SORTING

SELECTION SORT

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


void selectionsort(int input[], int n){
 for (int i=0; i<n-1 ;i++){
       int min=input[i], minindex=i;
for (int j=i+1;j<n;j++ ){
    if (input[j]<min){
        min = input [j];
        minindex =j;
    }
}
int temp=input[i];
input[i ]=input[minindex];
input[minindex] = temp;
 }
}

int main (){
int input[]={3,1,6,9,0,4};
selectionsort(input ,6);
for(int i=0;i<6;i++){
    cout <<input[i]<<" ";
}
}
```

BUBBLE SORT

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

void bubblesort( int arr[], int n){
    for (int j=0; j<n; j++){
    for (int i=0;i<n-j-1 ;i++){
        if(arr[i]>arr[i+1]){
        int temp=arr[i];
        arr[i]=arr[i+1];
        arr[i+1]= temp;

        }
    }
}
}

int main (){
    int arr[]={8,6,7,4,6,3,5};
    bubblesort(arr,7);
    for(int i=0; i<7; i++){
        cout << arr[i]<<" ";
    }
}
```

INSERTION SORT

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

void insertionsort(int arr[],int n){
    for (int i=1; i<n ; i++){
        int current = arr[i];
        int j;
         for ( j=i-1; j>=0 ; j--){
            if( current < arr[j]){
                arr[j+1]=arr[j];
            }
            else {
                break ;
            }
         }
         arr[j+1]= current ;
    }
}

int main (){
    int arr[]={8,6,7,4,6,3,5};
    insertionsort(arr,7);
    for(int i=0; i<7; i++){
        cout << arr[i]<<" ";
    }
}
```

QUICK SORT

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

// A utility function to swap two elements
void swap(int* a, int* b)
{
    int t = *a;
    *a = *b;
    *b = t;
}

/* This function takes last element as pivot, places
the pivot element at its correct position in sorted
array, and places all smaller (smaller than pivot)
to left of pivot and all greater elements to right
of pivot */
int partition(int arr[], int low, int high)
{
    int pivot = arr[high]; // pivot
    int i = (low-1); // Index of smaller element and indicates
                // the right position of pivot found so far
   for (int j = low; j <= high - 1; j++) {
        // If current element is smaller than the pivot
        if (arr[j] < pivot) {
            i++; // increment index of smaller element
            swap(&arr[i], &arr[j]);
        }
    }
    swap(&arr[i + 1], &arr[high]);
    return (i + 1);
}

/* The main function that implements QuickSort
arr[] --> Array to be sorted,
low --> Starting index,
high --> Ending index */
void quickSort(int arr[], int low, int high)
{
    if (low < high) {
        /* pi is partitioning index, arr[p] is now
        at right place */
        int pi = partition(arr, low, high);

        // Separately sort elements before
        // partition and after partition
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);
    }
}

/* 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 Code
int main()
{
    int arr[] = { 10, 7, 8, 9, 1, 5 };
    int n = sizeof(arr) / sizeof(arr[0]);
    quickSort(arr, 0, n - 1);
    cout << "Sorted array: \n";
    printArray(arr, n);
    return 0;
}
```

RADIX SORT

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

// A utility function to get maximum value in arr[]
int getMax(int arr[], int size)
{
    int max = arr[0];
    for (int i = 1; i < size; i++)
        if (arr[i] > max)
            max = arr[i];
    return max;
}

void CountingSort(int arr[], int size, int div)
{
    int output[size];
    int count[10] = {0};

    for (int i = 0; i < size; i++)
        count[ (arr[i]/div)%10 ]++;

    for (int i = 1; i < 10; i++)
        count[i] += count[i - 1];

    for (int i = size - 1; i >= 0; i--)
    {
        output[count[ (arr[i]/div)%10 ] - 1] = arr[i];
        count[ (arr[i]/div)%10 ]--;
    }

    for (int i = 0; i < size; i++)
        arr[i] = output[i];
}


void RadixSort(int arr[], int size)
{
    int m = getMax(arr, size);
    for (int div = 1; m/div > 0; div *= 10)
        CountingSort(arr, size, div);
}


int main()
{
        int size;
        cout<<"Enter the size of the array: "<<endl;
        cin>>size;
        int arr[size];
        cout<<"Enter "<<size<<" integers in any order"<<endl;
        for(int i=0;i<size;i++)
        {
                cin>>arr[i];
        }
   cout<<endl<<"Before Sorting: "<<endl;
   for(int i=0;i<size;i++)
        {
            cout<<arr[i]<<" ";
        }

        RadixSort(arr, size);

        cout<<endl<<"After Sorting: "<<endl;
    for(int i=0;i<size;i++)
        {
            cout<<arr[i]<<" ";
        }

    return 0;
}
```

MERGE SORT

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

void merge(int* arr, int si, int mid, int ei) {
    int finalSize = ei - si;
    int merged[finalSize];

    int mergePointer = 0;
    int i = si, j = mid+1;

    while(i <= mid && j <= ei)
    {
        if(arr[i] <= arr[j]) {
            merged[mergePointer] = arr[i];
            i++;
            mergePointer++;
        } else {
            merged[mergePointer] = arr[j];
            j++;
            mergePointer++;
        }
    }

    while(i <= mid) {
        merged[mergePointer] = arr[i];
        mergePointer++;
        i++;
    }

    while(j <= ei) {
        merged[mergePointer] = arr[j];
        mergePointer++;
        j++;
    }

    int k = 0;
    for(i = si; i <= ei; i++) {
        arr[i] = merged[k];
        k++;
    }
}

void mergeSortHelper(int* arr, int si, int ei) {
    if(si >= ei)
        return;

    int mid = (si + ei)/2;

    mergeSortHelper(arr, si, mid);
    mergeSortHelper(arr, mid+1,ei);

    merge(arr, si, mid, ei);
}

void mergeSort(int* arr, int n) {
    int si = 0;
    int ei = n-1;

    mergeSortHelper(arr, si, ei);
}

int main() {

    int arr[5] = {2,53,7,26,4};
    mergeSort(arr, 5);

    for (int i = 0; i < 5; i++)
    {
        cout << arr[i] << " ";
    }

}
```