***Assaignment 1***

|  |  |
| --- | --- |
| Course Name:  Course No: | Data Structure and Algorithm  ICT 5102 |

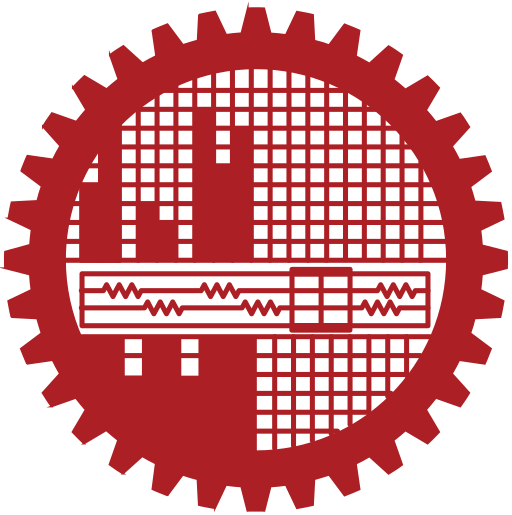
**Submited By:**

Sk. Md. Zubayer-Al-Mahmud

Student ID: 1017311014

Session: Oct, 2017

PGD in IICT ,BUET



Bangladesh University of Engineering and Technology (BUET)

**INSTITUTE OF**

**INFORMATION AND COMMUNICATION TECHNOLOGY (IICT)**

3/8/2019

# INTRODUCTION :

Insertion Sort, Merge Sort & Quick Sort Algorithms are used for sorting array .Here I use these algorithm separtely to sort given array with Total Comparison & Swap Count during Sort Process. Array Sorting from Ascending to Ascending, Descending to Ascending & given Random Order to Ascending with count.

# GIVEN INPUT ARRAY:

{195, 134, 144, 141, 145, 197, 177, 101, 196, 146, 175, 173, 154, 171, 111, 136, 115, 162,165,

192,131, 142, 120, 185, 102, 181, 107, 198, 106, 176, 121, 178, 119, 128, 193, 127, 123, 143,

155, 186,191, 122, 132, 158, 129, 183, 163, 180, 103, 188, 150, 151, 172, 118, 174, 170, 104,

130, 116, 117,112, 139, 194, 147, 153, 164, 169, 199, 148, 138, 200, 190, 126, 152, 161, 179,

149, 137, 133, 110,159, 113, 140, 160, 105, 184, 182, 135, 114, 125, 168, 189, 124, 108, 187,

166, 156, 109, 167, 157}

# MATERIALS:

1. Code Block IDE Tools

2. GDB/CDB debugger

4. Subline (Optional)

3. Paper and Pen

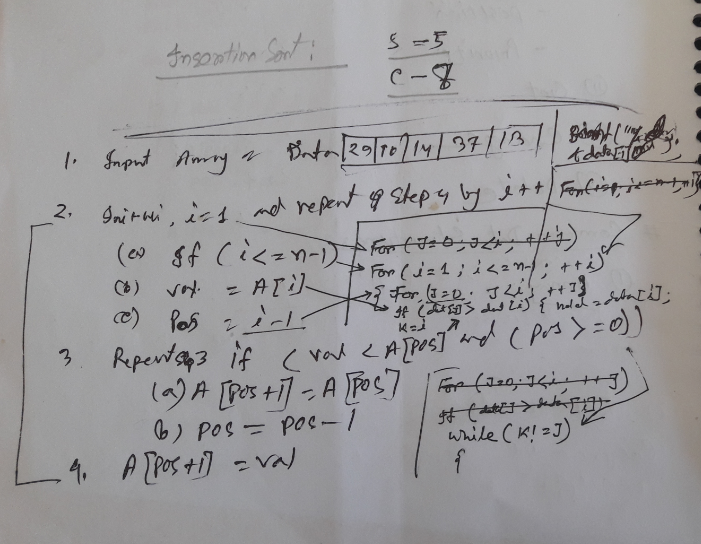
PROCEDURE:

Here, I am coding in C programming language by using Code Block IDE Tools. To complete full code, I take some step before coding for each sort by handwriting and sketching. these step are:

1. Writing algorithm
2. Writing code from algorithm
3. Scenario view from code
4. Typing code in CodeBlock and debugging to check
5. Finally merge these 3 types of sorting code and implementation

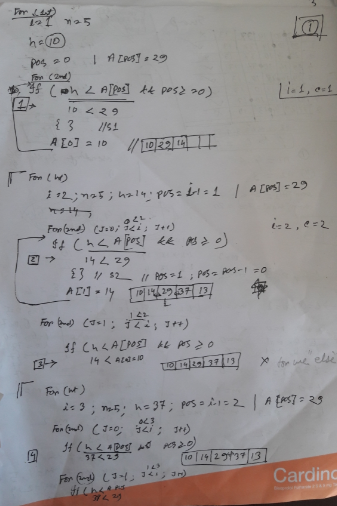
HANDWRITING AND SKETCH: In the below, I show my handwriting and sketch:

1. For insertion sort: a) algorithm-



b) A part of code from algorithm with debug

\



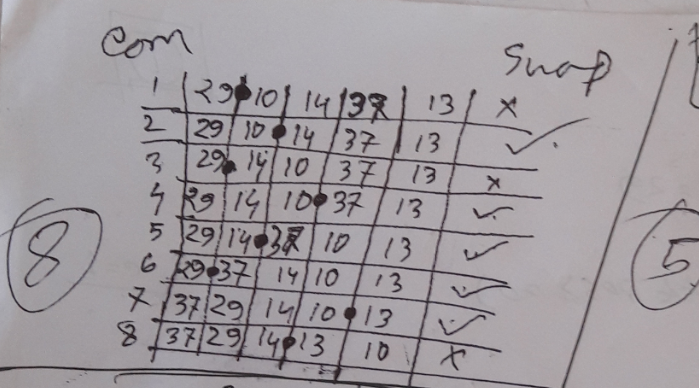
c) Scenario view of insertion sort in ascending order:

for {29, 10, 14, 37, 3}; array, comparison 8 and swap 5.

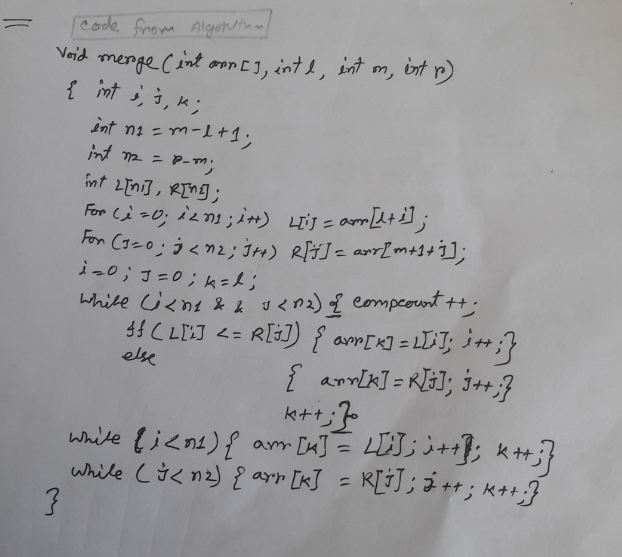
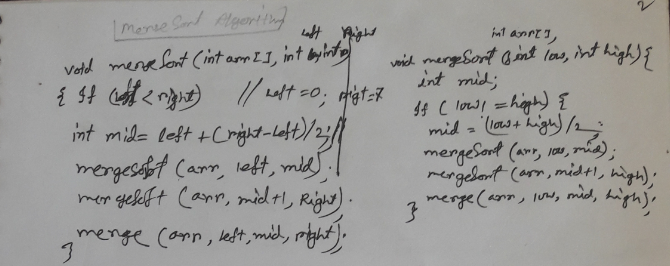


d) Scenario view of insertion sort in descending order:

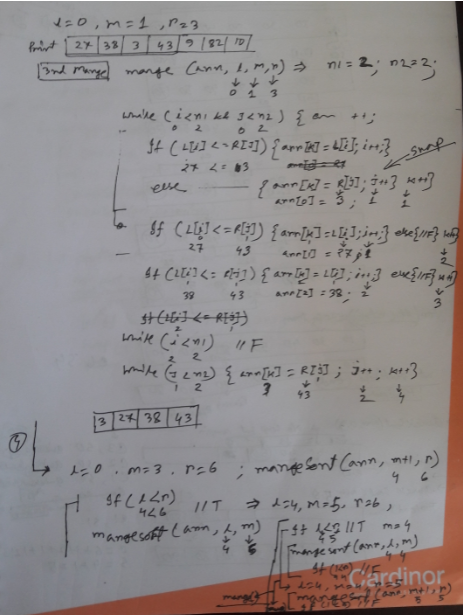
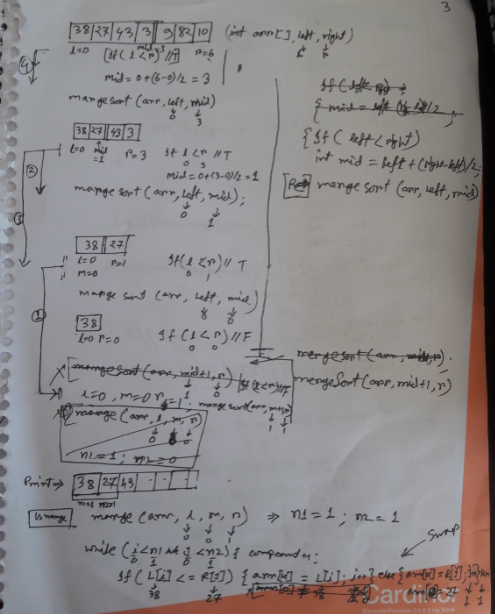
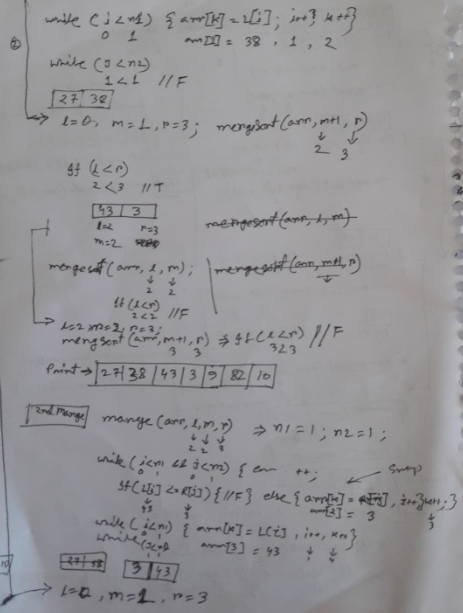
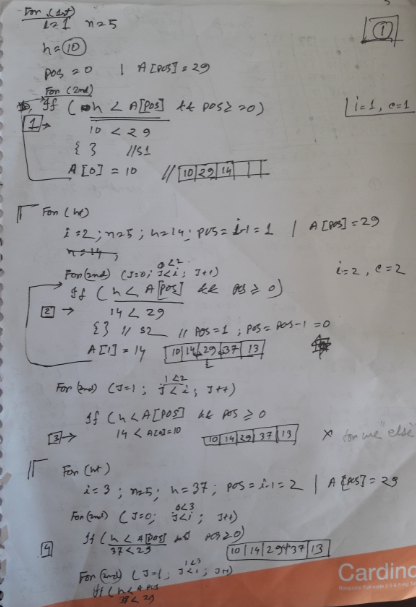
for {29, 10, 14, 37, 3}; array, comparison 8 and swap 5.



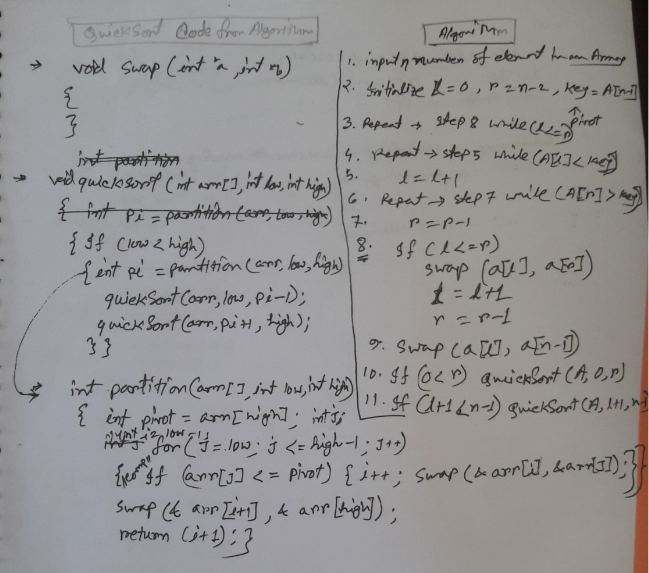
1. For Merse sort: a) algorithm and code-



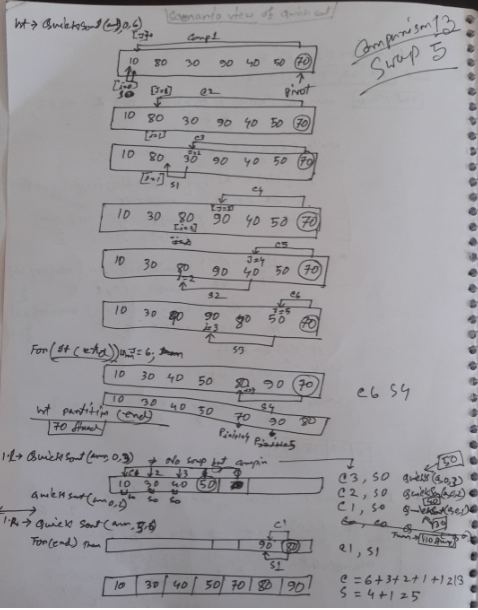
b) a part of code debugging and scenario::



1. For Quick sort: : a) algorithm and code-



b)a part of code debugging and scenario::



# FULL CODE FOR COMBAINED SORT WITH COMPARISON & SWAP:

**#include<stdio.h>**

**// int array[]={29, 10, 14, 37, 3};**

**// int array[]={38, 27, 43, 3, 9, 82, 10};**

**// int array[]={10, 80, 30, 90, 40, 50, 70};**

**int array[]={195, 134, 144, 141, 145, 197, 177, 101, 196, 146, 175, 173, 154, 171, 111, 136, 115, 162, 165, 192,**

**131, 142, 120, 185, 102, 181, 107, 198, 106, 176, 121, 178, 119, 128, 193, 127, 123, 143, 155, 186,**

**191, 122, 132, 158, 129, 183, 163, 180, 103, 188, 150, 151, 172, 118, 174, 170, 104, 130, 116, 117,**

**112, 139, 194, 147, 153, 164, 169, 199, 148, 138, 200, 190, 126, 152, 161, 179, 149, 137, 133, 110,**

**159, 113, 140, 160, 105, 184, 182, 135, 114, 125, 168, 189, 124, 108, 187, 166, 156, 109, 167, 157};**

**int n = sizeof(array)/sizeof(array[0]);**

**int A[]; // for tamporary array to copy from given narray**

**//0.1. utility function(userdefined function): arrayCopy// //zubayer-1017311014//**

**void arrayCopy(int copied[],int n)**

**{ int loop;**

**for(loop = 0; loop < n; loop++)**

**{ copied[loop] = array[loop];**

**}**

**}**

**//0.2. utility function(userdefined function): printArray//**

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

**{ int i;**

**for (i=0; i < n; i++)**

**printf("%d ", arr[i]);**

**printf("\n");**

**}**

**//0.3. utility function(userdefined function): swap two elements//**

**void swap(int \*a, int \*b)**

**{ int temp = \*a;**

**\*a = \*b;**

**\*b = temp;**

**}**

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

**{ int i,j,hold,pos;**

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

**{ hold = arr[i];**

**pos = i-1;**

**for(j=0;j<i;j++)**

**{**

**if(hold<arr[pos] && pos>=0)**

**{ arr[pos+1]=arr[pos];**

**pos=pos-1;**

**}**

**else break;**

**arr[pos+1]=hold;**

**}**

**}**

**}**

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

**{ int i,j,hold,pos;**

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

**{ hold = arr[i];**

**pos = i-1;**

**for(j=0;j<i;j++)**

**{**

**if(hold>arr[pos] && pos>=0)**

**{ arr[pos+1]=arr[pos];**

**pos=pos-1;**

**}**

**else break;**

**arr[pos+1]=hold;**

**}**

**}**

**}**

**//1.1 Ascending Order by insertion Sort to Sorted array// //zubayer-1017311014//**

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

**{ int compcount=0;**

**int swapcount=0;**

**int i,j,hold,pos;**

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

**{**

**hold = arr[i];**

**pos = i-1;**

**for(j=0;j<i;j++)**

**{ compcount++;**

**if(hold<arr[pos] && pos>=0) // for descending order change to >**

**{ arr[pos+1]=arr[pos];**

**pos=pos-1;**

**swapcount++;**

**}**

**else break;**

**arr[pos+1]=hold;**

**}**

**}**

**printArray(arr, n);**

**printf("Number of comparsions is %d\n", compcount);**

**printf("Number of swaps is %d\n", swapcount);**

**}**

**//1.2 Descending Order by insertion Sort to Sorted array// //zubayer-1017311014//**

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

**{ int compcount=0;**

**int swapcount=0;**

**int i,j,hold,pos;**

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

**{ hold = arr[i];**

**pos = i-1;**

**for(j=0;j<i;j++)**

**{ compcount++;**

**if(hold>arr[pos] && pos>=0)**

**{ arr[pos+1]=arr[pos];**

**pos=pos-1;**

**swapcount++;**

**}**

**else break;**

**arr[pos+1]=hold;**

**}**

**}**

**printArray(arr, n);**

**printf("Number of comparsions is %d\n", compcount);**

**printf("Number of swaps is %d\n", swapcount);**

**}**

**//2.1 Ascending Order by merge Sort to Sorted array// //zubayer-1017311014//**

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

**{ int compcount=0;**

**int swapcount=0;**

**// Merges two subarrays of arr[].First (left) subarray is arr[l..m],Second (right) subarray is arr[m+1..r]**

**void merge(int arr[], int l, int m, int r) // Merge for Ascending order**

**{ int i, j, k;**

**int n1 = m - l + 1;// tamp left arr size**

**int n2 = r - m;// tamp right arr size**

**int L[n1], R[n2];/\* create temp arrays \*/**

**for (i = 0; i < n1; i++) /\* Copy data to temp arrays L[] \*/**

**L[i] = arr[l + i];**

**for (j = 0; j < n2; j++) /\* Copy data to temp arrays R[]\*/**

**R[j] = arr[m + 1+ j];**

**i = 0; // Initial index of first (left) subarray**

**j = 0; // Initial index of second (right) subarray**

**k = l; // Initial index of merged subarray**

**while (i < n1 && j < n2)**

**{ compcount++;**

**if (L[i] <= R[j]) // for descending order change to >=**

**{ arr[k] = L[i];**

**i++;**

**}**

**else**

**{ swapcount++;**

**arr[k] = R[j];**

**j++;**

**}**

**k++;**

**}**

**/\* Copy the remaining elements of L[], if there are any \*/**

**while (i < n1)**

**{ arr[k] = L[i];**

**i++;**

**k++;**

**}**

**/\* Copy the remaining elements of R[], if there are any \*/**

**while (j < n2)**

**{ arr[k] = R[j];**

**j++;**

**k++;**

**}**

**}/\*End of merge()\*/**

**/\* l is for left index and r is right index of the sub-array of arr to be sorted \*/**

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

**{ if (l < r)**

**{ // Same as (l+r)/2, but avoids overflow for**

**// large l and h**

**int m = l+(r-l)/2;**

**// Sort first and second halves**

**mergeSort(arr, l, m);**

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

**merge(arr, l, m, r);**

**}**

**}/\*End of merge\_sort\*/**

**mergeSort(A,0,n-1);**

**printArray(A, n);**

**printf("Number of comparsions is %d\n", compcount);**

**printf("Number of swaps is %d\n", swapcount);**

**}**

**//2.2 Descending Order by merge Sort to Sorted array// //zubayer-1017311014//**

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

**{ int compcount=0;**

**int swapcount=0;**

**// Merges two subarrays of arr[].First (left) subarray is arr[l..m],Second (right) subarray is arr[m+1..r]**

**void merge(int arr[], int l, int m, int r) // Merge for Descending order**

**{ int i, j, k;**

**int n1 = m - l + 1;// tamp left arr size**

**int n2 = r - m;// tamp right arr size**

**int L[n1], R[n2];/\* create temp arrays \*/**

**for (i = 0; i < n1; i++) /\* Copy data to temp arrays L[] \*/**

**L[i] = arr[l + i];**

**for (j = 0; j < n2; j++) /\* Copy data to temp arrays R[]\*/**

**R[j] = arr[m + 1+ j];**

**i = 0; // Initial index of first (left) subarray**

**j = 0; // Initial index of second (right) subarray**

**k = l; // Initial index of merged subarray**

**while (i < n1 && j < n2)**

**{ compcount++;**

**if (L[i] >= R[j]) // for ascending order change to <=**

**{ arr[k] = L[i];**

**i++;**

**}**

**else**

**{ swapcount++;**

**arr[k] = R[j];**

**j++;**

**}**

**k++;**

**}**

**/\* Copy the remaining elements of L[], if there are any \*/**

**while (i < n1)**

**{ arr[k] = L[i];**

**i++;**

**k++;**

**}**

**/\* Copy the remaining elements of R[], if there are any \*/**

**while (j < n2)**

**{ arr[k] = R[j];**

**j++;**

**k++;**

**}**

**}/\*End of merge()\*/**

**/\* l is for left index and r is right index of the sub-array of arr to be sorted \*/**

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

**{ if (l < r)**

**{ // Same as (l+r)/2, but avoids overflow for**

**// large l and h**

**int m = l+(r-l)/2;**

**// Sort first and second halves**

**mergeSort(arr, l, m);**

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

**merge(arr, l, m, r);**

**}**

**}/\*End of merge\_sort\*/**

**mergeSort(A,0,n-1);**

**printArray(A, n);**

**printf("Number of comparsions is %d\n", compcount);**

**printf("Number of swaps is %d\n", swapcount);**

**}**

**//3.1 Ascending Order by Quick Sort to Sorted array// //zubayer-1017311014//**

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

**{ int compcount=0;**

**int swapcount=0;**

**/\* 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**

**int j;**

**for ( j = low; j <= high- 1; j++)**

**{ compcount++;**

**// If current element is smaller than or**

**// equal to pivot**

**if (arr[j] <= pivot) // for descending order change <= to >=**

**{**

**i++; // increment index of smaller element**

**swap(&arr[i], &arr[j]);**

**if (i<j){swapcount++;}**

**}**

**}**

**swap(&arr[i + 1], &arr[high]);// no comparsion only one swap**

**if (i+1<high){swapcount++;}**

**return (i + 1);**

**}/\*End of partition()\*/**

**/\* 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);**

**quickSort(arr, low, pi - 1); // Separately sort elements before pi**

**quickSort(arr, pi + 1, high); // partition and after partition**

**}**

**}/\*End of quickSort()\*/**

**quickSort(A,0,n-1);**

**printArray(A, n);**

**printf("Number of comparsions is %d\n", compcount);**

**printf("Number of swaps is %d\n", swapcount);**

**}**

**////3.2 Descending Order by Quick Sort to Sorted array// //zubayer-1017311014//**

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

**{ int compcount=0;**

**int swapcount=0;**

**/\* 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**

**int j;**

**for ( j = low; j <= high- 1; j++)**

**{ compcount++;**

**// If current element is smaller than or**

**// equal to pivot**

**if (arr[j] >= pivot) // for ascending order change >= to <=**

**{**

**i++; // increment index of smaller element**

**swap(&arr[i], &arr[j]);**

**if (i<j){swapcount++;}**

**}**

**}**

**swap(&arr[i + 1], &arr[high]);// no comparsion only one swap**

**if (i+1<high){swapcount++;}**

**return (i + 1);**

**}/\*End of partition()\*/**

**/\* 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);**

**quickSort(arr, low, pi - 1); // Separately sort elements before pi**

**quickSort(arr, pi + 1, high); // partition and after partition**

**}**

**}/\*End of quickSort()\*/**

**quickSort(A,0,n-1);**

**printArray(A, n);**

**printf("Number of comparsions is %d\n", compcount);**

**printf("Number of swaps is %d\n", swapcount);**

**}**

**//4. Array In Random Order// //zubayer-1017311014//**

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

**{ int compcount=0;**

**int swapcount=0;**

**int i;**

**// Use a different seed value so that we don't get same**

**// result each time we run this program**

**srand( time(NULL) );**

**// Start from the last element and swap one by one. We don't**

**// need to run for the first element that's why i > 0**

**for ( i = n-1; i > 0; i--)**

**{ compcount++;**

**// Pick a random index from 0 to i**

**int j = rand() % (i+1);**

**// Swap arr[i] with the element at random index**

**swap(&arr[i], &arr[j]);**

**swapcount++;**

**}**

**printArray(A, n);**

**printf("Number of comparsions is %d\n", compcount);**

**printf("Number of swaps is %d\n", swapcount);**

**}**

**int main()**

**{ int choice;**

**printf("Given Unsorted array is \n");**

**printArray(array, n);**

**while(1)**

**{ printf("\n====================================");**

**printf("\n1. Show Result of the Insertion Sort");**

**printf("\n2. Show Result of the Merge Sort");**

**printf("\n3. Show Result of the Quick Sort");**

**printf("\n4. Show Result of the each sort in both order from given");**

**printf("\n5. Quit");**

**printf("\nEnter Your Option:\t");**

**scanf("%d", &choice);**

**switch(choice)**

**{**

**case 1: arrayCopy(A,n);**

**AscSort(A, n);**

**printf("\n1.1 Insertion Sorted array From (101-200) Ascending to Ascending Order:\n");**

**insertionSortAsc(A, n);**

**arrayCopy(A,n);**

**DscSort(A, n);**

**printf("\n1.2 Insertion Sorted array From (200-101) Descending to Ascending Order:\n");**

**insertionSortAsc(A, n);**

**arrayCopy(A,n);**

**printf("\n1.3 Insertion Sorted array From (101-200) following Random to Ascending Order:\n");**

**insertionSortAsc(A, n);**

**break;**

**case 2: arrayCopy(A,n);**

**AscSort(A, n);**

**printf("\n2.1 Merge Sorted array From (101-200) Ascending to Ascending Order:\n");**

**mergeSortAsc(A, n);**

**arrayCopy(A,n);**

**DscSort(A, n);**

**printf("\n2.2 Merge Sorted array From (200-101) Descending to Ascending Order:\n");**

**mergeSortAsc(A, n);**

**arrayCopy(A,n);**

**printf("\n2.3 Merge Sorted array From (101-200) following Random to Ascending Order:\n");**

**mergeSortAsc(A, n);**

**break;**

**case 3: arrayCopy(A,n);**

**AscSort(A, n);**

**printf("\n3.1 Quick Sorted array From (101-200) Ascending to Ascending Order:\n");**

**quickSortAsc(A,n);**

**arrayCopy(A,n);**

**DscSort(A, n);**

**printf("\n3.2 Quick Sorted array From (200-101) Descending to Ascending Order:\n");**

**quickSortAsc(A,n);**

**arrayCopy(A,n);**

**printf("\n3.3 Quick Sorted array From (101-200) following Random to Ascending Order:\n");**

**quickSortAsc(A,n);**

**break;**

**case 4: printf("\nInsertion Sorted array In Ascending Order:\n");**

**arrayCopy(A,n);**

**insertionSortAsc(A, n);**

**printf("\nInsertion Sorted array In Descending Order:\n");**

**arrayCopy(A,n);**

**insertionSortDsc(A, n);**

**printf("\nMerge Sorted array In Ascending Order:\n");**

**arrayCopy(A,n);**

**mergeSortAsc(A,n);**

**printf("\nMerge Sorted array In Descending Order:\n");**

**arrayCopy(A,n);**

**mergeSortDsc(A, n);**

**printf("\nQuick Sorted array In Ascending Order:\n");**

**arrayCopy(A,n);**

**quickSortAsc(A,n);**

**printf("\nQuick Sorted array In Descending Order:\n");**

**arrayCopy(A,n);**

**quickSortDsc(A,n);**

**printf("\nArray In Random Order:\n");**

**arrayCopy(A,n);**

**randomize (A, n);**

**break;**

**case 5: exit(0);**

**default: printf("Enter a correct input\n");**

**}**

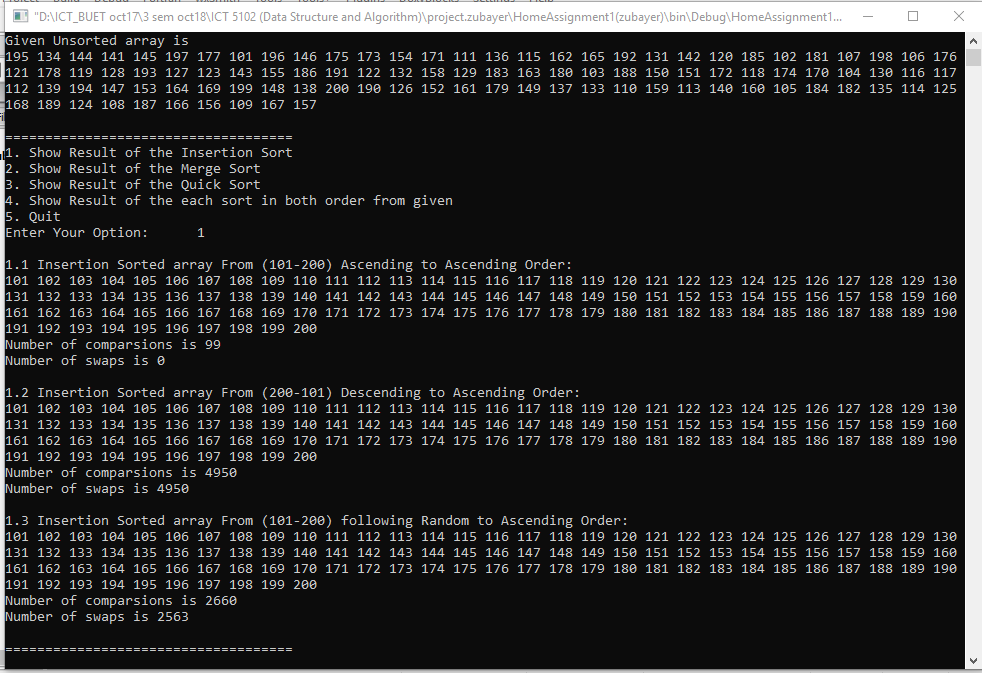
**}**

**return 0;**

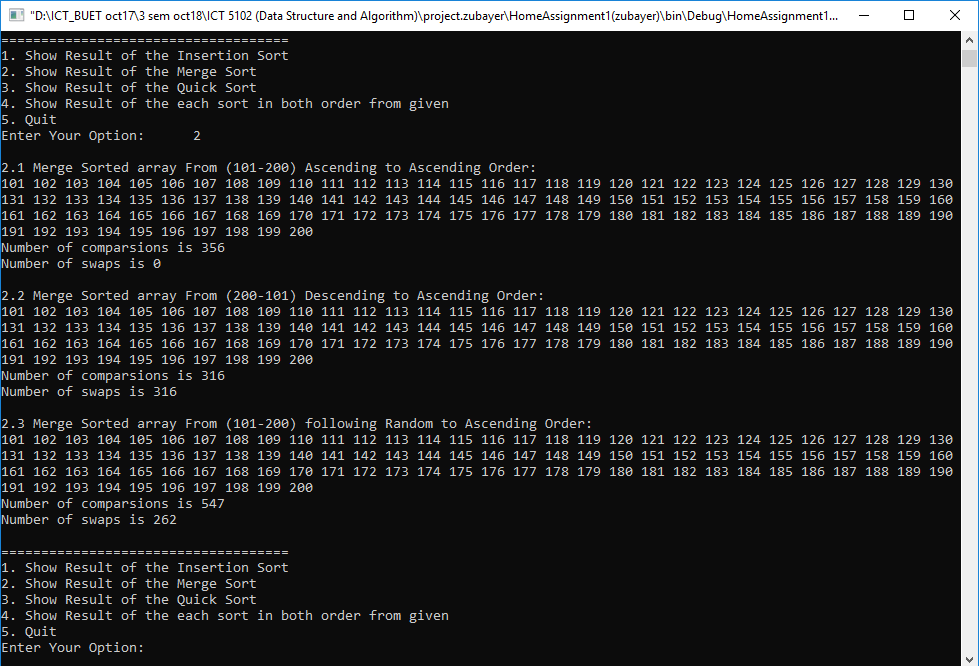
**}**

# OUTPUT: For Combined 3 types of sorts in a program, I use “switch case” in the program to show result each sort separately. I have used 5 options by “switch case” operation.

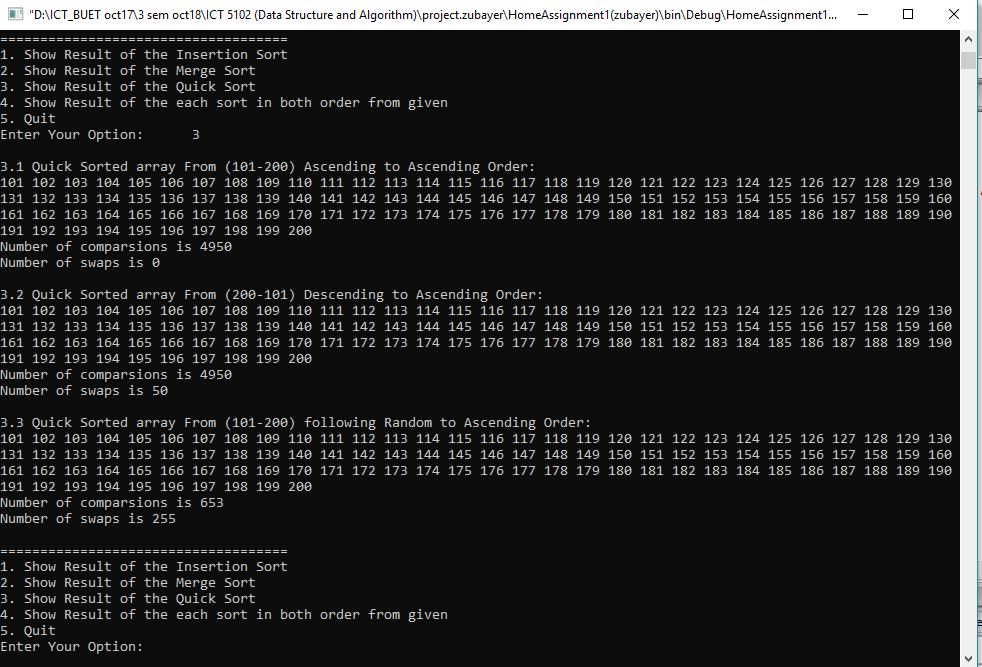
Option 1 press: show result from Ascending, Descending, random to Insertion ascending sort :



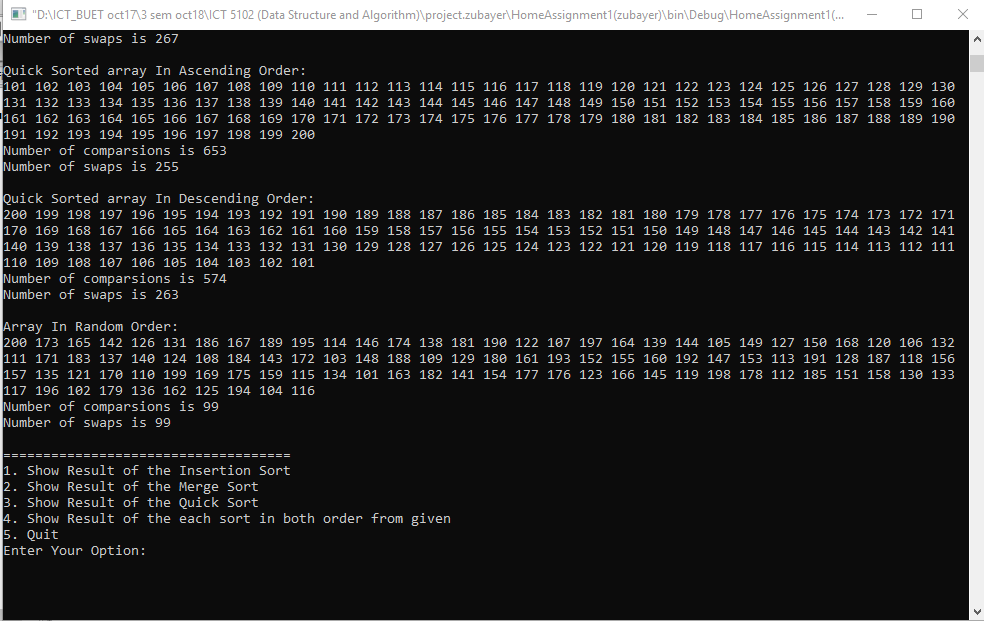
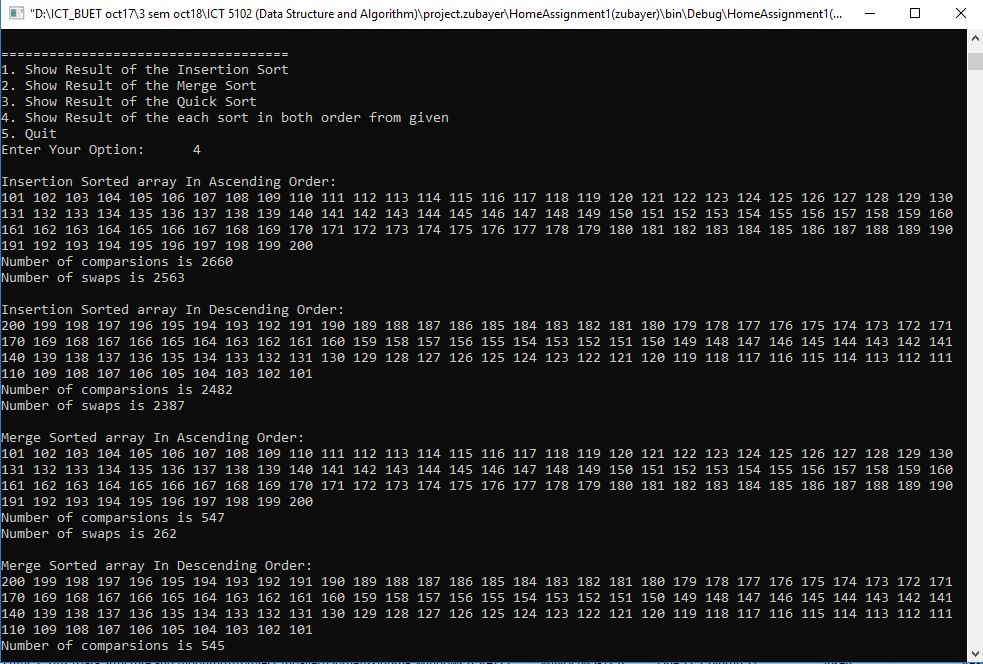
Option 2 press: show result from Ascending, Descending, random to Merge ascending sort :



Option 3 press: show result from Ascending, Descending, random to Quick ascending sort :



Option 4 press:show result from random to Ascending, Descending of Insertion, Merge, Quick sort :



Option 4 press: to Quit program.

# OUTPUT DATA REPORT FOR COMPARISONS & SWAPS COUNT:

|  |  |  |  |
| --- | --- | --- | --- |
| ALGORITHM | IN Order | NO OF COMPARISONS | NO OF SWAPS |
| **Insertion Sort** | From (101-200) Ascending to Ascending Order | 99 | 0 |
| From (200-101) Descending to Ascending Order | 4950 | 4950 |
| From (101-200) following Random to Ascending Order | 2660 | 2563 |
| **Merge Sort** | From (101-200) Ascending to Ascending Order | 356 | 0 |
| From (200-101) Descending to Ascending Order | 316 | 316 |
| From (101-200) following Random to Ascending Order | 547 | 262 |
| **Quick Sort** | From (101-200) Ascending to Ascending Order | 4950 | 0 |
| From (200-101) Descending to Ascending Order | 4950 | 50 |
| From (101-200) following Random to Ascending Order | 653 | 255 |

Additional Result: option 4 of switch case

|  |  |  |  |
| --- | --- | --- | --- |
| ALGORITHM | IN Order | NO OF COMPARISONS | NO OF SWAPS |
| **Insertion Sort** | From (101-200) following Random to Ascending Order | 2660 | 2563 |
| From (101-200) following Random to Descending Order | 2482 | 2387 |
|  | | |
| **Merge Sort** | From (101-200) following Random to Ascending Order | 547 | 262 |
| From (200-101) Descending to Ascending Order | 545 | 267 |
|
| **Quick Sort** | From (101-200) following Random to Ascending Order | 653 | 255 |
| From (101-200) following Random to Descending Order | 574 | 263 |
|  | | |

#CONCLUSION:

I have used C programming language for that assignment completion. This is given the ultimate result for Insertion Sort, Merge Sort & Quick Sort using their algorithm. And comparison both sort with each other from these 3 orders to ascending order. Show additional option where show count of comparison and swap from Random to Ascending and descending Order. It’s able to upgrade another option as from these 3 orders to ascending order.

From these sorts Algorithm, we also know comparison of their time complexity and take better option.

#REFERENCES:

[1] Class Lectures of “Dr. Hossen A Mustafa”

[2] https:// www.geeksforgeeks.org

[3] https://tutorialpoint.com

[4] youtube