[LAB SUBMISSION – 6](https://github.com/shrishtinigam/DSA_Lab/tree/main/Lab_6)  <-link

# Meher Shrishti Nigam – 20BRS1193

BUBBLE SORT – REGULAR AND MODIFIED BUBBLE SORT

*// BubbleSort.c*

#include <stdio.h>

void bubblesort\_1(int *arr*[], int *n*);

void bubblesort\_2(int *arr*[], int *n*);

void bubblesort\_3(int *arr*[], int *n*);

*// At the first passing, the largest element is moved to the end of the list. Similarly, at the second passing*

*// the second largest element is moved to the second last index of the list.*

*// Time complexity:O(n\*n)*

*// Sort stability: Stable sort*

*// Space complexity: O(1) - inplace*

int main()

{

    int x, n, hold;

    int flag = 0;

    printf("Number of integers to sort: ");

    scanf("%d", &n);

    int arr[n];

    printf("Enter the integer values\n");

*// Takes in the values from the user*

    for(int i = 0; i < n; i++)

    {

        scanf("%d", &x);

        arr[i] = x;

    }

    bubblesort\_3(arr, n);

    printf("Sorted Array - ");

    for (int a = 0; a < n; ++a)

    {

        printf(" %d", arr[a]);

    }

}

*// Worst method, will always go through the n\*n iterations*

void bubblesort\_1(int *arr*[], int *n*)

{

    int hold;

    for(int i = 0; i < n-1; i++)

    {

        for(int i = 0; i < n-1; i++)

        {

            if(arr[i]>arr[i+1])

            {

                hold = arr[i];

                arr[i] = arr[i+1];

                arr[i+1] = hold;

            }

        }

    }

}

*// Second method, if the total number of swaps during a pass is 0 then it exits as the list is sorted*

void bubblesort\_2(int *arr*[], int *n*)

{

    int flag, hold;

    do{

        flag = 0;

        for(int i = 0; i < n-1; i++)

        {

            if(arr[i]>arr[i+1])

            {

                hold = arr[i];

                arr[i] = arr[i+1];

                arr[i+1] = hold;

                flag = flag + 1;

            }

        }

    }while(flag != 0);

}

*/\*OPTIMIZED BUBBLE SORT\*/*

*// Best method, along with exiting once the list is sorted (like second method), the value of n decreases every*

*// iteration (by 1). This is because after a pass, the largest element of the pass is moved to the last and we*

*// don't need to check for swaps there.*

void bubblesort\_3(int *arr*[], int *n*)

{

    int flag, hold;

    do{

        flag = 0;

        for(int i = 0; i < n-1; i++)

        {

            if(arr[i]>arr[i+1])

            {

                hold = arr[i];

                arr[i] = arr[i+1];

                arr[i+1] = hold;

                flag = flag + 1;

            }

        }

        n--;  *// to remove unneccessary comparisions towards the end*

    }while(flag != 0);

}

*// Another code for optimized bubble sort*

*/\**

*int flag = 0;*

*for(int j = 0; j < n -1; j++)*

*{*

*flag = 0;*

*for(int i = 0; i < n-j-1; i++)*

*{*

*if(arr[i] > arr[i+1])*

*{*

*hold = arr[i];*

*arr[i] = arr[i+1];*

*arr[i+1] = hold;*

*flag = 1;*

*}*

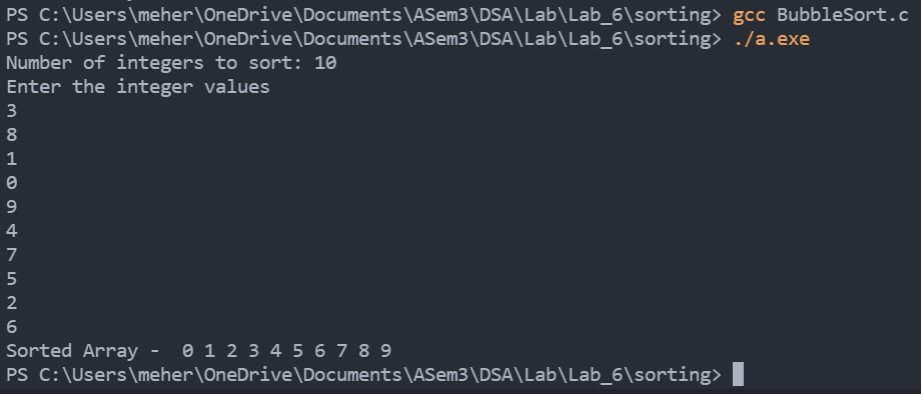
*}*

*if(flag == 0)*

*break;*

*}*

*\*/*



SELECTION SORT

#include <stdio.h>

*// This program accepts values from users and performs selection sort*

*// O(n^2) time complexity*

*// Sort Stability - unstable sort*

*// O(1) space complexity*

*// Stable with O(n) extra space or when using linked lists, or when made as a variant of Insertion Sort instead of swapping the two items*

int main()

{

    int x, n, hold, hold2;

    printf("Number of integers to sort: ");

    scanf("%d", &n);

    int arr[n];

    printf("Enter the integer values\n");

*// Takes in the values from the user*

    for(int i = 0; i < n; i++)

    {

        scanf("%d", &x);

        arr[i] = x;

    }

    int j = 0;

    while(j<n)

    {

        x = arr[j]; *// Assume the first element of the unsorted part of the array is the smallest element*

        hold2 = j;

        for(int i = j;  i < n; i++) *// So, iterating over the unsorted part of the array,*

        {

            if(x > arr[i]) *// If our assumption is greater than an element,*

            {

                x = arr[i]; *// Change assumption to that element*

                hold2 = i; *// Remember the location of that element in 'hold2' var*

            }

        } *// Repeat till the end, now you have the smallest element in the unsorted array*

*// Now you have to swap the first element of the unsorted array with the smallest element in the unsorted array*

        hold = arr[j]; *// Save the value of the first element of unsorted array in 'hold' var*

        arr[j] = x; *// Put the smallest element at the beginning of the unsorted array*

        arr[hold2] = hold; *// Put the value stored in 'hold' var in the location of the smallest elemnt (which we saved in 'hold2' var)*

        j = j + 1;

    }

    printf("Sorted Array - ");

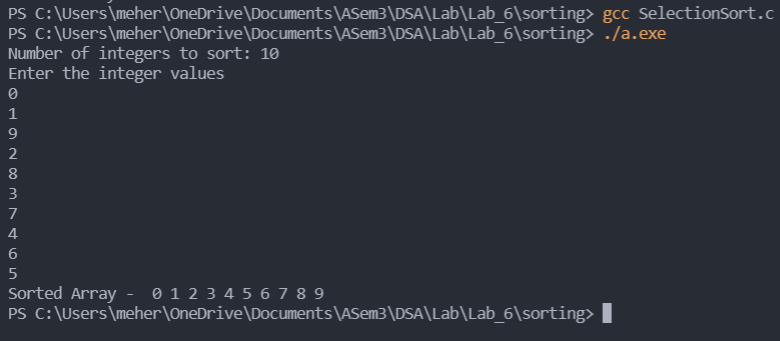
    for (int a = 0; a < n; ++a)

    {

        printf(" %d", arr[a]);

    }

}



INSERTION SORT

#include <stdio.h>

void InsertionSort(int *arr*[], int *n*);

int main()

{

    int n, x;

    printf("Number of integers to sort: ");

    scanf("%d", &n);

    int arr[n];

    printf("Enter the integer values\n");

*// Takes in the values from the user*

    for(int i = 0; i < n; i++)

    {

        scanf("%d", &x);

        arr[i] = x;

    }

    InsertionSort(arr, n);

    printf("Sorted Array - ");

    for(int i = 0; i < n; i++)

    {

        printf("%d ", arr[i]);

    }

}

*// https://www.youtube.com/watch?v=OAyj2d-GH0c*

*// TC - BEST CASE - O(n)*

*// TC - WORST CASE CASE - O(n^2)*

*// SPACE COMPLEXITY - O(1) - IN PLACE*

*// Sort Stability - Stable sort*

*// its most helpful when you have an almost sorted array*

*// used in online query - when elements are coming in one by one at random*

void InsertionSort(int *arr*[], int *n*)

{

    for(int i = 1; i < n ; i++)

    {

        int key = arr[i];

*// start of the unsorted region*

        int j = i -1;

*// element in the unsorted region is less than the current element*

        while(j>=0 && arr[j]>key) *// if the back of sorted region is greater than our key*

*//"arr[j]>key" - here for equal elements, the ones on theleft will remain of the left side, making the algorithm "stable"*

*//"arr[j]>=key" - here even if the element is equal to the current it will be moved to the right, and the cuurent element which was originally on the right side wiil come to the left, making the sort "unstable"*

*// Stability of a sort can matter when elements being sorted are dictionaried/mapped with other data.*

        {

            arr[j+1]=arr[j]; *// shift that to the right*

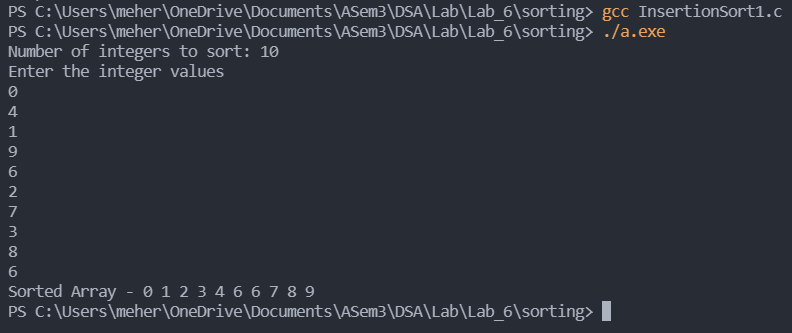
            j=j-1; *// decrement the index by 1*

        }

        arr[j+1]=key; *// put in the key at the appropriate place*

    }

}



BINARY SEARCH

#include <stdio.h>

void InsertionSort(int *arr*[], int *n*);

int main()

{

    int n, x;

    printf("Enter the integer values\n");

    scanf("%d", &n);

    int arr[n];

    printf("Number of integers to sort: ");

*// Takes in the values from the user*

    for(int i = 0; i < n; i++)

    {

        scanf("%d", &x);

        arr[i] = x;

    }

    InsertionSort(arr, n);

    printf("Sorted Array - ");

    for(int i = 0; i < n; i++)

    {

        printf("%d ", arr[i]);

    }

}

*// https://www.youtube.com/watch?v=OAyj2d-GH0c*

*// TC - BEST CASE - O(n)*

*// TC - WORST CASE CASE - O(n^2)*

*// SPACE COMPLEXITY - O(1) - IN PLACE*

*// Sort Stability - Stable sort*

*// its most helpful when you have an almost sorted array*

*// used in online query - when elements are coming in one by one at random*

void InsertionSort(int *arr*[], int *n*)

{

    for(int i = 1; i < n ; i++)

    {

        int key = arr[i];

*// start of the unsorted region*

        int j = i -1;

*// element in the unsorted region is less than the current element*

        while(j>=0 && arr[j]>key) *// if the back of sorted region is greater than our key*

*//"arr[j]>key" - here for equal elements, the ones on theleft will remain of the left side, making the algorithm "stable"*

*//"arr[j]>=key" - here even if the element is equal to the current it will be moved to the right, and the cuurent element which was originally on the right side wiil come to the left, making the sort "unstable"*

*// Stability of a sort can matter when elements being sorted are dictionaried/mapped with other data.*

        {

            arr[j+1]=arr[j]; *// shift that to the right*

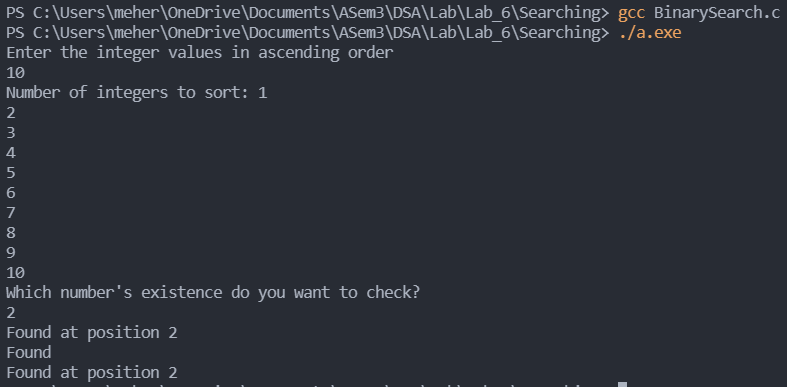
            j=j-1; *// decrement the index by 1*

        }

        arr[j+1]=key; *// put in the key at the appropriate place*

    }

}



QUICK SORT

#include <stdio.h>

*/\**

*\*   Comparision based sort*

*\*   Divide and Conquer*

*\*/*

int Partition(int *arr*[], int *beg*, int *end*)

{

    int pivot = arr[end];

    int i = beg-1;

    int hold;

    for(int j = beg; j <= end-1; j++)

    {

        if(arr[j] < pivot)

        {

            i++;

            hold = arr[j];

            arr[j] = arr[i];

            arr[i] = hold;

        }

    }

    hold = arr[end];

    arr[end] = arr[i+1];

    arr[i+1] = hold;

    return i + 1;

}

void QuickSort(int *arr*[], int *beg*, int *end*)

{

    if(beg < end)

    {

        int mid = Partition(arr, beg, end);

        QuickSort(arr, beg, mid - 1);

        QuickSort(arr, mid + 1, end);

    }

}

int main()

{

    int x, n, hold;

    int flag = 0;

    printf("Number of integers to sort: ");

    scanf("%d", &n);

    int arr[n];

    printf("Enter the integer values\n");

*// Takes in the values from the user*

    for(int i = 0; i < n; i++)

    {

        scanf("%d", &x);

        arr[i] = x;

    }

    QuickSort(arr, 0, n);

    printf("Sorted Array - ");

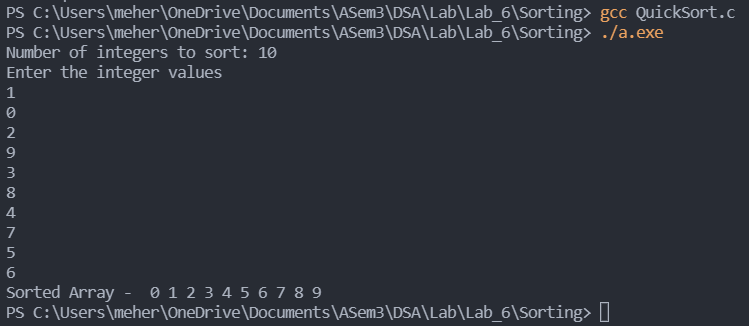
    for (int a = 0; a < n; ++a)

    {

        printf(" %d", arr[a]);

    }

}



MERGE SORT

#include <stdio.h>

void mergeSort(int *arr*[], int *l*, int *r*, int *size*);

void merge(int *arr*[],int *l*,int *m*, int *r*, int *size*);

int main()

{

    printf("Enter size of array: ");

    int size;

    scanf("%d", &size);

    int myarray[size];

    printf("Enter %d integers in any order: ", size);

    for(int i = 0; i < size; i++){

        scanf("%d", &myarray[i]);

    }

    printf("Before Sorting - \n");

    for (int i = 0; i < size; i++) {

        printf("%d ", myarray[i]);

    }

    printf("\n");

    mergeSort(myarray, 0, (size - 1), size);

    printf("After Sorting - \n");

    for (int i = 0; i < size; i++) {

        printf("%d ", myarray[i]);

    }

    return 0;

}

void mergeSort(int *arr*[], int *l*, int *r*, int *size*)

{

    if (l < r)

    {

        int m = (l + r)/2;

        mergeSort(arr, l, m, size);

        mergeSort(arr, m + 1, r, size);

        merge(arr, l, m, r, size);

    }

}

void merge(int *arr*[], int *l*, int *m*, int *r*, int *size*)

{

    int i = l;

    int j = m +1;

    int k = l;

    int temp[size];

    while(i<=m && j<=r)

    {

        if(arr[i]<=arr[j])

            temp[k++] = arr[i++];

        else

            temp[k++]=arr[j++];

    }

    while(i<=m)

        temp[k++] = arr[i++];

    while(j<=r)

        temp[k++] = arr[j++];

    for (int p = l; p <= r; p++)

        arr[p] = temp[p];

}

