# Assignment 4(Sorting)

1. Implementation of :
2. Bubble Sort
3. Insertion Sort
4. Merge Sort
5. Quick Sort
6. Counting Sort
7. Radix Sort

CODE :

#include<stdio.h>

#include<stdlib.h>

// Global Variables

int n;

// Functions

void setarray(int arr[])

{

    printf("Enter %d elements :\n", n);

    for (int i = 0; i < n; i++)

    {

        scanf("%d", &arr[i]);

    }

}

void getarray(int arr[])

{

    printf("Your elements are :\n");

    for (int i = 0; i < n; i++)

    {

        printf("%d ", arr[i]);

    }

    printf("\n\n");

}

void copyArray(int arr1[], int arr2[])

{

    for (int i = 0; i < n; i++)

    {

        arr1[i] = arr2[i];

    }

}

int findMax(int arr[])

{

    int max = arr[0];

    for (int i = 1; i < n; i++)

    {

        if (max < arr[i])

        {

            max = arr[i];

        }

    }

    return max;

}

void bubbleSort(int arr[])

{

    for(int i = 0;i < n-1;i++)

    {

        for(int j = 0;j<n-i-1;j++)

        {

            if(arr[j]>arr[j+1])

            {

                int temp = arr[j];

                arr[j] = arr[j+1];

                arr[j+1] = temp;

            }

        }

    }

}

void selectionSort(int arr[])

{

    for(int i = 0;i<n-1;i++)

    {

        int smallest = i;

        for(int j = i; j<n; j++)

        {

            if(arr[smallest]>arr[j])

            {

                smallest = j;

            }

        }

        int temp = arr[i];

        arr[i] = arr[smallest];

        arr[smallest] = temp;

    }

}

void insertionSort(int arr[])

{

    for(int i = 1; i<n;i++)

    {

        int key = arr[i];

        int j = i-1;

        while(key<arr[j] && j>=0)

        {

            arr[j+1] = arr[j];

             j--;

        }

        arr[j+1] = key;

    }

}

void mergeConqueror(int arr[],int si,int mid,int ei)

{

    int idx1 = si;

    int idx2 = mid+1;

    int x = 0;

    int sorter[ei-si+1];

    while(idx1<=mid && idx2<=ei)

    {

        if(arr[idx1]<=arr[idx2])

        {

            sorter[x++] = arr[idx1++];

        }

        else

        {

            sorter[x++] = arr[idx2++];

        }

    }

    while(idx1<=mid)

    {

        sorter[x++] = arr[idx1++];

    }

    while(idx2<=ei)

    {

        sorter[x++] = arr[idx2++];

    }

    for(int i=0,j=si;i<x;i++,j++)

    {

        arr[j] = sorter[i];

    }

}

void mergeSort(int arr[],int si,int ei)

{

    if(si<ei)

    {

        int mid = si + (ei-si)/2;

        mergeSort(arr,si,mid);

        mergeSort(arr,mid+1,ei);

        mergeConqueror(arr,si,mid,ei);

    }

}

int quickPartition(int arr[],int si,int ei)

{

    int pivot = arr[si];

    int i = si +1;

    int j = ei;

    do

    {

        while(pivot>=arr[i])

        {

            i++;

        }

        while(pivot<arr[j])

        {

            j--;

        }

        if(i<j)

        {

            int temp = arr[i];

            arr[i] = arr[j];

            arr[j] = temp;

        }

    } while(i<j);

    int temp = arr[si];

    arr[si] = arr[j];

    arr[j] = temp;

    return si;

}

void quickSort(int arr[],int si,int ei)

{

    if(si<ei)

    {

        int pivotIdx = quickPartition(arr,si,ei);

        quickSort(arr,si,pivotIdx-1);

        quickSort(arr,pivotIdx+1,ei);

    }

}

void countSort(int arr[])

{

    int max = findMax(arr);

    int \*counter = (int\*)calloc(max+1,sizeof(int));

    int \*final = (int\*)calloc(n,sizeof(int));

    for(int i = 0;i<n;i++)

    {

        counter[arr[i]]++;

    }

    for(int i = 1;i<max+1;i++)

    {

        counter[i] += counter[i-1];

    }

    for(int i=n-1;i>=0;i--)

    {

        final[--counter[arr[i]]] = arr[i];

    }

    copyArray(arr,final);

    free(counter);

    free(final);

}

void rCountSort(int arr[],int exp)

{

    int range = 10;

    int \*counter = (int\*)calloc(range,sizeof(int));

    int \*final = (int\*)calloc(n,sizeof(int));

    for(int i = 0;i<n;i++)

    {

        counter[(arr[i]/exp)%10]++;

    }

    for(int i = 1;i<range;i++)

    {

        counter[i] += counter[i-1];

    }

    for(int i = n-1;i>=0;i--)

    {

        final[--counter[(arr[i]/exp)%10]] = arr[i];

    }

    copyArray(arr,final);

    free(counter);

    free(final);

}

void radixSort(int arr[])

{

    int max = findMax(arr);

    for(int exp = 1;max/exp>0;exp\*=10)

    {

        rCountSort(arr,exp);

    }

}

void main()

{

    while(1)

    {

        printf("Enter no. of elements : ");

        scanf("%d",&n);

        int arr[n];

        setarray(arr);

        printf("Enter Algorithm :\n");

        printf("1 for Bubble Sort.\n");

        printf("2 for Insertion Sort.\n");

        printf("3 for Merge Sort.\n");

        printf("4 for Quick Sort.\n");

        printf("5 for Counting Sort.\n");

        printf("6 for Radix Sort.\n");

        printf("0 for exit.\n");

        int ch;

        scanf("%d",&ch);

        switch(ch)

        {

            case 0:

                printf("Exited Succesfully.\n");

                return;

            case 1:

                bubbleSort(arr);

                getarray(arr);

                break;

            case 2:

                insertionSort(arr);

                getarray(arr);

                break;

            case 3:

                mergeSort(arr,0,n-1);

                getarray(arr);

                break;

            case 4:

                quickSort(arr,0,n-1);

                getarray(arr);

                break;

            case 5:

                countSort(arr);

                getarray(arr);

                break;

            case 6:

                radixSort(arr);

                getarray(arr);

                break;

            default:

                printf("Error Try again.\n");

                break;

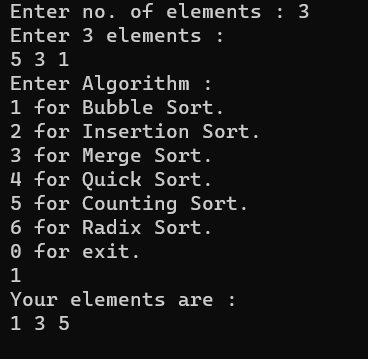
        }

    }

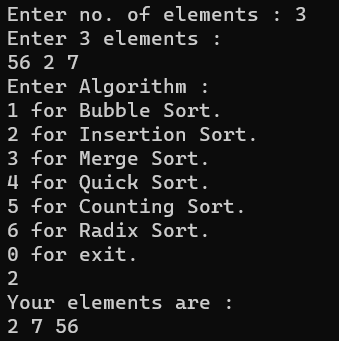
}

OUTPUT :

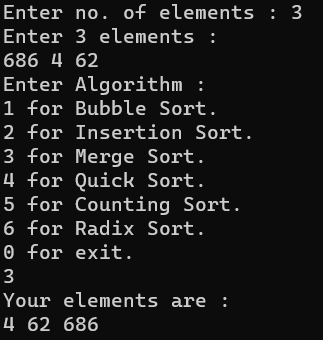
1. Bubble Sort :



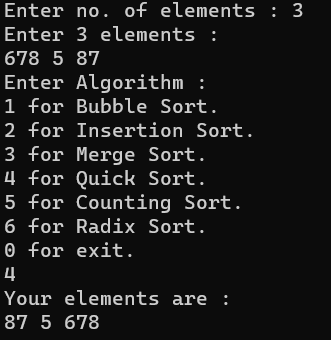
1. Insertion Sort :



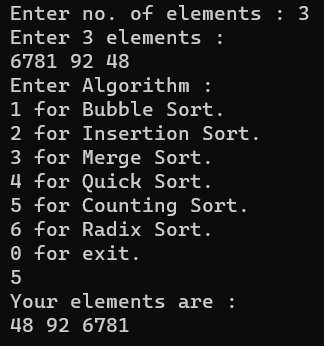
1. Merge Sort :



1. Quick Sort :



1. Counting Sort :



1. Radix Sort :

