1. Bubble Sort

```
# include <iostream>
#include <conio.h>
using namespace std;

void bubbleSort(int arr[], int n)
{
   int i, j, temp;
   for(i = 0; i < n; i++)
   {
      for(j = 0; j < n-i-1; j++)
      {
         if( arr[j] > arr[j+1])
         {
            // swap the elements
            temp = arr[j];
            arr[j] = arr[j+1];
            arr[j+1] = temp;
         }
      }
   }

   // print the sorted array
   cout<<"Sorted Array: ";

   for(i = 0; i < n; i++)
   {
      cout<<arr[i]<<" ";
   }
}

int main()
{
   int arr[100], i, n, step, temp;
   // ask user for number of elements to be sorted
  cout<<"Enter the number of elements to be sorted: ";
  cin>>n;
   // input elements if the array
   for(i = 0; i < n; i++)
   {
      cout<<"Enter element no. "<< i+1<<": ";
```

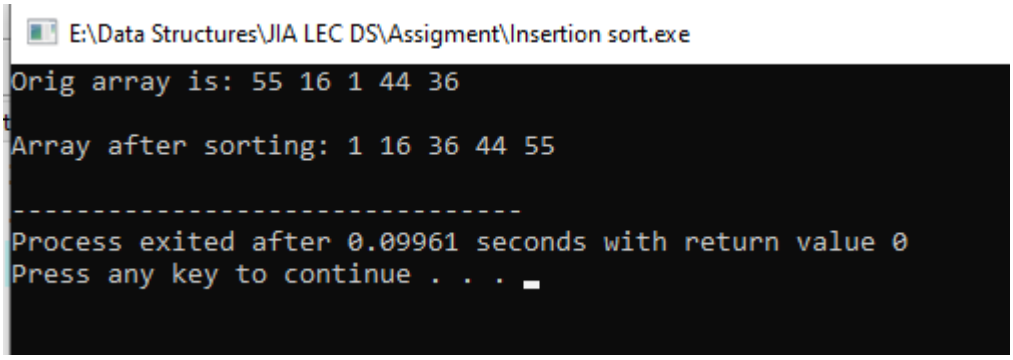
```
    cin>>arr[i];
  }
  // call the function bubbleSort
  bubbleSort(arr, n);

  return 0;
}
```

```
E:\Data Structures\JIA LEC DS\Assigment\Bubble Sort.exe
Enter the number of elements to be sorted: 4
Enter element no. 1: 6
Enter element no. 2: 45
Enter element no. 3: 9
Enter element no. 4: 33
Sorted Array: 6 9 33 45
--------------------------------
Process exited after 17.01 seconds with return value 0
Press any key to continue . . .
```

2. Insertion Sort

```
// C++ program for insertion sort
# include <iostream>
#include <conio.h>
using namespace std;

/* Function to sort an array using insertion sort*/
void insertionSort(int arr[], int n)
{
        int i, key, j;
        for (i = 1; i < n; i++)
        {
                key = arr[i];
                j = i - 1;

                /* Move elements of arr[0..i-1], that are
                greater than key, to one position ahead
                of their current position */
                while (j >= 0 && arr[j] > key)
                {
                        arr[j + 1] = arr[j];
                        j = j - 1;
                }
                arr[j + 1] = key;
        }
```

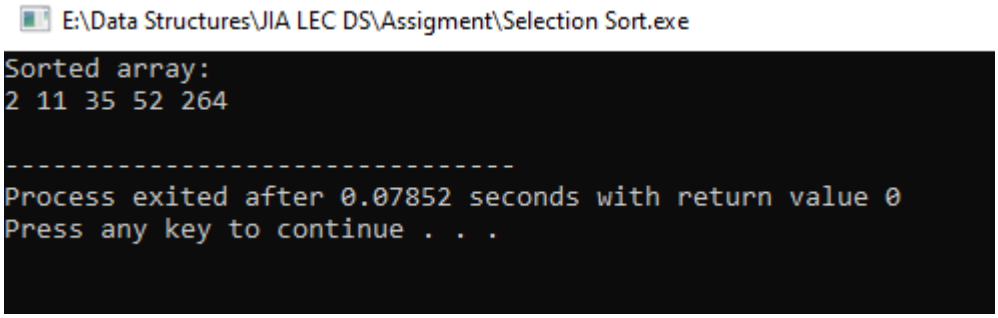
```
}

// A utility function to print an array of size n
void printArray(int arr[], int n)
{
	int i;
	for (i = 0; i < n; i++)
		cout << arr[i] << " ";
	cout << endl;
}

/* Driver code */
int main()
{
	int arr[5] = { 55, 16, 1, 44, 36 };
	int n = 5;

	insertionSort(arr, n);
	printArray(arr, n);

	return 0;
}
```



3. Selection Sort

```
// C++ program for implementation of selection sort
# include <iostream>
#include <conio.h>
using namespace std;

void swap(int *xp, int *yp)
{
	int temp = *xp;
	*xp = *yp;
```

```
	*yp = temp;
}

void selectionSort(int arr[], int n)
{
	int i, j, min_idx;

	// One by one move boundary of unsorted subarray
	for (i = 0; i < n-1; i++)
	{
		// Find the minimum element in unsorted array
		min_idx = i;
		for (j = i+1; j < n; j++)
		if (arr[j] < arr[min_idx])
			min_idx = j;

		// Swap the found minimum element with the first element
		swap(&arr[min_idx], &arr[i]);
	}
}

/* Function to print an array */
void printArray(int arr[], int size)
{
	int i;
	for (i=0; i < size; i++)
		cout << arr[i] << " ";
	cout << endl;
}

// Driver program to test above functions
int main()
{
	int arr[] = {264, 35, 52, 2, 11};
	int n = 5;
	selectionSort(arr, n);
	cout << "Sorted array: \n";
	printArray(arr, n);
	return 0;
}
```



4. Merge Sort

```
// C++ program for Merge sort
/*
   a[] is the array, p is starting index, that is 0,
   and r is the last index of array.
*/
#include <iostream>
using namespace std;

// lets take a[5] as the array to be sorted.
// function to merge the subarrays
void merge(int a[], int p, int q, int r)
{
   int b[5];   //same size of a[]
   int i, j, k;
   k = 0;
   i = p;
   j = q + 1;
   while(i <= q && j <= r)
   {
      if(a[i] < a[j])
      {
         b[k++] = a[i++];    // same as b[k]=a[i]; k++; i++;
      }
      else
      {
         b[k++] = a[j++];
      }
   }

   while(i <= q)
   {
      b[k++] = a[i++];
   }
```

```
   while(j <= r)
   {
      b[k++] = a[j++];
   }

   for(i=r; i >= p; i--)
   {
      a[i] = b[--k];  // copying back the sorted list to a[]
   }
}

// merge sort function
void mergeSort(int a[], int p, int r)
{
   int q;
   if(p < r)
   {
      q = (p + r) / 2;
      mergeSort(a, p, q);
      mergeSort(a, q+1, r);
      merge(a, p, q, r);
   }
}

// function to print the array
void printArray(int a[], int size)
{
   int i;
   for (i=0; i < size; i++)
   {
      Cout<< a[i];
   }
Cout<<"\n";
}

int main()
{
   int arr[5] = {3, 95, 63, 22, 47};
   int len = 5;

   cout<<"Given array: \n";
   printArray(arr, len);

   // calling merge sort
```

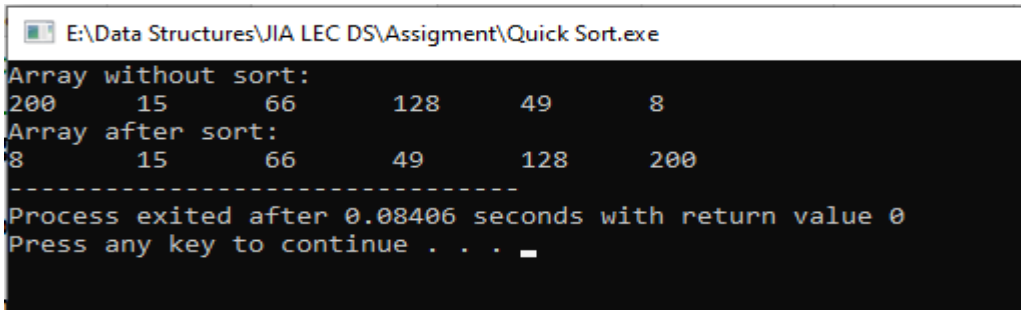
```
    mergeSort(arr, 0, len - 1);

    cout<<"\nSorted array: \n";
    printArray(arr, len);
    return 0;
}
```

```
E:\Data Structures\JIA LEC DS\Assigment\Merge sort.exe
Given array:
3 95 63 22 47

Sorted array:
3 22 47 63 95

--------------------------------
Process exited after 0.1226 seconds with return value 0
Press any key to continue . . .
```

5. Quick Sort

```
# include <iostream>
#include <conio.h>
using namespace std;
 int quicksort(int [], int, int);
int partition(int [], int, int);

int main(){
int i , arr[]={200,15,66,128,49,8};
cout<<"Array without sort: "<<endl;
for(i=0; i<=5;i++)
{
        cout<<arr[i]<<"\t";
}
cout<<"\n";
quicksort(arr,0,5);
cout<<"Array after sort: "<<endl;
for(i=0; i<=5;i++)
{
        cout<<arr[i]<<"\t";
}

}

quicksort(int A[],int start,int stop)
{
```

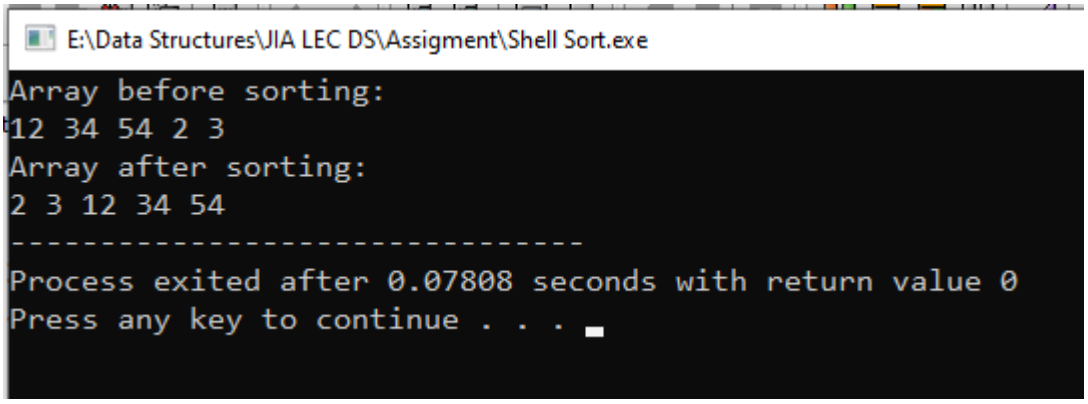
```
	int split;
	if(start<stop)
	{
		split= partition(A,start,stop);
		//sort left & right half's
		quicksort(A,start, split-1);
		quicksort(A,split+1,stop);
	}
}

int partition(int A[], int start, int stop)
{
int temp, L, R, pivot = A[start];
L = start + 1;
R = stop;
	while(L<=R)
	{
		while(A[L]<=pivot && L<=R)
		{
			L++;
		}
			while(A[R]>=pivot && R>=L)
		{
			R--;
		}
		if(R<L)
		{
			temp = A[start];
			A[start] = A[R];
			A[R]= temp;
		}
		else
		{
			temp = A[L];
			A[L] = A[R];
			A[R]= temp;
			L++;
			R--;
		}
	}
	return R;
}
```



6. Shell Sort

```
// C++ implementation of Shell Sort
#include <iostream>
using namespace std;

/* function to sort arr using shellSort */
int shellSort(int arr[], int n)
{
	// Start with a big gap, then reduce the gap
	for (int gap = n/2; gap > 0; gap /= 2)
	{
		// Do a gapped insertion sort for this gap size.
		// The first gap elements a[0..gap-1] are already in gapped order
		// keep adding one more element until the entire array is
		// gap sorted
		for (int i = gap; i < n; i += 1)
		{
			// add a[i] to the elements that have been gap sorted
			// save a[i] in temp and make a hole at position i
			int temp = arr[i];

			// shift earlier gap-sorted elements up until the correct
			// location for a[i] is found
			int j;
			for (j = i; j >= gap && arr[j - gap] > temp; j -= gap)
				arr[j] = arr[j - gap];

			// put temp (the original a[i]) in its correct location
			arr[j] = temp;
		}
	}
	return 0;
}

void printArray(int arr[], int n)
```

```
{
	for (int i=0; i<n; i++)
		cout << arr[i] << " ";
}

int main()
{
	int arr[5] = {12, 34, 54, 2, 3}, i;
	int n = 5;

	cout << "Array before sorting: \n";
	printArray(arr, n);

	shellSort(arr, n);

	cout << "\nArray after sorting: \n";
	printArray(arr, n);

	return 0;
}
```



7. HeapSort

```
// C++ program for implementation of Heap Sort
#include <iostream>

using namespace std;

// To heapify a subtree rooted with node i which is
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
```

```
	int largest = i; // Initialize largest as root
	int l = 2 * i + 1; // left = 2*i + 1
	int r = 2 * i + 2; // right = 2*i + 2

	// If left child is larger than root
	if (l < n && arr[l] > arr[largest])
		largest = l;

	// If right child is larger than largest so far
	if (r < n && arr[r] > arr[largest])
		largest = r;

	// If largest is not root
	if (largest != i) {
		swap(arr[i], arr[largest]);

		// Recursively heapify the affected sub-tree
		heapify(arr, n, largest);
	}
}

// main function to do heap sort
void heapSort(int arr[], int n)
{
	// Build heap (rearrange array)
	for (int i = n / 2 - 1; i >= 0; i--)
		heapify(arr, n, i);

	// One by one extract an element from heap
	for (int i = n - 1; i > 0; i--) {
		// Move current root to end
		swap(arr[0], arr[i]);

		// call max heapify on the reduced heap
		heapify(arr, i, 0);
	}
}

/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
	for (int i = 0; i < n; ++i)
		cout << arr[i] << " ";
	cout << "\n";
```

```
}

// Driver code
int main()
{
	int arr[] = { 12, 11, 13, 5, 6, 7 };
	int n = 6;

	heapSort(arr, n);

	cout << "Sorted array is \n";
	printArray(arr, n);
}
```