Find the maximum and minimum element in an array

```
  pair<long long, long long> getMinMax(long long a[], int n) {
    long long min=INT_MAX;
    long long max= INT_MIN;
    for(int i=0;i<n;i++){
        if(a[i]>max)
        max=a[i];

        if(a[i]<min)
        min=a[i];
    }
    return{min, max};
}
```

Sort an array of 0s, 1s and 2s

```
void sort012(int a[], int n){
 int low = 0, mid = 0, high = nums.size()-1;
        while(mid <= high){
            if(nums[mid] == 0){
                swap(nums[low], nums[mid]);
                low++;
                mid++;
            }
            else if(nums[mid] == 1){
                mid++;
            }
            else{
                swap(nums[mid], nums[high]);
                high--;
            }
}
```

Cyclically rotate an array by one

```
 void rotate(int arr[], int n)
{ int j=arr[n-1];
    for(int i=n-1; i>0;i--){
        arr[i]=arr[i-1];
    }
   arr[0]=j;
```

```
}
```

Kth smallest element

```
int kthSmallest(int arr[], int l, int r, int k) {
    priority_queue<int> q;
    for(int i=0;i<=r;++i){
        q.push(arr[i]);
        }
        int n=r-k+2;
    while(--n){
        q.pop();
    }
    return q.top();
}
```

Write a program to reverse an array or string

```
string reverseWord(string str)
  {
   int i=0,j= str.length()-1;
   while(i<j){
       swap(str[i++], str[j--]);
   }
   return str;
  }
```

Move all negative elements to end

```
for(int i = 0; i < n; i++) {
        if(arr[i] > 0) {
        new_arr[p] = arr[i];
        p++;
    }
}
```

```
    for(int i = 0 ; i < n ; i++){
        if(arr[i] < 0) {
        new_arr[p] = arr[i];
        p++;
     }
  }
  for(int i = 0; i < n ; i++) {
        arr[i] = new_arr[i];
  }
```

Union of two arrays

```
  int doUnion(int a[], int n, int b[], int m)  {
    map<int,bool>rep;
    int k=n+m;
    int un[k];
    int j=0;
    for(int i=0;i<n;i++){
        int key=a[i];
        if(rep[key]==false){
            rep[key]=true;
            un[j]=a[i];
            j++;
        }
    }
    for(int i=0;i<m;i++){
        int key=b[i];
        if(rep[key]==false){
            rep[key]=true;
            un[j]=b[i];
            j++;
        }
    }
    return j;
  }
```

Kadane's Algorithm

```
long long maxSubarraySum(int arr[], int n){
      long long sum=0;
      long long maxi=arr[0];
```

```
        for(int i=0;i<n;i++){
            sum+=arr[i];
            maxi=max(sum,maxi);
            if(sum<0)
            sum=0;
        }
        return maxi;
    }
```

Find the Duplicate Number

```
  int findDuplicate(vector<int>& nums) {
      map<int, bool>rep;
      for(int i=0; i<nums.size();i++){
          int key=nums[i];
          if(rep[key]==true)
          return key;
          rep[key]=true;
      }
      return 0;
  }
```

m-2

```
  int findDuplicate(vector<int>& nums) {
      int low = 1, high = nums.size() - 1, cnt;

      while(low <=  high)
      {
          int mid = low + (high - low) / 2;
          cnt = 0;
          for(int n : nums)
          {
              if(n <= mid)
                  ++cnt;
          }
          if(cnt <= mid)
              low = mid + 1;
          else
              high = mid - 1;

      }
```

```
    return low;
  }
```

Minimum number of jumps

```
  int jump(vector<int>& nums) {
    int j=0,x=0,count=0,n=nums.size();
    if(n<=1){
      return 0;
    }
    while(j<nums.size()){
      if(nums[j]+j>=n-1)
      return count+1;
      int maxi=0,ind;
      for(int i=nums[j]+j;i>j;i--){
        if(maxi<nums[i]+i){
          ind=i;
          maxi=nums[i]+i;
        }
      }
      j=ind;
      count++;
    }
    return count;
  }
};
```

Common elements

```
vector <int> commonElements (int A[], int B[], int C[], int n1, int n2, int n3)
    { int i = 0,j = 0,k = 0;
    set<int>s;
    vector<int>ans;
    while (i < n1 && j < n2 && k < n3) {
      if (A[i] == B[j] && B[j] == C[k]) {
        s.insert(A[i]);
        i++; j++; k++;
      }
      else if (A[i] < B[j])
        i++;
      else if (B[j] < C[k])
```

```
            j++;
        else
            k++;
    }
    for(auto i : s)
    ans.push_back(i);

    return ans;
    }
```

Longest consecutive subsequence

```
int findLongestConseqSubseq(int arr[], int N){
      int maxi= 0,k=1;
      if(N == 0)
          return 0;
      std::sort(arr, arr + N);
      for(int i=1;i < N;i++){
          if(arr[i] != arr[i-1]){
          if(arr[i] == arr[i-1]+1)
          k++;
          else{
              maxi=max(maxi,k);
              k=1;
          }
      }
      }
      return max(maxi,k);
   }
```