**Homework Assignment 1**

**ICT:5102, Data Structure & Algorithm**



**Md.Dedarul Hasan**

**PGD in IT, Registration No: 0417311011**

**Session: April, 2017**

**IICT-BUET**

# **INTRODUCTION OF TASK TO DO :**

Insertion Sort, Merge Sort & Quick Sort Algorithms.Total Comparison & Swap Count During Sort Proces. Array Sorting By Ascending, Descending & Random Order Implementations.

# **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 Blocks IDE Tools
2. GDB/CDB Debugger Tools
3. IntelliJ IDE /Netbeans IDE
4. JDK
5. Sublime Text Editor

# **PROCEDURE :**

1. C programming
2. C++ programming
3. Java programming

# 

# 

# 

# **PROGRAM CODE IN C :**

We have implemented one C class file with user defined method to make that insertion ,merge & quick sorting algorithm.Now ,we will see bellow with that codes:

#include<stdio.h>

//GIVEN INPUT ARRAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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]);//LENGTH OF GIVEN ARRAY

int A[]; // TEMP ARRAY TO STORE THE COPIED FORM GIVEN INPUT ARRAY

//METHOD TO COPY array BY int A[]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int loop;

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

copied[loop] = array[loop];

}

}

//METHOD TO PRINT ARRAY ALWAYS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int i;

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

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

printf("\n");

}

//METHOD TO MAKE SWAP WITH ONE ELEMENT TO ANOTHER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

//METHOD TO MAKE ARRAY IN ASCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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;

}

}

}

//METHOD TO MAKE ARRAY IN DESCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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;

}

}

}

//METHOD TO MAKE ARRAY SORT USING INSERTION SORT WITH ASCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int comparisonCount=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++){

comparisonCount++;

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("TOTAL NUMBER OF COMPARISONS: %d\n", comparisonCount);

printf("TOTAL NUMBER OF SWAPS: %d\n", swapCount);

}

//METHOD TO MAKE ARRAY SORT USING INSERTION SORT WITH DESCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int comparisonCount=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++){

comparisonCount++;

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("TOTAL NUMBER OF COMPARISONS: %d\n", comparisonCount);

printf("TOTAL NUMBER OF SWAPS: %d\n", swapCount);

}

//METHOD TO MAKE ARRAY SORT USING MERGE SORT WITH ASCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int comparisonCount=0;

int swapCount=0;

//METHOD TO MAKE ARRAY SORT USING MERGE SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int i, j, k;

int n1 = m - l + 1;// TEMP LEFT ARRAY SIZE

int n2 = r - m;// TEMP RIGHT ARRAY 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;

j = 0;

k = l;

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

comparisonCount++;

if (L[i] <= R[j]){

arr[k] = L[i];

i++;

}

else{

swapCount++;

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1){

arr[k] = L[i];

i++;

k++;

}

while (j < n2){

arr[k] = R[j];

j++;

k++;

}

}

//MERGE SORT METHOD TO IMPLEMENT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

if (l < r){

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

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

mergeSort(A,0,n-1);

printArray(A, n);

printf("TOTAL NUMBER OF COMPARISONS: %d\n", comparisonCount);

printf("TOTAL NUMBER OF SWAPS: %d\n", swapCount);

}

//MARGE SORT DESCENDING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int comparisonCount=0;

int swapCount=0;

// MERGE TWO SUBARRAY OF arr[].First (left) SUB ARRAY IS arr[l..m],SECOND (right) SUBARRAY IS arr[m+1..r] \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int i, j, k;

int n1 = m - l + 1;// TEMP LEFT ARR SIZE

int n2 = r - m;// TEMP 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;

j = 0;

k = l;

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

comparisonCount++;

if (L[i] >= R[j]){

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++;

}

}

//MERGE SORT ARRAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

if (l < r){

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

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

mergeSort(A,0,n-1);

printArray(A, n);

printf("TOTAL NUMBER OF COMPARISONS: %d\n", comparisonCount);

printf("TOTAL NUMBER OF SWAPS: %d\n", swapCount);

}

//METHOD FOR QUICK SORT IN ASCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int comparisonCount=0;

int swapCount=0;

//METHOD FOR PARTITIONING \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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++){

comparisonCount++;

if (arr[j] <= pivot){

i++; // INCREMENT INDEX OF SMALLER ELEMENT

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

if (i<j){

swapCount++;

}

}

}

swap(&arr[i + 1], &arr[high]);// NO COMPARISON ONLY ONE SWAP

if (i+1<high){

swapCount++;

}

return (i + 1);

}/\*END OF PARTITION()\*/

//QUICK SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void quickSort(int arr[], int low, int high){

if (low < high){

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

quickSort(A,0,n-1);

printArray(A, n);

printf("TOTAL NUMBER OF COMPARISONS: %d\n", comparisonCount);

printf("TOTAL NUMBER OF SWAPS: %d\n", swapCount);

}

//METHOD FOR QUICK SORT DESCENDING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void quickSortDsc(int arr[],int n)

{ int comparisonCount=0;

int swapCount=0;

//PARTITIOON MAKING\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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++){

comparisonCount++;

if (arr[j] >= pivot){

i++; // INCREMENT INDEX OF SMALLER ELEMENT

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

if (i<j){

swapCount++;

}

}

}

swap(&arr[i + 1], &arr[high]);// NO COMPARISON ONLY ONE SWAP

if (i+1<high){

swapCount++;

}

return (i + 1);

}/\*END OF partition()\*/

//METHOD FOR QUICK SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

void quickSort(int arr[], int low, int high){

if (low < high){

int pi = partition(arr, low, high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}/\*END OF quickSort()\*/

quickSort(A,0,n-1);

printArray(A, n);

printf("TOTAL NUMBER OF COMPARISONS: %d\n", comparisonCount);

printf("TOTAL NUMBER OF SWAPS: %d\n", swapCount);

}

//METHOD FOR RANDOMIZE THE ARRAY ELEMENTS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int comparisonCount=0;

int swapCount=0;

int i;

srand( time(NULL) );

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

comparisonCount++;

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

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

swapCount++;

}

printArray(A, n);

printf("TOTAL NUMBER OF COMPARISONS: %d\n", comparisonCount);

printf("TOTAL NUMBER OF SWAPS: %d\n", swapCount);

}

int main(){

//PROGRAMMER INFORMATION INDETAILS-\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

const char assignmentName[200] = " \*\*\* Home Work Assignment \*\*\*";

const char programmerName[200] = " MD.DEDARUL HASAN,IICT-BUET";

const char registrationID[200] = " REG. NO: 0417311011";

const char sessionID[200] = " SESSION: APRIL,2017";

const char submissionDate[200] = " COMPLETED: 22/02/2019";

printf("\n %s\n", assignmentName );

printf(" %s\n", programmerName );

printf(" %s\n", registrationID );

printf(" %s\n", sessionID );

printf(" %s\n", submissionDate );

printf("\n");

//MAIN METHOD UTILIZATION CODE STARTED HERE\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int chooseOption;

printf("GIVEN INPUT ARRAY \n");

printf("int A[]=\n{");

printArray(array, n);

printf("}\n");

while(1){

printf("TASKS TO DO:");

printf("\n1. OPTION FOR INSERTION SORT");

printf("\n2. OPTION FOR MARGE SORT");

printf("\n3. OPTION FOR QUICK SORT");

printf("\n4. OPTION FOR EACH SORT IN BOTH ORDER");

printf("\n5. EXIT\n");

printf("\nENTER OPTION TO VIEW:\t");

scanf("%d", &chooseOption);

//

switch(chooseOption){

case 1: arrayCopy(A,n);

AscSort(A, n);

printf("\n1.1 INSERTION SORTED ARRAY [101-200] ASCENDING TO ASCENDING ORDER:\n");

insertionSortAsc(A, n);

arrayCopy(A,n);

DscSort(A, n);

printf("\n1.2 INSERTION SORTED ARRAY [200-101] DESCENDING TO ASCENDING ORDER:\n");

insertionSortAsc(A, n);

arrayCopy(A,n);

printf("\n1.3 INSERTION SORTED ARRAY [101-200] RANDOM TO ASCENDING ORDER:\n");

insertionSortAsc(A, n);

break;

case 2: arrayCopy(A,n);

AscSort(A, n);

printf("\n2.1 MERGE SORTED ARRAY[101-200] ASCENDING TO ASCENDING ORDER:\n");

mergeSortAsc(A, n);

arrayCopy(A,n);

DscSort(A, n);

printf("\n2.2 MERGE SORTED ARRAY [200-101] DESCENDING TO ASCENDING ORDER:\n");

mergeSortAsc(A, n);

arrayCopy(A,n);

printf("\n2.3 MERGE SORTED ARRAY [101-200] RANDOM TO ASCENDING ORDER:\n");

mergeSortAsc(A, n);

break;

case 3: arrayCopy(A,n);

AscSort(A, n);

printf("\n3.1 QUICK SORTED ARRAY [101-200] ASCENDING TO ASCENDING ORDER:\n");

quickSortAsc(A,n);

arrayCopy(A,n);

DscSort(A, n);

printf("\n3.2 QUICK SORTED ARRAY [200-101] DESCENDING TO ASCENDING ORDER:\n");

quickSortAsc(A,n);

arrayCopy(A,n);

printf("\n3.3 QUICK SORTED ARRAY [101-200] 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("PLEASE ENTER CORRECT INPUT!\n");

}

}

return 0;

}

# 

# 

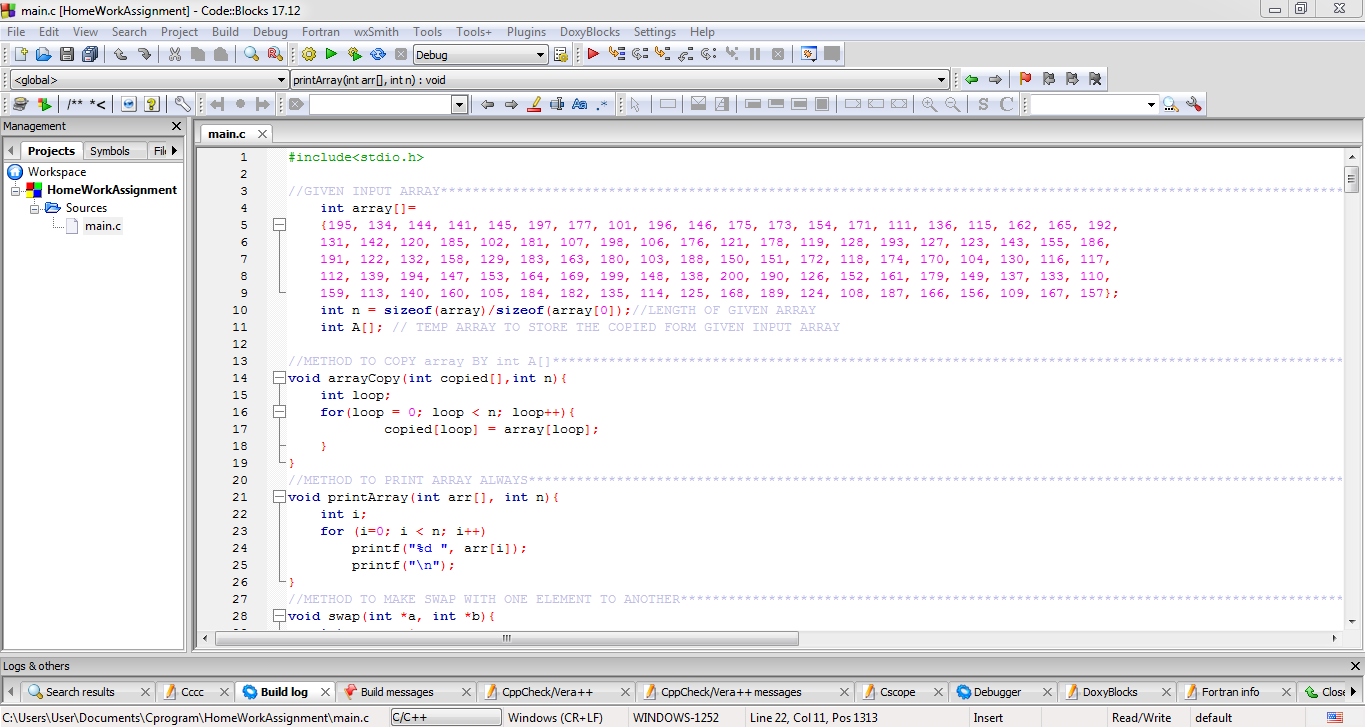
# 

# 

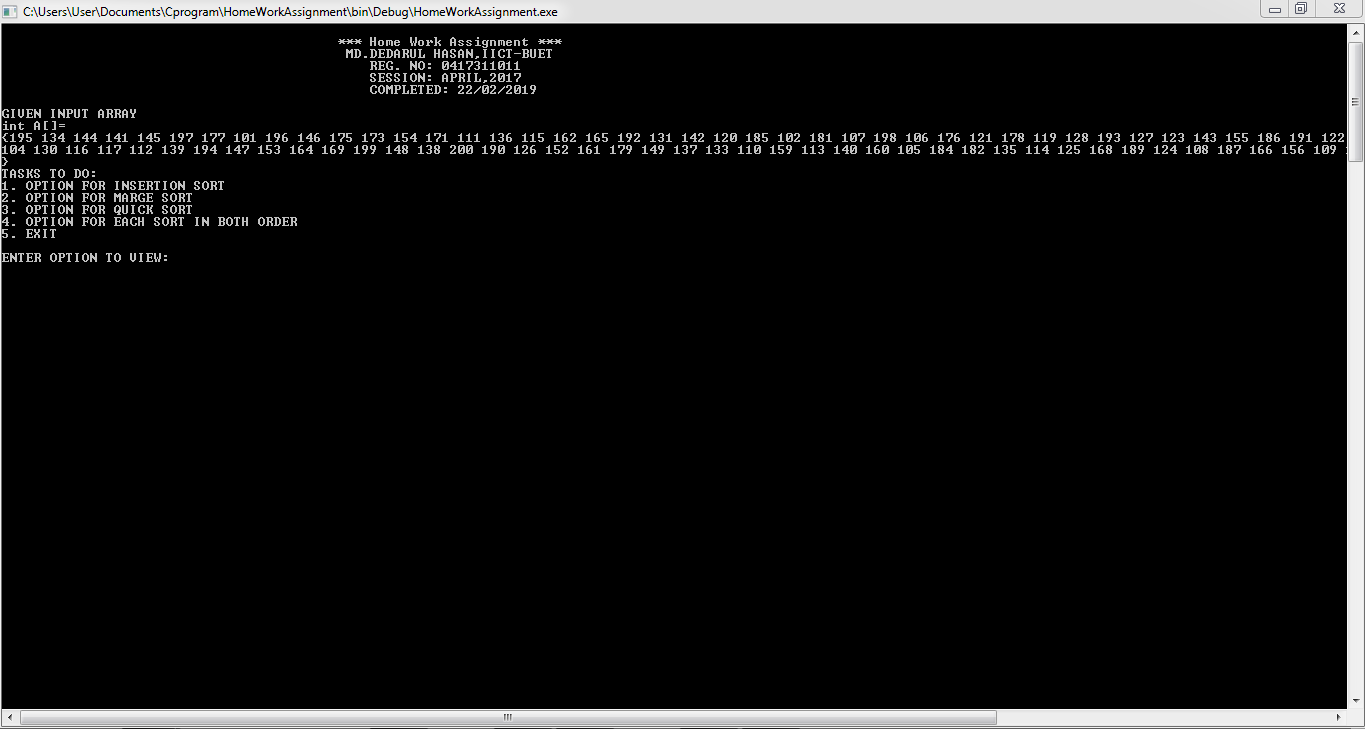
# 

# **PROGRAM OUTPUT FOR THAT ALGORITHMS :**

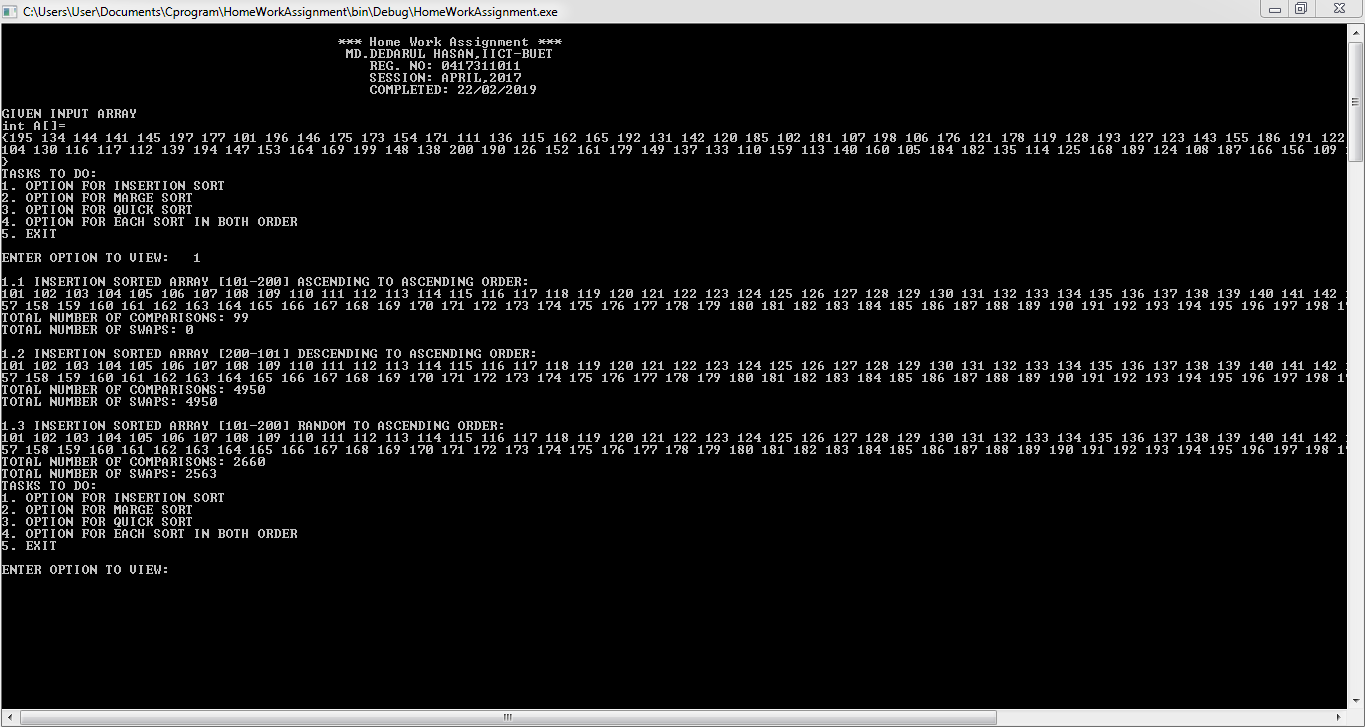
**Fig :1-Code Block Project Overview**



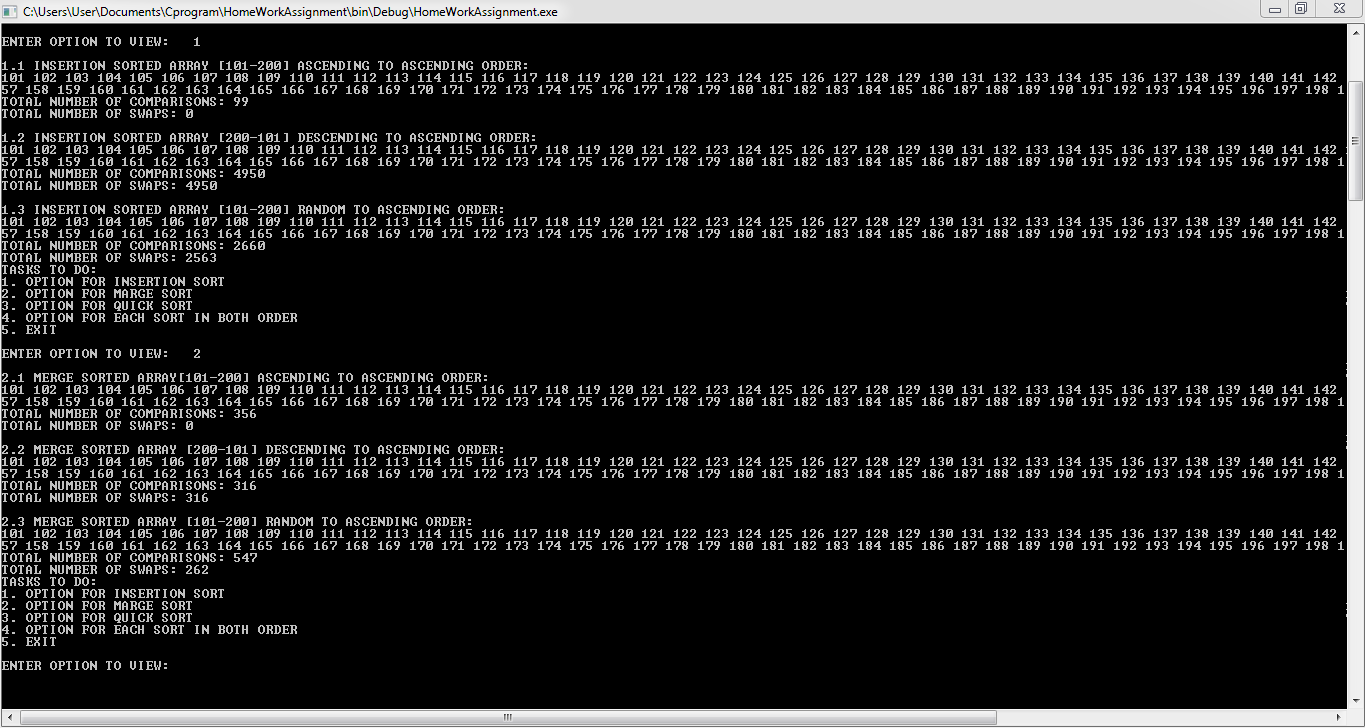
**Fig :2- Output of Assignment Class File Execution- Starting Phase**



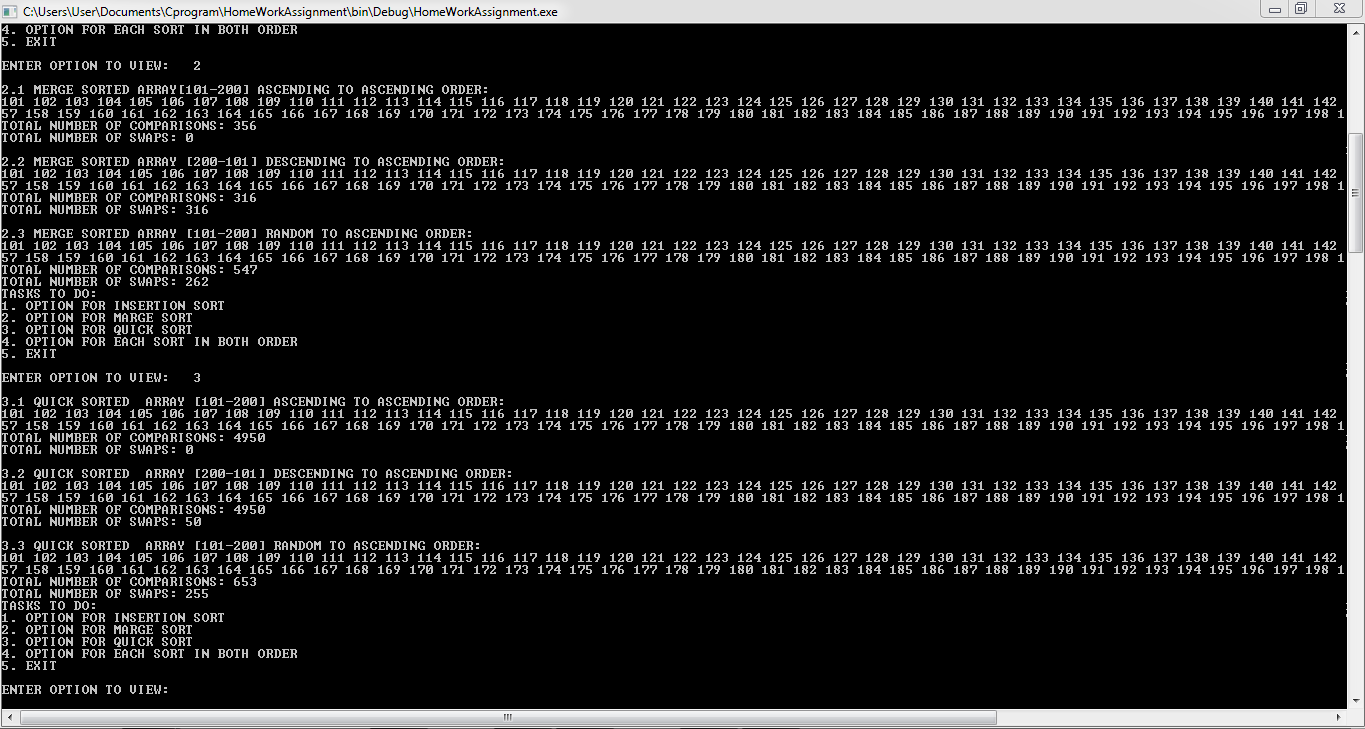
**Fig :3- Entered Option 1 For Insertion Sort With Total Comparisons & Swap Count**



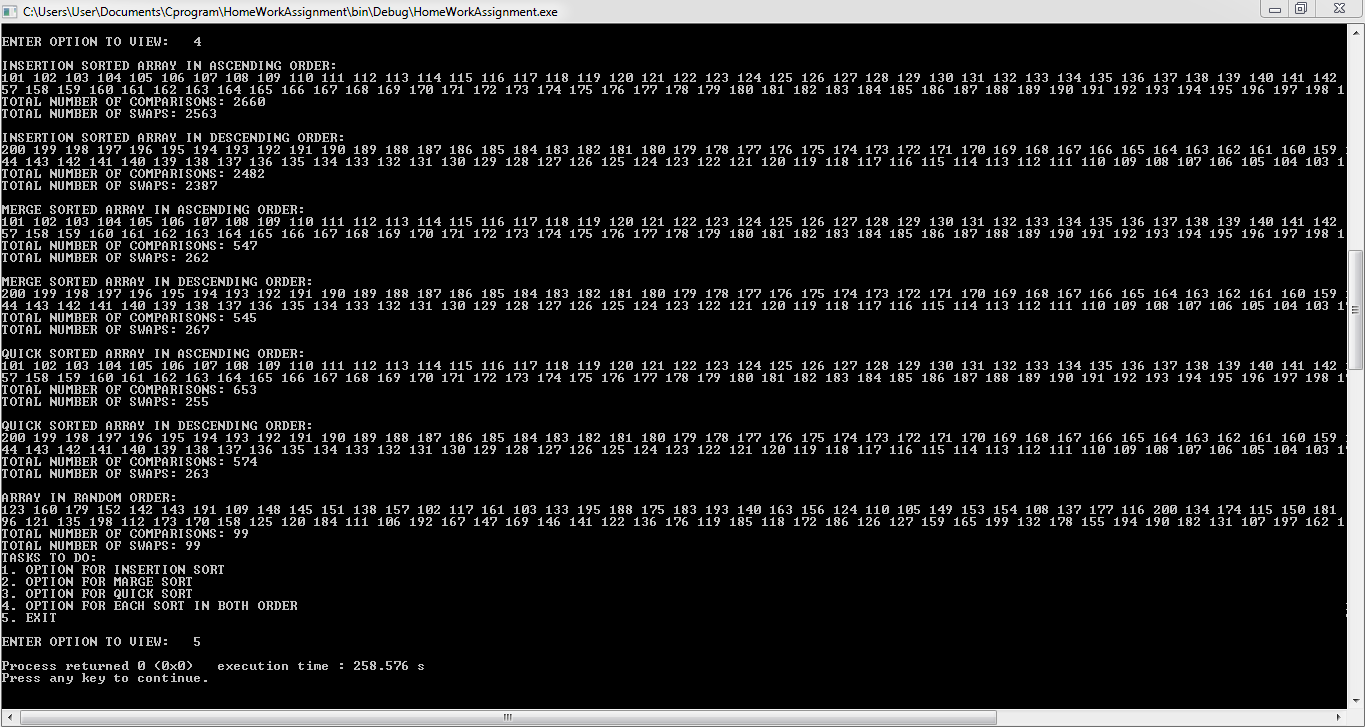
**Fig :4- Entered Option 2 For Merge Sort With Total Comparisons & Swap Count**



**Fig :5- Entered Option 3 For Quick Sort With Total Comparisons & Swap Count**



**Fig :7- Entered Option 4 For Random Sort With Total Comparisons & Swap Count**



# **OUTPUT DATA REPORT FOR TOTAL COMPARISONS & SWAPS COUNT :**

|  |  |  |
| --- | --- | --- |
| **ALGORITHM** | **NO OF COMPARISONS** | **NO OF SWAPS** |
| **Insertion Sort** | **2660** | **2563** |
| **Merge Sort** | **547** | **262** |
| **Quick Sort** | **653** | **255** |

# 

# **RESULTS :**

We have used C programming language for that assignment completions. Which has given the ultimate result for Insertion Sort, Merge Sort & Quick Sort using their algorithm & pseudo codes. After that we have sorted the given input array [101-200] in Ascending ,Descending & also Randomize options too.

# **CONCLUSION :**

1. **Insert sort** is more efficient than bubble sort and selection sort.In this algo we divide the entire array into parts; the sorted array and the unsorted array.With every iteration we have to place the first element of the unsorted array in the correct position of the sorted array and increase the sorted list by one.

**Time Complexity:**

when elements are sorted there are no swaps and the correct position of the element in the sorted list is the current index itself.The time complexity is : **O(n**)

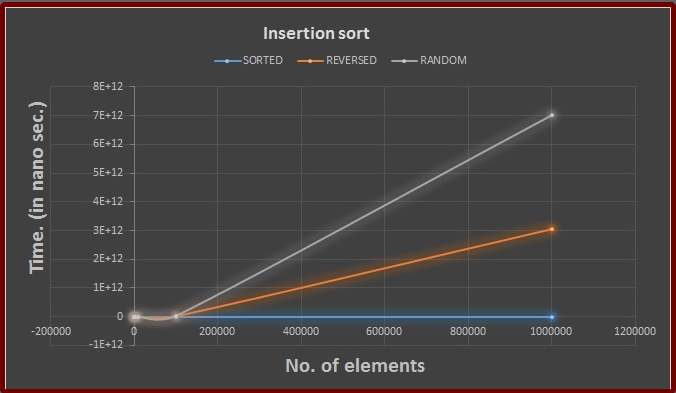
Insertion sort gets penalized if comparison or copying is slow. In other words, the maximum array size which is faster to sort with insertion sort compared to O(*n*log*n*) algorithms gets smaller if comparison or copying of the array elements is slow.

**Divide and Conquer Approach-**

In this approach the algorithm is divide into multiple sub-problems.Each of these sub-problems is then solved separately and the solution of each sub-problem is used to solve the original problem.

Divide and conquer technique uses recursion to solve each of the sub-problem.

**Fig : Insertion Sort Time Complexity**



1. **Merge sort** uses the divide and conquer approach.It is one of the most efficient algo for sorting.In this algo,we divide the list into two from the middle and keep dividing the list until the sub-problems has only one element list.

We then merge the list and while merging the list we sort them in the ascending order.

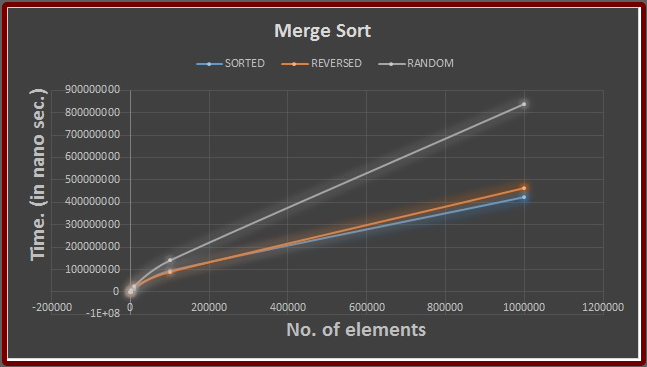
**Time complexity:**

We are dividing the list into no matter if the list is sorted or no.But if the array is sorted,while merging the list there are no swaps merging results into an array itself.Thus, the best ,average and worst case time complexity is: O(nlogn)

Surprisingly fast, at least with the optimizations used in this test (ie. the sorting function doesn't need to allocate the secondary array each time it is called). Given that it is always O(*n*log*n*), it is a very good alternative if the extra memory requirement is not a problem.

Array elements with fast comparisons and slow copying seem to slightly penalize merge sort.

**Fig: Merge Sort Time Complexity**



1. **Quick sort** uses the similar approach of divide and conquer technique

In this technique, element is selected which is the pivot element.Now the element which are less than the pivot are placed to the left of the array and the element which are more than the pivot are placed to the right of the pivot.The index of the pivot element is then returned back to the function.The same function is called to the sub-array left to the pivot which has all the elements less than the pivot and also to the right of the sub-array which has the elements more than the pivot.

After calling the function recursively,the resulting function will be a sorted array.

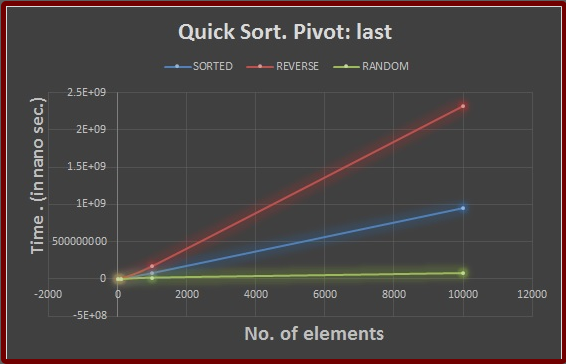
**Time complexity:**

The **best case** is when the elements are in a sorted manner. The best and average case time complexity is : **O(nlogn**)

The **worst case** time complexity is when the elements are in a reverse sorted manner.The time complexity is :**O(n2)**

In this Quick Sort,the last element in the list is taken as the pivot element.This Quick Sort is a bit slow as compared to other approaches of Quick Sort.

**Fig: Quick Sort Time Complexity**



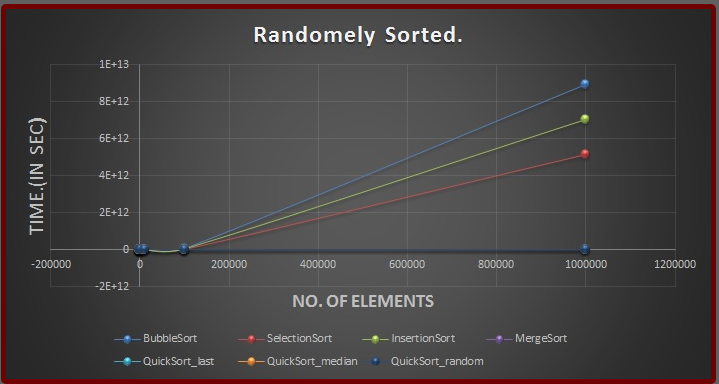
1. For **Random sort** approach,Most of the times this is going to be the approach since we are going to sort random number elements

Other sort,should never be employed.They take hours to sort a million elements.We can see that by the readings provided in the zip file of the assignment 5 folder.

Merge sort performs very well for this sort.It should be employed.

Also the quick sort algorithm performs very well for a million elements.All three approaches of quick sort are pretty fast and any of them can be employed.However,if one requires very good efficiency,he should use Quicksort pivot-median approach.

**Fig: Random Sort Time Complexity**



# **REFERENCES**

1. ***Class Lectures & Pseudo-Code***
2. ***Stack Overflow***
3. ***GeeksforGeeks***
4. ***Others Blog Sites***
5. ***Books Help***

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* THANKS \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***