ASSIGNMENT VII:

Bubble And Merge Sort

*U19CS012 [D-12]*

1.) Implement the Bubble Sort Algorithm

Code:

*//Implement Bubble Sort Algorithm*

*#include* <stdio.h>

*#define* MAX 10001

*//Predefined a Static Array of MAX Size*

int arr[MAX];

*//1 -> Simple Bubble Sort Implementation Always -> 0(n^2)*

void Bubble\_Sort(int arr[], int n);

*//2 -> Optimised Bubble Sort Implementation*

*//Takes Less Iterations if Array is Already Sorted*

void Optimized\_Bubble\_Sort(int arr[], int n);

*//Small Helper Function to Print the Array*

void print(int arr[], int sz);

int main()

{

    int n;

    printf("\nEnter the Number of Elements [Max: 1e5] to Sort : ");

    scanf("%d", &n);

*// Invalid Input Entered*

*if* (n < 0)

    {

        printf("\nInvalid Input!\nEnter Positive Number of Elements [>0]!!\n");

*return* 0;

    }

*// No Element to Sort*

*if* (n == 0)

    {

        printf("\nWe Need to Enter (atleast) One Element to Sort!!");

*return* 0;

    }

    printf("\nEnter the Values of Array to Sort : \n");

*for* (int i = 0; i < n; i++)

    {

        printf("arr[%d] = ", i);

        scanf("%d", &arr[i]);

    }

    int choice;

    printf("Which Bubble Sort to Use \n1 -> Simple \n2 -> Optimized\n");

    printf("Choice : ");

    scanf("%d", &choice);

*switch* (choice)

    {

*case* 1:

        Bubble\_Sort(arr, n);

*break*;

*case* 2:

        Optimized\_Bubble\_Sort(arr, n);

*break*;

*default*:

        printf("Enter a Valid Choice!");

*break*;

    }

*return* 0;

}

void Bubble\_Sort(int arr[], int n)

{

*// Iterators*

    int i, j;

*// tmp Variable For Swapping*

    int tmp;

    printf("Initial Array : ");

    print(arr, n);

    printf("\n");

    int pass = 1;

*for* (i = 0; i < n - 1; i++)

    {

*for* (j = 0; j < n - i - 1; j++)

*if* (arr[j] > arr[j + 1])

            {

                tmp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = tmp;

            }

        printf("Array After Pass %d : ", pass);

        print(arr, n);

        printf("\n");

        pass += 1;

    }

    printf("Sorted Array : ");

    print(arr, n);

    printf("\n");

}

void Optimized\_Bubble\_Sort(int arr[], int n)

{

*// Iterators*

    int i, j;

*// tmp Variable For Swapping*

    int tmp;

    printf("Initial Array : ");

    print(arr, n);

    printf("\n");

    int pass = 1;

    int flag = 0;

*for* (i = 0; i < n - 1; i++)

    {

        flag = 0;

*for* (j = 0; j < n - i - 1; j++)

*if* (arr[j] > arr[j + 1])

            {

                tmp = arr[j];

                arr[j] = arr[j + 1];

                arr[j + 1] = tmp;

                flag = 1;

            }

        printf("Array After Pass %d : ", pass);

        print(arr, n);

        printf("\n");

        pass += 1;

*// flag -> 0 ; When No Element is Swapped*

*if* (flag == 0)

        {

*break*;

        }

    }

    printf("Sorted Array : ");

    print(arr, n);

    printf("\n");

}

void print(int arr[], int sz)

{

    int i;

*for* (i = 0; i < sz; i++)

        printf("%d ", arr[i]);

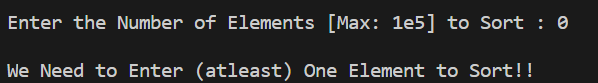
    printf("\n");

}

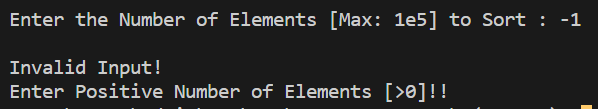
Test Cases:

A.) *Check for Invalid inputs*

-> No Element to Sort {0}

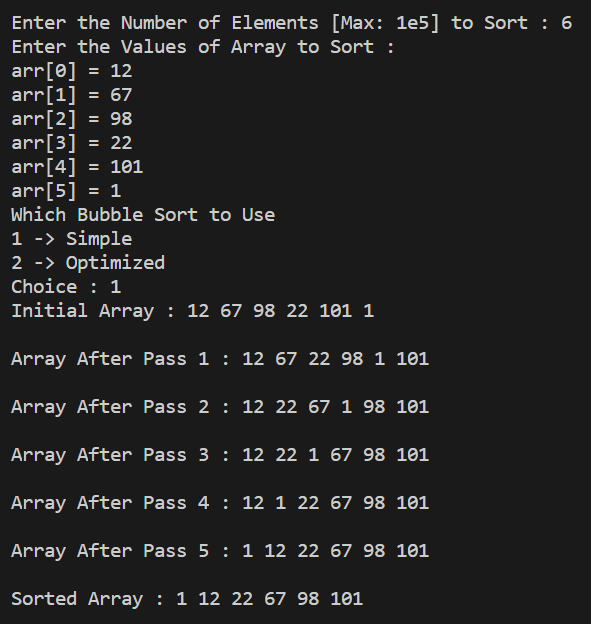


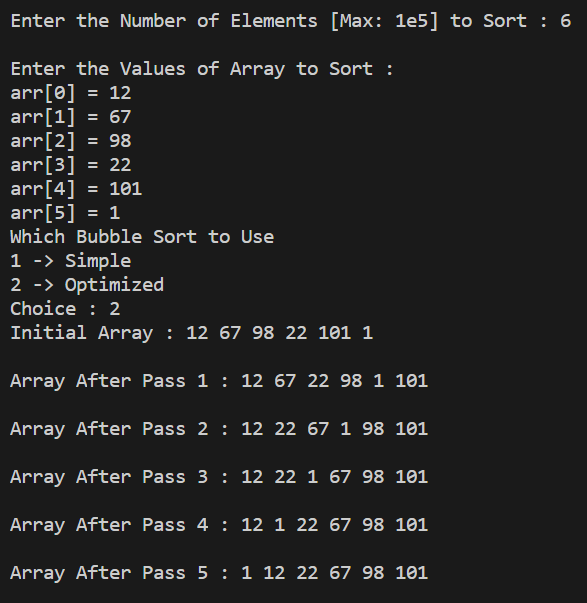
-> Invalid Input {-1}



B.) *Random Unsorted Array*

{12,67,98,22,101,1} -> {1,12,22,67,98,101} [Both Simple & Optimized]



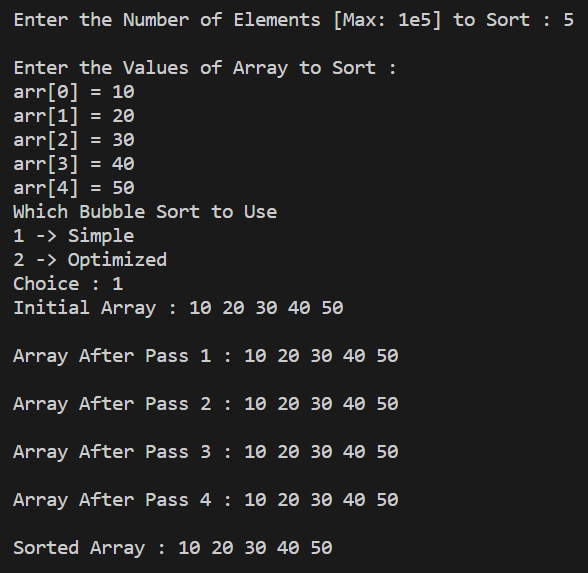


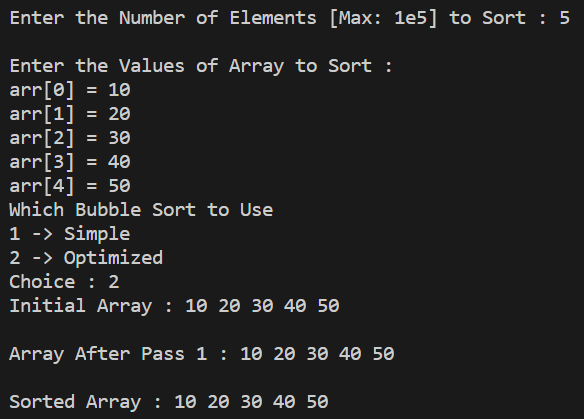
C.) *Sorted Array*

{10, 20, 30, 40, 50} -> {10, 20, 30, 40, 50} [Simple] (n-1 Iterations)

{10, 20, 30, 40, 50} -> {10, 20, 30, 40, 50} [Optimized] (1 Iteration)

[If Array is Sorted, Loop Breaks in Optimized]





2) Implement the Merge Sort Algorithm

Code:

*//Implement Bubble Sort Algorithm*

*#include* <stdio.h>

*#define* MAX 10001

*//Predefined a Static Array of MAX Size*

int arr[MAX];

*//Recursive Function To Keep on Dividing the Array*

void MergeSort(int arr[], int l, int r);

*//Function to Merge the Divided Sub-Arrays*

void merge(int arr[], int start, int mid, int end);

*//Small Helper Function to Print the Array*

void print(int arr[], int sz);

int main()

{

    int n;

    printf("\nEnter the Number of Elements [Max: 1e5] to Sort : ");

    scanf("%d", &n);

*// Invalid Input Entered*

*if* (n < 0)

    {

        printf("\nInvalid Input!\nEnter Positive Number of Elements [>0]!!\n");

*return* 0;

    }

*// No Element to Sort*

*if* (n == 0)

    {

        printf("\nWe Need to Enter (atleast) One Element to Sort!!");

*return* 0;

    }

    printf("\nEnter the Values of Array to Sort : \n");

*for* (int i = 0; i < n; i++)

    {

        printf("arr[%d] = ", i);

        scanf("%d", &arr[i]);

    }

    printf("Initial Array : ");

    print(arr, n);

    printf("\n");

    MergeSort(arr, 0, n - 1);

    printf("Sorted Array : ");

    print(arr, n);

    printf("\n");

*return* 0;

}

void merge(int arr[], int start, int mid, int end)

{

*// create a temp array*

    int temp[end - start + 1];

*// Iterators for both intervals*

    int i = start, j = mid + 1;

*// Iterators for temp*

    int k = 0;

*// traverse both arrays and in each iteration add smaller of both elements in temp*

*while* (i <= mid && j <= end)

    {

*if* (arr[i] <= arr[j])

        {

            temp[k] = arr[i];

            k += 1;

            i += 1;

        }

*else*

        {

            temp[k] = arr[j];

            k += 1;

            j += 1;

        }

    }

*// add elements left in the first interval*

*while* (i <= mid)

    {

        temp[k] = arr[i];

        k += 1;

        i += 1;

    }

*// add elements left in the second interval*

*while* (j <= end)

    {

        temp[k] = arr[j];

        k += 1;

        j += 1;

    }

*// Copy Elements Back to Original Array*

*for* (i = start; i <= end; i += 1)

    {

        arr[i] = temp[i - start];

    }

}

*//Recursive Function To Keep on Dividing the Array*

void MergeSort(int arr[], int start, int end)

{

    int n = start + end + 1;

    int mid;

*if* (start < end)

    {

*// To Avoid Overflow*

        mid = start + (end - start) / 2;

*// Divide and Sort First Half*

        MergeSort(arr, start, mid);

*// Divide and Sort Second Half*

        MergeSort(arr, mid + 1, end);

*//Now Merge the Two Halfs*

        merge(arr, start, mid, end);

    }

}

void print(int arr[], int sz)

{

    int i;

*for* (i = 0; i < sz; i++)

        printf("%d ", arr[i]);

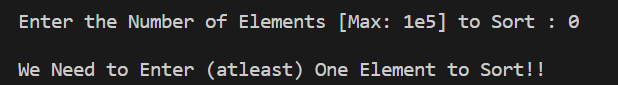
    printf("\n");

}

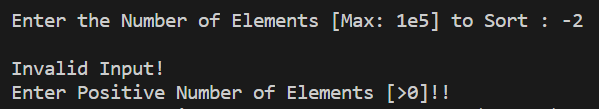
Test Cases:

A.) *Check for Invalid inputs*

-> No Element to Sort {0}



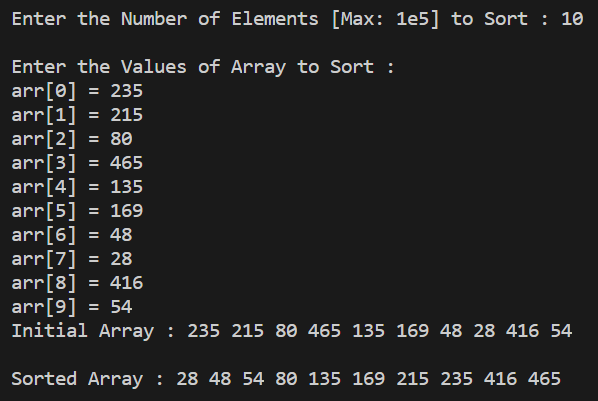
-> Invalid Input {-2}



B.)

{235, 215, 80, 465, 135, 169, 48, 28, 416, 54}

{28, 48, 54, 80, 135, 169, 215, 235, 416, 465}



Conclusion:

1.) It Doesn’t Matter the Array is *Sorted or Not*, Merge Sort Always takes O(n\*log(n)) Complexity .i.e. It will Always Divide & Merge the Array No Matter What the Array is! [Non-Adaptive Algorithm]

2.) Similar thing can be said for Bubble Sort [Simple One], it will always takes

O (n\*n) Complexity .i.e. will Check *n\*(n-1)/2 Pairs* for Sorting. This can be taken care using *Flag* in Optimized Bubble Sort!