EASY :

1. Union and intersection of 2 array

```
class Solution{
    public:
       int doUnion(int a[], int n, int b[], int m)  {
        unordered_set<int>st;
        for(int i=0;i<n;i++)
        st.insert(a[i]);
        for(int i=0;i<m;i++)
        st.insert(b[i]);
        return st.size();
    }
};
```

2. Largest Sum contiguous Subarray (Kadane’s Algorithm)

```
class Solution{
    public:
    int maxSubarraySum(int arr[], int n)
{
        int sum=0;
        int sum1=INT_MIN;
        for(int i=0;i<n;i++)
        {
          sum=sum+arr[i];
          if(sum<arr[i])
          sum=arr[i];
          if(sum1<sum)
         sum1=sum;
       }
    return sum1;
    }
};
```

3. Best time to buy and sell stock

```
class Solution {
public:
    int maxProfit(vector<int>& prices)
    {
        int minnn=INT_MAX;
        int diff=INT_MIN;
    for(int i=0;i<prices.size();i++)
    {
        if(minnn>prices[i])
        minnn=prices[i];
        if(prices[i]-minnn>diff)
            diff=prices[i]-minnn;
    }
```

```
        return diff;
    }
};
```

MEDIUM :

1. Sort array 0,1 and 2.

```
class Solution {
public:
    void sortColors(vector<int>& nums)
    {
        int z=0;
        int o=0;
        int t=0;
        vector<int>ans;
        for(int i=0;i<nums.size();i++)
        {
            if(nums[i]==0)
            z++;
            else if(nums[i]==1)
            o++;
            else
            t++;
            nums[i]=0;
        }
        int i=0;
        while(z--)
        {
           nums[i]=0;
            i++;
        }
        while(o--)
        {
           nums[i]=1;
            i++;
        }
        while(t--)
        {
            nums[i]=2;
            i++;
        }
     }
};
```

2. Maximize difference between heights

```
class Solution {
  public:
    int getMinDiff(int arr[], int n, int k) {
       sort(arr, arr+n);
```

```
        int ans=arr[n-1]-arr[0];
        int minn=arr[0]+k;
        int maxx=arr[n-1]-k;
        int miin;
        int maax;
        for(int i=0; i<n-1; i++)
        {
            miin=min(minn,arr[i+1]-k);
            maax=max(maxx,arr[i]+k);
            if(miin<0)
            continue;
            else
            ans=min(ans,maax-miin);
        }
        return ans;

    }
};
```

3. Minimum jump need to reach to end.

```
class Solution{
  public:
    int minJumps(int arr[], int n){
        int sum=arr[0];
        int next=arr[0];
        int c=1;
        if (arr[0]==0)
        return -1;
        for (int i=1;i<n;i++)
       {
         if (i>=n-1)
         return c;
         next=max(next,i+arr[i]);
         sum--;
         if(sum==0)
        {
            c++;
            if(next<=i)
            return -1;
            sum=next-i;
        }
        }
     }
};
```

4. Find duplicate in Array

```
class Solution {
public:
    int findDuplicate(vector<int>& nums)
    {
        int slow =nums[0];
        int fast =nums[0];

        do{
            slow = nums[slow];
            fast = nums[nums[fast]];
        }while(fast!=slow);


             fast = nums[0];
        while(slow!=fast)
        {
            slow = nums[slow];
            fast = nums[fast];
        }
        return slow;
    }
};
```

5. Merge intervals

```
class Solution {
public:
    vector<vector<int>> merge(vector<vector<int>>& intervals)
    {
        int n=intervals.size();
        if(n<=1)
        return intervals;
        sort(intervals.begin(),intervals.end());

        vector<int>ans1=intervals[0];
        vector<vector<int>>ans;

        for(auto it:intervals)
        {

            if(ans1[1]>=it[0])
            ans1[1]=max(ans1[1],it[1]);
            else
            {
                ans.push_back(ans1);
                ans1=it;
            }
        }

        ans.push_back(ans1);
        return ans;
    }
};
```

6. Next permutation

```
class Solution {
public:
    void nextPermutation(vector<int>& nums)
    {
        int i=0;
        if(nums.size()<=1)
        return ;
        for(i = nums.size() - 1 ; i > 0; i--)
        {
            if(nums[i]>nums[i - 1])
            break;

        }
        if(i == 0)
        reverse(nums.begin() , nums.end());
        else
        {
            int x = nums[i - 1];
            int s=i;
            for(int j = i + 1; j < nums.size(); j++)
            {
                if(nums[j] > x && nums[j] <= nums[s])
                s = j;
            }
            swap(nums[i - 1] , nums[s]);
            cout<<x<<" "<<nums[s]<<"\n";

            sort(nums.begin() + i, nums.end());
        }
    }
};
```

7. Count inversion

```
int merge(int *Arr, int start, int mid, int end)
{
         int temp[end-start+1];
         int i = start;
    int j = mid;
    int k = 0;
    int ans=0;
   while(i <= mid-1 && j <= end)
    {
         if(Arr[i] <= Arr[j])
```

```
        {
                    temp[k] = Arr[i];
                    k++;
                    i++;
         }
         else
        {
                              temp[k] = Arr[j];
                              k++;
                             j++;
                            ans=ans+(mid-i);
         }
     }
    while(i <= mid-1)
    {
                    temp[k] = Arr[i];
                    k++;
                     i++;
         }
    while(j <= end)
    {
                    temp[k] = Arr[j];
                    k++;
                    j++;
         }
    for (i = start; i <= end; i++)
    Arr[i] = temp[i-start];
    return ans;
         }
int  mergeSort(int *Arr, int start, int end)
{
     int mid;
    int ans=0;
         if(start < end)
    {
                    int mid = (start + end) / 2;
                    ans+=mergeSort(Arr,start, mid);
                    ans+=mergeSort(Arr,mid+1, end);
                    ans+=merge(Arr, start, mid+1, end);
         }
    return ans;
}
```