1. A highly efficient sorting algorithm based on partitioning of array of data into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than specified value on which the partition is made and another array holds values greater than specified value. This algorithm is quite efficient for large sized data sets as its average and worst case complexity are of O(nlogn).
   Inputs:
   First input the 'n' i.e. size of array, followed by elements in the array.
   Output:
   Displays sorted array.

   Sample Input 1:
   5 10 100 200 1000 890

   Sample Output 1:
   10 100 200 890 1000

   Testcase:
   Input:

```
5
1
3
4
5
2
```

   Output:

```
1
2
3
4
5
```

```
#include<iostream>
using namespace std;
int parts(int *a,int start,int end)
{
   //complete the function
}
void sorting(int *a,int start, int end)
{
   if(start<end)
   {
      int partition_index=parts(a,start,end);
      sorting(a,start,partition_index-1);
      sorting(a,partition_index+1,end);
   }
```

```
}
int main()
{
   int n;
   cin>>n;
   int a[n];
   for(int i=0;i<n;i++)
      cin>>a[i];
   sorting(a,0,n-1);
   for(int i=0;i<n;i++)
      cout<<a[i]<<"\n";}
```

2. Perform a simple sorting algorithm i.e. Insertion sort that builds the final sorted array (or list) one item at a time. It iterates and consumes one input element of each repetition. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain. 'x' is the an unsorted number in the rightmost cell, write a code to insert 'x' into the array so that it remains sorted.
   Print the array every time a value is shifted in the array until the array is fully sorted.
   Input Format
   There will be two lines of input:
   • *Size* - the size of the array
   • *Arr* - the array containing "*Size*-1" sorted integers and 1 unsorted integer 'x' in the rightmost cell
   Output Format
   On each line, output the entire array every time an item is shifted in it.
   Sample Input
   5
   2 4 6 8 3
   Sample Output
   2 4 6 8 8
   2 4 6 6 8
   2 4 4 6 8
   2 3 4 6 8

```
#include <iostream>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <assert.h>
using namespace std;
void insertionSort(int ar_size, int *  ar) {
```

```
//Write your code here.}

int main(void) {
   int _ar_size;
   cin>>_ar_size;
   int _ar[_ar_size], _ar_i;
   for(_ar_i = 0; _ar_i < _ar_size; _ar_i++) {
      cin>>_ar[_ar_i];
   }

   insertionSort(_ar_size, _ar);
   return 0;
}
```

3. You are given two sorted arrays A and B of size M and N respectively. Write code that can these two arrays to form another sorted array C. The constraint here is that this procedure is to be completed by using minimum no. of comparisons 'X' between elements of A and B.

   Sample Input: First line inputs **M**, no. of elements in array **A** Second line inputs **N**, no. of elements in array **B** Third line inputs **M** elements in **A** Fourth line inputs **N** elements in **B**

   Sample Output: First line shows the contents of array **C** after merging **A** and **B** Second line shows the number of comparisons **X**, between elements of **A** and **B.**

   Input:

```
4
6
1 5 9 13
2 4 6 8 10 12
```

   Output:

```
1 2 4 5 6 8 9 10 12 13
9
```

```
#include<iostream>
using namespace std;

int merge(int *A, int M, int *B, int N, int *C){
//Enter your code here
}
int main(){
 int i,M,N;
```

```
    cin >> M >> N;
    int A[M],B[N],C[M+N];
    int X;
    for(i=0;i<=M-1;i++)
       cin >> A[i];
    for(i=0;i<=N-1;i++)
       cin >> B[i];

    X = merge(A,M,B,N,C);

    for(i=0;i<=M+N-1;i++)
       cout << C[i] << " ";
    cout << endl << X;

    return 0;
}
```

4. Insertion sort has better efficiency than bubble sort and selection sort algorithms. It is an online sorting method that can be used to efficiently insert an item into a sorted list so that even after insertion, the updated list remains sorted.
   The objective of this problem is to find out if you can follow the correct ordering algorithm of insertion sort.

   The input consists of two lines. First line corresponds to size of array Second line consists of size-1 sorted elements and 1 unsorted element at the rightmost position.

   Sample Output:

   On each line, print the entire array every time a value is shifted in it.

   2 is removed from the end of the array. In the 1st line 6 > 2, so 6 is shifted one position to the right. In the 2nd line 5 > 2, so 5 is shifted one position to the right. In the 3rd line 3 > 2, so 3 is shifted one position to the right. In the 4th line 1 < 2, so 2 is placed at position 2.

```
Input:
5
2 3 4 5 1
```

   Output:

```
2 3 4 5 5
2 3 4 4 5
2 3 3 4 5
2 2 3 4 5
1 2 3 4 5
```

```
#include<iostream>
using namespace std;
```

```
void insertion_sort(int n, int *B)
//Enter your code here

}

int main(void) {
   int size;
   cin >> size;
   int A[size], i;
   for(i = 0; i < size; i++)
      cin >> A[i];

   insertion_sort(size, A);
   return 0;
}
```

5. Selection sort is a simple sorting algorithm. This sorting algorithm is an in-place comparison-based algorithm in which the list is divided into two parts, the sorted part at the left end and the unsorted part at the right end. Initially, the sorted part is empty and the unsorted part is the entire list.
   The smallest element is selected from the unsorted array and swapped with the leftmost element, and that element becomes a part of the sorted array. This process continues moving unsorted array boundary by one element to the right.
   Following example explains the above steps:
   Linked List = 64 25 12 22 11
   Itr 1: Find the minimum element in Linked List and place it at beginning
   11 25 12 22 64

   Itr 2: Find the minimum element from second position in Linked List and place it at beginning of Linked List selected from 2nd position
   11 **12** 25 22 64

   Itr 3: Find the minimum element from 3rd position in Linked List and place it at beginning of Linked List selected from 3rd position
   11 12 **22** 25 64

   Itr 4: Find the minimum element from 4th position in Linked List and place it at beginning of Linked List selected from 4th position
   11 12 22 **25** 64

   Sample Input:
   64 25 12 22 11 -999

   Sample Output:
   64, 25, 12, 22, 11
   11, 64, 25, 22, 12
   11, 12, 64, 25, 22

11, 12, 22, 64, 25
11, 12, 22, 25, 64
11, 12, 22, 25, 64

Sample Input:

It consist of the number of values in the nodes followed by -999 which represents termination of linked list

Sample Output:

```
64, 25, 12, 22, 11    //linked list formed
11, 64, 25, 22, 12    //ITR1
11, 12, 64, 25, 22    //ITR2
11, 12, 22, 64, 25    //ITR3
11, 12, 22, 25, 64    //ITR4
11, 12, 22, 25, 64    //Final Sorted List
```

Input:

```
89 23 76 11 95 25 8 16 -999
```

Output:

```
89, 23, 76, 11, 95, 25, 8, 16
8, 89, 76, 23, 95, 25, 11, 16
8, 11, 89, 76, 95, 25, 23, 16
8, 11, 16, 89, 95, 76, 25, 23
8, 11, 16, 23, 95, 89, 76, 25
8, 11, 16, 23, 25, 95, 89, 76
8, 11, 16, 23, 25, 76, 95, 89
8, 11, 16, 23, 25, 76, 89, 95
8, 11, 16, 23, 25, 76, 89, 95
```

```
#include <iostream>
#include <string>

using namespace std;

struct node
{
    int data;
    struct node *next;
};

class linkedList
{
    node *head;
    public:
     void create();
     void display();
     void selectionSort();
};

void linkedList :: create()
```

```
{

//write your code here

}

void linkedList :: display()
{
  node *start;
  start=head;
  while(start!=NULL){
     cout<<start->data;
     if(start->next!=NULL)
         cout<<", ";
     start=start->next;
  }
}
void linkedList :: selectionSort()
{


// write your code here

}

int main()
{
    linkedList list;
  list.create();
  list.display();
  cout<<endl;
  list.selectionSort();
    return 0;
}
```

6. A class Student having private data members and member functions as:

   char name[100]
   int marks
   Student() Default constructor which will initialize marks with 0.
   void getdata(void) function which will accept name,marks of Student from keyboard.
   void display(void) function which will display name,marks of the student.
   friend void SortRankWise(Student[],int) which will sort Student record in descending order according to marks.
   Complete functions getdata(),display(),SortRankWise(Student[],int) .

   Sample Input
   3
   sathal 45
   kevin 66
   tanya 88

   Sample Output

tanya 88
kevin 66
sathal 45

In Sample Input first line contains total number of students i.e n.
Next lines contain name and marks of n students.
In Sample output record of n students is displayed after marks wise sorting in descending order.

Input2:

```
kap 98
iii 90
def 67
abc 45
```

Output2:

```
4
abc 45
def 67
iii 90
kap 98
```

```
#include <iostream>
#include <iomanip>
#include<cstring>

using namespace std;
class Student
{
char name[100];
int marks;
public:
Student();
void getdata(void);
void display(void);
friend void sortRankWise( Student S[],int);
};

//write your code here

int  main()
{
int n;
cin>>n;
Student s[n];
for(int i=0;i<n;i++)
{
s[i].getdata();
}
sortRankWise(s,n);
for(int i=0;i<n;i++)
{
s[i].display();
}
return 0;}
```

7. Merge Sort is a Divide and Conquer algorithm. It divides input array in two halves, calls itself for the two halves and then merges the two sorted halves. **The merge() function** is used for merging two halves.

- user will enter the list.
- a sorted list will be display.

Sample Input:
2, 3,6,11,33,1,33,88,22

Sample Output:
[1, 2, 3, 6, 11, 22, 33, 33, 88]

Input2:
2, 3, 1

Output2:
```
[1, 2, 3]
```

8. John has data set of numbers which he want to sort in complexity of O(log n) as the numbers are large and random in nature he chooses quick sort, But in parallel he wants to check the output after each iteration. In the code provided, *quicksort* class has been created with no_of_elements and elements[] array. Functions *display()* used to show the contents of array, *getarray()* to get input into the array has already been created. You are required to write the code for the functions void *QuickSort::sortit()* and void *QuickSort::partition()*

****Constraint: The first element (lower bound) of the array is to be considered as the Pivot element for first call.****

****Sample Input 1****

5 //No. of elements in the array

14 86 45 38 24 //Elements of the array

Sample Output 1 // Displays elements of array after each iteration

14 86 45 38 24
14 86 45 38 24
14 24 45 38 86
14 24 45 38 86
14 24 38 45 86

Sample Input 2 7 // No. of elements in the array 32 45 98 76 54 11 60 // Elements of the array.

Sample Output 2 // Displays elements of array after each iteration
32 45 98 76 54 11 60
11 32 98 76 54 45 60
11 32 60 76 54 45 98
11 32 54 45 60 76 98
11 32 45 54 60 76 98
Input3:
15
70 80 90 10 11 22 76 88 64 33 23 12 89 99 1

Output3:

```
70 80 90 10 11 22 76 88 64 33 23 12 89 99 1
64 1 12 10 11 22 23 33 70 88 76 90 89 99 80
33 1 12 10 11 22 23 64 70 88 76 90 89 99 80
23 1 12 10 11 22 33 64 70 88 76 90 89 99 80
22 1 12 10 11 23 33 64 70 88 76 90 89 99 80
11 1 12 10 22 23 33 64 70 88 76 90 89 99 80
10 1 11 12 22 23 33 64 70 88 76 90 89 99 80
1 10 11 12 22 23 33 64 70 88 76 90 89 99 80
1 10 11 12 22 23 33 64 70 80 76 88 89 99 90
1 10 11 12 22 23 33 64 70 76 80 88 89 99 90
1 10 11 12 22 23 33 64 70 76 80 88 89 99 90
1 10 11 12 22 23 33 64 70 76 80 88 89 90 99
```

```
#include<iostream>
using namespace std;
class QuickSort{
   public:
      int no_of_elements;
      int elements[100];
   public:
      void getarray();
      void sortit(int [], int, int);
      void partition(int [],int ,int,int&);
      void display();
};

void QuickSort::getarray(){

   cin>>no_of_elements;

   for(int i=0;i<no_of_elements;i++){
      cin>>elements[i];
            }
         }

void QuickSort::sortit(int x[], int lb, int ub){
//write your code here
}
```

```
void QuickSort::partition(int x[],int lb,int ub,int &pj){
//write your code here
}

void QuickSort::display(){
   for(int i = 0 ; i < no_of_elements; i++){
      cout<<elements[i]<<" ";
   }
   cout<<endl;
}

int main(){
   QuickSort QS;
   QS.getarray();
   QS.sortit(QS.elements,0,QS.no_of_elements-1);
   QS.display();
   return 0;
}
```

9. Certain Marks are to be sorted in ascending order in such a way that the complexity of the sorting algorithm remains O(nlogn). We choose merge sort to obtain the desired result. The code needs to print the result of each of the pass.

   Sample Input 1

   8 // no. of elements passed to the array

   12 5 61 60 50 1 70 81 //Elements of the array

   Sample Output after each pass

   5 12

   60 61

   5 12 60 61

   1 50

   70 81

   1 50 70 81

   1 5 12 50 60 61 70 81

   1 5 12 50 60 61 70 81

   Sample Input2:

   8

   12 5 61 60 6 7 2 186

   Sample Output2:

   5 12

   60 61

   5 12 60 61

   6 7

   2 186

   2 6 7 186

   2 5 6 7 12 60 61 186

   2 5 6 7 12 60 61 186

```
#include<iostream>
using namespace std;
void mergesort(int *,int,int);
void merge(int *,int,int,int);
int a[20],i,n,b[20];

int main()
{

cin >> n;

for(i=0;i<n;i++)
cin >> a[i];
mergesort(a,0,n-1);

for(i=0;i<n;i++)
cout << a[i] << " ";
return 0;
}

void mergesort(int a[],int i,int j)
{
//Write your code here
}
```

10. You have been given the task to sort the contents of a list using insertion sort. you have been provided the list and pass number. Your task is to display the list for the nth pass.

    Sample Input
    4 8 3 77 100 177
    3
    Sample Output
    [3, 4, 8, 77, 100, 177]

    Explanation:
    4 8 3 77 100 177 is the list accepted
    3 is the pass number for which the list has to be displayed

    for ex
    [4, 8, 3, 77, 100, 177] pass1
    [3, 4, 8, 77, 100, 177] pass2
    [3, 4, 8, 77, 100, 177] pass3
    [3, 4, 8, 77, 100, 177] pass4
    [3, 4, 8, 77, 100, 177] pass5

    Input:
    10 20 30 1 3 5 7
    5

    Output:
    [1, 3, 5, 10, 20, 30, 7]