      int ans = 0;
      for(auto &x: events){
          if(x.second == 0){
              freeplat++;
          }
          else{
            if(freeplat == 0)
                  ans++;
            else
                  freeplat--;
          }
      }
      return ans;
    }
};
```

Minimum Cost to Connect all ropes:

```
class Solution
{
    public:
    //Function to return the minimum cost of connecting the ropes.
    long long minCost(long long arr[], long long n) {
        priority_queue<long long> pq;
        long long ans = 0;
        for(int i = 0; i < n; ++i){
            pq.push(-arr[i]);
        }
        for(int i = 0; i < n-1; ++i){
            long long a = -pq.top();
            pq.pop();
            long long b = -pq.top();
            pq.pop();
            ans+= a+b;
            pq.push(-(a+b));
        }
        return ans;
    }
};
```

Largest rectangular area in a histogram:

```
class Solution
{
    public:
    //Function to find largest rectangular area possible in a given histogram.
    long long getMaxArea(long long a[], int n)
    {
        vector<long long> left(n,-1),right(n,n);
        stack<long long> stk1,stk2;
        for(int i = 0; i < n; ++i){
            while(stk1.size()>0 and a[i] < a[stk1.top()]){
                right[stk1.top()] = i;
                stk1.pop();
            }
            stk1.push(i);
        }
        for(int i = n-1; i>=0; i--){
            while(stk2.size()>0 and a[i] < a[stk2.top()]){
                left[stk2.top()] = i;
                stk2.pop();
            }
            stk2.push(i);
        }
        long long ans = 0;
        for(int i = 0; i < n; ++i){
            ans = max(ans,a[i]*(right[i]-left[i]-1));
        }
        return ans;
    }
};
```

Longest Unique SubString:

```
int longestSubstrDistinctChars (string s)
{
    int n = s.size();
    unordered_map<int,int> mp;
    int len = 0;
    int start = 0;
    for(int i = 0; i < n; ++i){
        if(mp.find(s[i]) != mp.end()){
            start = max(start,mp[s[i]]+1);
        }
        mp[s[i]] = i;
        len = max(len,i-start+1);
    }
    return len;
}
```

Longest Consequtive Band:

```
class Solution{
  public:

    //Function to return length of longest subsequence of consecutive integers.
    int findLongestConseqSubseq(int arr[], int n)
    {
        unordered_set<int> uset;
        for(int i = 0; i < n; ++i){
            uset.insert(arr[i]);
        }
        int maxCnt = 0;
        for(auto &element: uset){
            int prev = element-1;
            if(uset.find(prev) == uset.end()){
                int next = element+1;
                int currCnt = 1;
                while(uset.find(next)!=uset.end()){
                    next++;
                    currCnt++;
                }
                maxCnt = max(currCnt,maxCnt);
            }
        }
        return maxCnt;
    }
};
```

Marge Overlapping Intervals:

```
class Solution {
public:
    vector<vector<int>> overlappedInterval(vector<vector<int>>& iv) {
         int n = iv.size();
         sort(iv.begin(),iv.end());
         stack<vector<int> > stk;
         stk.push(iv[0]);
         for(int i = 1; i < n; ++i){
             auto top = stk.top();
             if(top[1] < iv[i][0]){
                 stk.push(iv[i]);
             }
             else if(top[1] < iv[i][1]){
                 top[1] = iv[i][1];
                 stk.pop();
                 stk.push(top);
             }
         }
         vector<vector<int> > ans;
         while(stk.size()>0){
             auto top = stk.top();
             ans.push_back(top);
             stk.pop();
         }
         sort(ans.begin(),ans.end());
         return ans;
    }
};
```

Zero Sum Subarray:

```
class Solution{
    public:
    //Function to count subarrays with sum equal to 0.
    ll findSubarray(vector<ll> arr, int n ) {
        ll sum=0;
        unordered_map<ll,ll>mp;
        ll cnt=0;
        for(ll i=0;i<n;i++)
        {
            sum+=arr[i];
            if(sum==0)
             cnt++;
             if(mp.find(sum)!=mp.end())
             cnt+=mp[sum];
            mp[sum]++;
        }

        return cnt ;
    }
};
```

Longest SubArray With 0 sum:

```
class Solution{
    public:
    int maxLen(vector<int>&a, int n)
    {
        map<int,int> mp;
        int sum = 0;
        int ans = 0;
        for(int i = 0; i < n; ++i){
            sum+=a[i];
            if(sum == 0)
                ans = i+1;
            else if(mp.find(sum)!=mp.end()){
                ans = max(ans,i-mp[sum]);
            }
            else
            mp[sum] = i;
        }
        return ans;
    }
};
```

Number of Subarray having sum == k:

```
class Solution{
    public:
    int findSubArraySum(int a[], int n, int k)
    {
        map<int,int> mp;
        int sum = 0;
        int cnt = 0;
        for(int i = 0; i < n; ++i){
            sum+=a[i];
            if(sum == k)
                cnt++;
            if(mp.find(sum-k)!=mp.end())
                cnt+=mp[sum-k];
            mp[sum]++;
        }
        return cnt;
    }
};
```

Equilibrium Point of an array:

```
class Solution{
    public:

    int equilibriumPoint(long long v[], int n) {
        if(n == 1)
            return 1;

        int pre[n],suff[n];
        pre[0] = v[0];
        suff[n-1] = v[n-1];
        for(int i = 1; i < n; i++){
            pre[i] = v[i]+pre[i-1];
            suff[n-i-1] = suff[n-i]+v[n-i-1];
        }
        int idx = -1;
        for(int i = 1; i < n-1;i++){
            if(pre[i-1] == suff[i+1]){
                idx = i+1;
                break;
            }
        }
        return idx;
    }

};
```

Optimized:

```
class Solution{
    public:
    // Function to find equilibrium point in the array.
    // a: input array
    // n: size of array
    int equilibriumPoint(long long v[], int n) {
        if(n == 1)
            return 1;

        int left = 0, right = 0;
        for(int i = 1; i < n; ++i){
            right+=v[i];
        }
        int idx = -1;
        for(int i = 1; i < n-1; ++i){
            if(left == right)
                idx = i;
            left+=v[i-1];
            right -=v[i];

        }
        return idx;
    }

};
```

Leaders in an Array:

```
class Solution{
    //Function to find the leaders in the array.
    public:
    vector<int> leaders(int a[], int n){
        vector<int> lead;
        lead.push_back(a[n-1]);
        int max = a[n-1];
        for(int i = n-2; i>= 0; i--){
            if(a[i]>=max){
                max = a[i];
                lead.push_back(max);
            }
        }
        reverse(lead.begin(),lead.end());
        return lead;


    }
};
```

SubArray With Given Sum:

```
class Solution
{
    public:
    //Function to find a continuous sub-array which adds up to a given number.
    vector<int> subarraySum(int a[], int n, long long s)
    {
        long long l =0, r = 0, sum = 0;
        while(l<n){
            if(sum<s){
                sum+=a[r++];
            }
            if(sum>s){
                sum -= a[l++];
            }
            if(sum == s)
                return {l+1,r};
        }
        return {-1};
    }
};
```

Check Two Arrays are equal or not:

```
class Solution{
    public:

    //Function to check if two arrays are equal or not.
    bool check(vector<ll> A, vector<ll> B, int n) {
        unordered_map<int,int> umap;
        for(int i = 0; i < n; ++i){
            umap[A[i]]++;
        }
        for(int i = 0; i < n; ++i){
            if(umap.find(B[i]) == umap.end())
                return false;
            if(umap[B[i]] == 0)
                return false;
            umap[B[i]]--;
        }
        return true;
    }
};
```

Swapping two Elements to make their sum equal:

```
class Solution{

      public:
      int findSwapValues(int a[], int n, int b[], int m)
      {
       long long s1 = accumulate(a,a+n,0);
       long long s2 = accumulate(b,b+m,0);
       int target;

       if(abs(s1-s2)%2)
            return -1;
        else
            target = (s1-s2)/2;

        sort(a,a+n);
        sort(b,b+m);

        int i = 0, j = 0;

        while(i<n and j < m){
            if(a[i]-b[j] == target)
                return 1;
            if(a[i]-b[j]<target)
                i++;
            else
                j++;
        }
        return -1;

      }


};
```

Kadane Algo:

```
class Solution{
    public:
    // arr: input array
    // n: size of array
    //Function to find the sum of contiguous subarray with maximum sum.
    long long maxSubarraySum(int a[], int n){

        long long maxsum = INT_MIN;
        long long curr = 0;
        for(int i = 0; i < n; i++){
            curr+=a[i];
            maxsum = max(maxsum,curr);
            if(curr<0)
                curr = 0;

        }
        return maxsum;

    }
};
```

Activity Selection Problem:

```
class Solution
{
    public:
    //Function to find the maximum number of activities that can
    //be performed by a single person.
    static bool compare(pair<int,int> &a, pair<int,int> &b){
        return a.second<b.second;
    }
    int activitySelection(vector<int> start, vector<int> end, int n)
    {
        vector<pair<int,int> > vp(n);

        for(int i = 0; i < n; ++i){
            vp[i] = {start[i],end[i]};
        }

        sort(vp.begin(),vp.end(),compare);
        int cnt = 1;
        int i = 0;
        for(int j = 1; j < n; ++j){
            if(vp[i].second < vp[j].first){
                cnt++;
                i = j;
            }
        }
        return cnt;
    }
};
```

Count Inversion in an array:

```
class Solution{
  public:
    // arr[]: Input Array
    // N : Size of the Array arr[]
    // Function to count inversions in the array.
    long long marge(long long a[], long long b[],int min, int mid, int max){

        int i = min;
        int j = mid;
        int k = min;
        long long inv = 0;
        while(i<= mid-1 and j <= max){
            if(a[i]<=a[j]){
                b[k++] = a[i++];
            }
            else{
                b[k++] = a[j++];
                inv+=(mid-i);
            }
        }
        while(i<=mid-1){
            b[k++] = a[i++];
        }
        while(j<=max){
            b[k++] = a[j++];
        }
        for(int i = min; i <=max;++i)
            a[i] = b[i];
        return inv;
    }
    long long margesort(long long a[],long long b[],int min, int max){
        long long inv = 0;
        if(max>min){
            int mid = (min+max)/2;
            inv+=margesort(a,b,min,mid);
            inv+=margesort(a,b,mid+1,max);
            inv+=marge(a,b,min,mid+1,max);
        }
        return inv;
    }
    long long int inversionCount(long long a[], long long n)
    {
        long long b[n];
        return margesort(a,b,0,n-1);
    }

};
```

Minimum Height of Tower:

```
class Solution {
  public:
    int getMinDiff(int a[], int n, int k) {
        if(n==0)
            return -1;
        sort(a,a+n);
        int minh = 0,maxh = 0, res = a[n-1] - a[0];

        for(int i = 1; i < n; ++i){
            if(a[i]>=k){
                minh = min(a[0]+k,a[i]-k);
                maxh = max(a[n-1]-k,a[i-1]+k);
                res = min(res,maxh-minh);
            }
        }
        return res;
    }
};
```

Minimize the sum of product:

```
class Solution{
    public:
    long long int minValue(int a[], int b[], int n)
    {
        sort(a,a+n);
        sort(b,b+n,greater<int>());
        long long int sumpro = 0;
        for(int i = 0; i < n; ++i){
            sumpro+=1LL*a[i]*b[i];
        }
        return sumpro;
    }
};
```

The Celebrity Problem:

```
class Solution
{
    public:
    //Function to find if there is a celebrity in the party or not.
    int celebrity(vector<vector<int> >& M, int n)
    {
        stack<int> stk;
        for(int i = 0; i < n; i++)
            stk.push(i);
        while(stk.size()>1){
            int a = stk.top();
            stk.pop();
            int b = stk.top();
            stk.pop();
            if(M[a][b])
                stk.push(b);
            else
                stk.push(a);
        }

        int celeb = stk.top();
        stk.pop();
        for(int i = 0; i < n; ++i){
            if(i!=celeb){
                if(!M[i][celeb] or M[celeb][i])
                    return -1;
            }
        }
        return celeb;

    }
};
```

Longest Consequitive one:

```
class Solution
{
    public:
    int maxConsecutiveOnes(int n)
    {
        int cnt = 0;
        while(n){
            cnt++;
            n = (n & (n<<1));
        }
        return cnt;
    }
};
```

Product of all elements in array except self:

```
class Solution{
  public:
    // nums: given vector
    // return the Product vector P that hold product except self at each index
    vector<long long int> productExceptSelf(vector<long long int>& a, int n) {

       vector<long long int> pro(n);

       long long int temp = 1;
       for(int i = 0; i < n; ++i){
           pro[i] = temp;
           temp*=a[i];
       }
       temp = 1;

       for(int i = n-1; i >= 0; --i){
           pro[i]*=temp;
           temp*=a[i];
       }
       return pro;

    }
};
```

K-th Permutation:

```
class Solution {
public:
    string getPermutation(int n, int k) {
        int fact = 1;
        vector<int> nums;
        for(int i =1; i < n; ++i){
            fact*=i;
            nums.push_back(i);
        }
        nums.push_back(n);
        string ans = "";
        k--;
        while(true){
            ans+=to_string(nums[k/fact]);
            nums.erase(nums.begin()+k/fact);
            if(nums.size() == 0)
                break;
            k = k%fact;
            fact/=nums.size();
        }
        return ans;


    }
};
```

Find All Pairs in an two arrays with Given sum:

```
class Solution{
    public:
    vector<pair<int,int>> allPairs(int a[], int b[], int n, int m, int x)
    {
        sort(a,a+n);
        sort(b,b+m);

        int i = 0, j = m-1;
        vector<pair<int,int>> ans;
        while(i<n and j>=0){
            if(a[i]+b[j] == x){
                ans.push_back({a[i],b[j]});
                i++;
                j--;
            }
            else if(a[i]+b[j]>x){
                j--;
            }
            else{
                i++;
            }
        }
        return ans;

    }
};
```

Find All Four Sum numbers:

```
class Solution{
    public:
    // arr[] : int input array of integers
    // k : the quadruple sum required
    vector<vector<int> > fourSum(vector<int> &a, int k) {

        int n = a.size();
        sort(a.begin(),a.end());
        set<vector<int>> sett;
        vector<vector<int>> ans;

        for(int i = 0; i < n-2; ++i){
            for(int j = i+1; j < n-1; ++j){
                int sum = a[i]+a[j];
                int l = j+1, r = n-1;
                while(l<r){
                    if(sum+a[l]+a[r] == k){
                        sett.insert({a[i],a[j],a[l],a[r]});
                        l++;
                        r--;
                    }
                    else if(sum+a[l]+a[r]>k){
                        r--;
                    }
                    else
                        l++;
                }
            }
        }

        for(auto &x: sett){
            ans.push_back(x);
        }return ans;}};
```

Valid Parenthesis Check:

```
class Solution
{
    public:
    //Function to check if brackets are balanced or not.
    bool opening(char x){
        return x == '(' or x == '{' or x == '[';
    }
    int typeof(char x){
        if(x == '(' or x == ')')
            return 1;
        if(x == '{' or x == '}')
            return 2;
        else
            return 3;
    }
    bool ispar(string s)
    {
        stack<char> stk;

        for(auto &x: s){
            if(opening(x)){
                stk.push(x);
            }
            else{
                if(!opening(x) and stk.size() == 0)
                    return 0;
                if(stk.size()>0 and typeof(stk.top()) == typeof(x))
                    stk.pop();
                else
                    return 0;
            }
        }

        return stk.empty();
    }

};
```

Stock Span Problem:

```
class Solution
{
    public:
    //Function to calculate the span of stockâ€™s price for all n days.
    vector <int> calculateSpan(int price[], int n)
    {
        stack<int> stk;
        vector<int> span(n);
        stk.push(0);
        span[0] = 1;
        for(int i = 1; i < n; ++i){
            while(stk.size()>0 and price[i]>=price[stk.top()])
                stk.pop();
            if(stk.size()>0)
                span[i] = i-stk.top();
            else
                span[i] = i+1;
            stk.push(i);

        }
        return span;
    }};
```

Running Median:

```
priority_queue<int> maxpq;
priority_queue<int, vector<int>, greater<int> > minpq;

void addEle(int x){

      if(maxpq.empty() or maxpq.top()>x)
            maxpq.push(x);
      else
            minpq.push(x);

      if(maxpq.size()>minpq.size()+1){
            minpq.push(maxpq.top());
            maxpq.pop();
      }
      else if(minpq.size()>maxpq.size()+1){
            maxpq.push(minpq.top());
            minpq.pop();
      }
}

double findMedian(){
      double median;
      if(maxpq.size() == minpq.size()){
            median = (maxpq.top()+minpq.top())/(2.0);
      }
      else if(maxpq.size()>minpq.size()){
            median = maxpq.top();
      }
      else{
            median = minpq.top();
      }
      return median;
}
```