**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}

# 

# 

# 

# 

# 

# 

# 

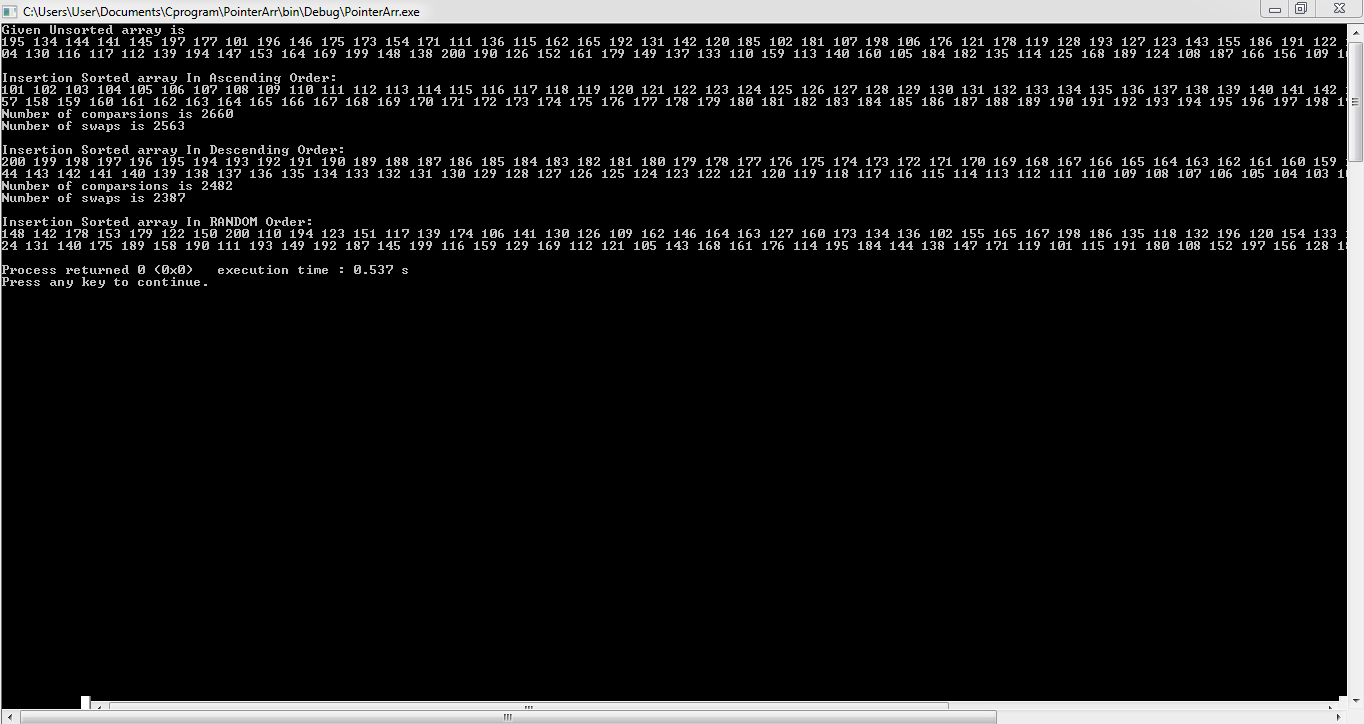
# 

# **CODE FOR INSERTION SORT WITH COMPARISON & SWAP :**

#include <stdio.h>  
 int array[100] =  
 {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 array[]={29, 10, 14, 37, 13};  
 int \* pointer=array; //WE HAVE STORED THAT ARRAY IN APOINTER  
 int n = sizeof(array)/sizeof(array[0]);  
 int A[];  
//METHOD TO COPY IN ARRAY A[] FROM array[]\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
void arrayCopy(int copied[],int n){  
 int loop;  
 for(loop = 0; loop < n; loop++) {  
 copied[loop] = array[loop];  
 }  
 /\* printf("A -> copied \n");  
  
 for(loop = 0; loop < n; loop++) {  
 printf(" %2d %2d\n", array[loop], copied[loop]);  
 }\*/  
}  
//METHOD TO SWAP TWO ELEMENT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
void swap (int \*a, int \*b){  
 int temp = \*a;  
 \*a = \*b;  
 \*b = temp;  
 }  
//METHOD OF INSERTION SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
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 COMPARISON IS %d\n", comparisonCount);  
 printf("TOTAL NUMBER OF SWAP IS %d\n", swapCount);  
}  
//METHOD OF INSERTION SORT FOR DESCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
void insertionSortDesc(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 COMPARISON IS :%d\n", comparisonCount);  
 printf("TOTAL NUMBER OF SWAP IS :%d\n", swapCount);  
}  
//METHOD TO PRINT ARRAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
void printArray(int arr[], int n)  
{  
 int i;  
 for (i=0; i < n; i++)  
 printf("%d ", arr[i]);  
 printf("\n");  
}  
// A FUNCTION TO GENERATE A RANDOM NUMBERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
void randomize ( int arr[], int n ){  
 srand ( time(NULL) );  
 for (int i = n-1; i > 0; i--){  
 // PIC A RANDOM INDEX FROM 0 TO i  
 int j = rand() % (i+1);  
 // SWAP arr[i] WITH THE ELEMENT AT RANDOM INDEXS  
 swap(&arr[i], &arr[j]);  
 }  
 }  
//MAIN METHOD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
int main(){  
 printf("GIVEN UNSORTED ARRAY IS :\n");  
 printArray(array, n);  
//PRINT ARRAY INSERTION SORT ASCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
 printf("\nINSERTION SORTED ARRAY IN ASCENDING ORDER IS :\n");  
 arrayCopy(A,n);  
 insertionSortAsc(A, n);  
 //free(A);  
//PRINT INSERTION SORT DESCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//  
 printf("\nINSERTION SORTED ARRAY IN DESCENDING ORDER IS :\n");  
 //arrayCopy(A,n);  
 insertionSortDesc(pointer, n);  
 printf("\nINSERTION SORTED ARRAY IN RANDOM ORDER IS :\n");  
 randomize (pointer, n);  
 printArray(pointer, n);  
  
 return 0;  
}

# **OUTPUT :**

**Fig :1- Insertion sort in ascending**



# 

# **CODE FOR MERGE SORT WITH COMPARISON & SWAP :**

#include<stdlib.h>

#include<stdio.h>

//MARGE SORT PROGRAM

int comparsioncounter=0;

int swapcounter=0;

int swapcounter1=0;

//METHOD FOR MERGE SORT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

//CREATE TEMP ARRAY

int L[n1], R[n2];

/\* COPY DATA TO TEMP ARRAYS L[] and R[] \*/

for (i = 0; i < n1; i++)

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

for (j = 0; j < n2; j++)

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

/\* MARGE THE TEMP ARRAYS BACK INTO ARRAY [l...r]\*/

i = 0;

j = 0;

k = l;

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

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

arr[k] = L[i];

i++;

}

else{

arr[k] = R[j];

swapcounter++;

j++;

}

comparsioncounter++;

k++;

}

/\* COPY THE REMAINING ELEMENTS OF L[],IF THERE ARE ANY \*/

while (i < n1){

arr[k] = L[i];

i++;

k++;

//count3++;

}

/\* COPY THE REMAINING ELEMENTS OF R[],ANY IF THERE ANY \*/

while (j < n2){

arr[k] = R[j];

j++;

k++;

}

}

/\* l IS FOR LEFT INDEX and r IS FOR RIGHT INDEX OF THE SUB ARRAY OF arr HAVE TO BE SORTED \*/

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

if (l < r){

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

// SORT FIRST & LAST HALVES

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

/\* UTILITY FUNCTIONS \*/

/\* PRINT ARRAY \*/

void printArray(int A[], int size){

int i;

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

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

printf("\n");

}

//MAIN METHOD \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main(){

int arr[]=

{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 arr[] = {12, 11, 13, 5, 6, 7};

//int arr[] = {3,2,1,5,4,8};

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

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("GIVEN ARRAY WHICH HAVE TO BE SORT IS: \n\n");

printArray(arr, arr\_size);

//MARGE SORTING METHOD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

mergeSort(arr, 0, arr\_size - 1);

printf("\nSORTED ARRAY USING MARGE SORT ALGORITHM IS:\n\n");

printArray(arr, arr\_size);

//PRINT TOTAL COMPARISON AND SWAP COUNT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

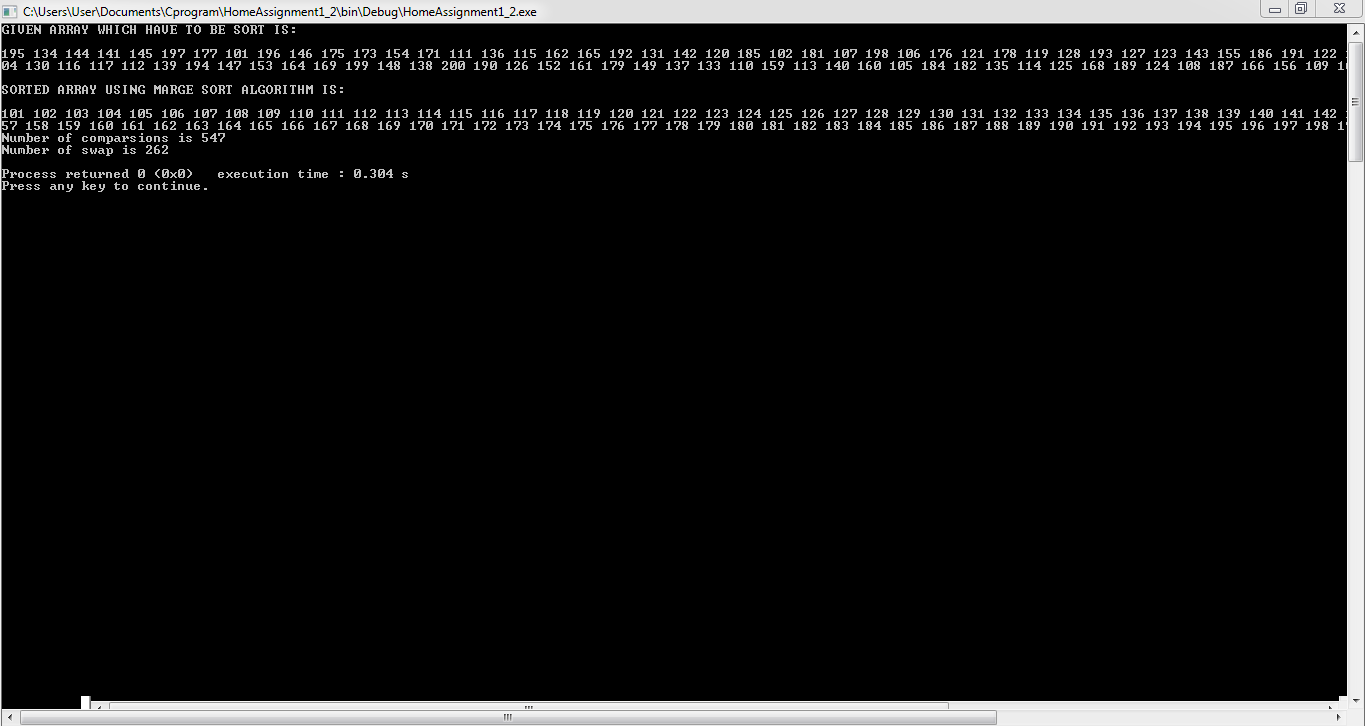
printf("Number of swap is %d\n", swapcounter);

return 0;

}

# **OUTPUT :**

**Fig :2- Merge Sort In Ascending**



# 

# 

# 

# 

# 

# **CODE FOR QUICK SORT WITH COMPARISON & SWAP :**

#include<stdio.h>

//QUICK SORT PROGRAM C CODE

// FUNCTION SWAP TWO ELEMENTS

int comparsioncounter=0;

int swapcounter=0;

//METHOD FOR SWAP \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int t = \*a;

\*a = \*b;

\*b = t;

}

// THIS FUNCTION TAKES LAST ELEMENTS 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];

int i = (low - 1);

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

comparsioncounter++;

if (arr[j] <= pivot)

{

i++;

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

if (i<j){

swapcounter++;

}

}

}

swap(&arr[i + 1], &arr[high]);

if (i+1<high){

swapcounter++;

}

return (i + 1);

}

// ARRAY TO BE SORTED, LOW-->STARTING INDEX, HIGH-->ENDING INDEX\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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);

}

}

//PRINT ARRAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

for (int i=0; i <=size; i++)

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

printf("\n");

}

// MAIN METHOD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main(){

int arr[]=

{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 arr[] = {10, 7, 8, 9, 1, 5};

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

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

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("GIVEN ARRAY WHICH HAVE TO BE SORTED: \n\n");

printArray(arr, arr\_size);

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

quickSort(arr, 0, arr\_size-1);

printf("QUICK SORTED ARRAY IS : \n\n");

printArray(arr, arr\_size);

//PRINT THE NUMBER OF COMPARISONS AND SWAPS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

printf("\n\nNUMBER OF COMPARISONS IS: %d\n\n", comparsioncounter);

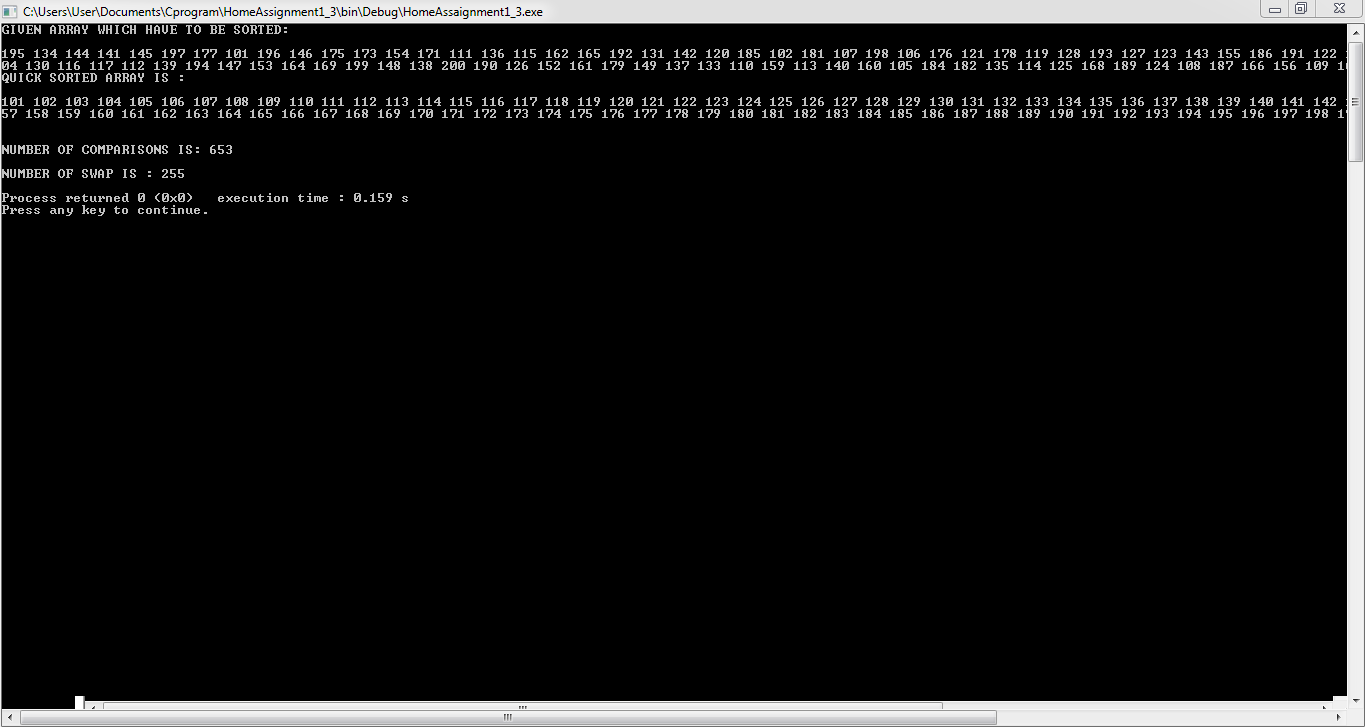
printf("NUMBER OF SWAP IS : %d\n", swapcounter);

return 0;

}

# **OUTPUT :**

**Fig :3- Quick Sort In Ascending order**



# **CODE FOR [101-200] SORT IN ASCENDING ORDER :**

#include <stdio.h>

//PRINT ARRAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int i;

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

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

printf("\n");

}

//MAIN METHOD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main(){

int arr[100]=

{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 temp=0;

int arr\_size = sizeof(arr)/sizeof(arr[0]);

//PRINT THE GIVEN ARRAY

printf("GIVEN ARRAY WHICH HAVE TO BE SORT IS: \n");

printArray(arr, arr\_size);

// CODE FOR ASCENDING ORDER OF ARRAY ELEMENT

for (int i = 0; i < arr\_size; i++){

for (int j = i+1; j < arr\_size; j++){

if (arr[j] < arr[i]){

int tmp = arr[i]; //Using temporary variable for storing

arr[i] = arr[j]; //replacing value

arr[j] = tmp;

}

}

}

//PRINT ARRAY IN ASCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

printf("\nALREADY IN ASCENDING SORT OF THAT ARRAY:\n");

for(int i=0; i< arr\_size; i++){

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

}

return 0;

}

# **OUTPUT :**

**Fig :4- [101-200] array in ascending order**



# 

# **CODE FOR [200-101] SORT IN DESCENDING ORDER :**

#include <stdio.h>

//PRINT ARRAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int i;

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

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

printf("\n");

}

//MAIN METHOD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main(){

int arr[100]=

{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 temp=0;

int arr\_size = sizeof(arr)/sizeof(arr[0]);

//PRINT THE GIVEN ARRAY\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

printf("GIVEN ARRAY WHICH HAVE TO BE SORT IS: \n\n");

printArray(arr, arr\_size);

// CODE FOR ASCENDING ORDER OF ARRAY ELEMENT

for (int i = 0; i < arr\_size; i++){

for (int j = i+1; j < arr\_size; j++){

if (arr[j] > arr[i]){

int tmp = arr[i]; //Using temporary variable for storing last value

arr[i] = arr[j]; //replacing value

arr[j] = tmp;

}

}

}

//PRINT ARRAY IN DESCENDING ORDER\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

printf("\nALREADY IN DESCENDING SORT OF THAT ARRAY:\n");

printf("\n");

for(int i=0; i< arr\_size; i++){

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

}

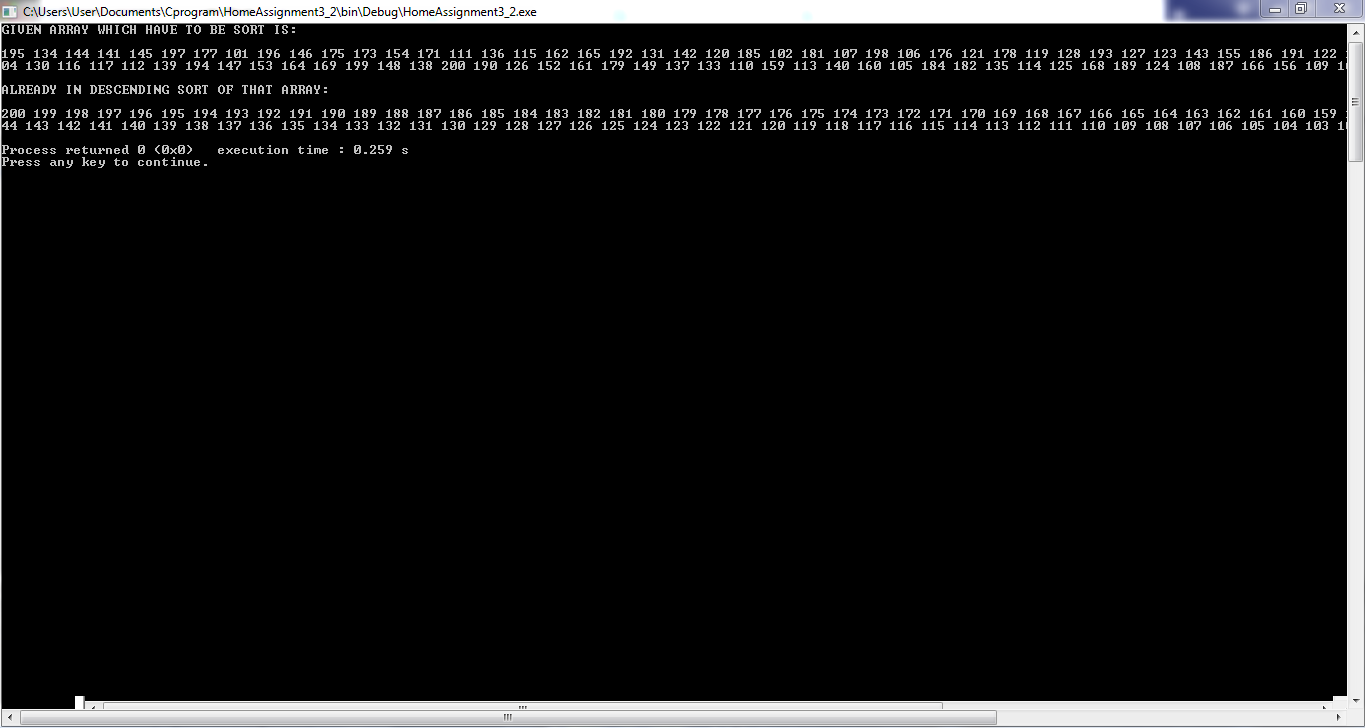
printf("\n");

return 0;

}

# **OUTPUT :**

**Fig :5- [200-101] Like Descending Order**



# 

# **CODE FOR [101-200] SORT IN RANDOM ORDER :**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

//101-200 SORT USING THE RANDOM FUNCTION

// FUNCTION TO SWAP THE INTEGER ARRAY ELEMENT to swap to integers

//METHOD FOR SWAP\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

int temp = \*a;

\*a = \*b;

\*b = temp;

}

// PRINT ARRAY FUNCTIOKN\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

for (int i = 0; i < size; i++)

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

printf("\n");

}

// A FUNCTION TO GENERATE A RANDOM NUMBERS\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

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

srand ( time(NULL) );

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

// PIC A RANDOM INDEX FROM 0 TO i

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

// SWAP arr[i] WITH THE ELEMENT AT RANDOM INDEXS

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

}

}

//MAIN METHOD\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

int main(){

int arr[100] =

{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 i;

int arr\_size = sizeof(arr)/sizeof(arr[0]);

printf("GIVEN ARRAY WHICH HAVE TO BE REARANGE WITH RANDOM: \n");

printf("\n");

for(int i = 0; i < arr\_size; i++){

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

}

//PRINT RANDOM ARRAY ELEMENT\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*//

printf("\n");

printf("\nSORTED USING RANDOM SORT IS : \n\n");

int n = sizeof(arr)/ sizeof(arr[0]);

randomize (arr, n);

printArray(arr, n);

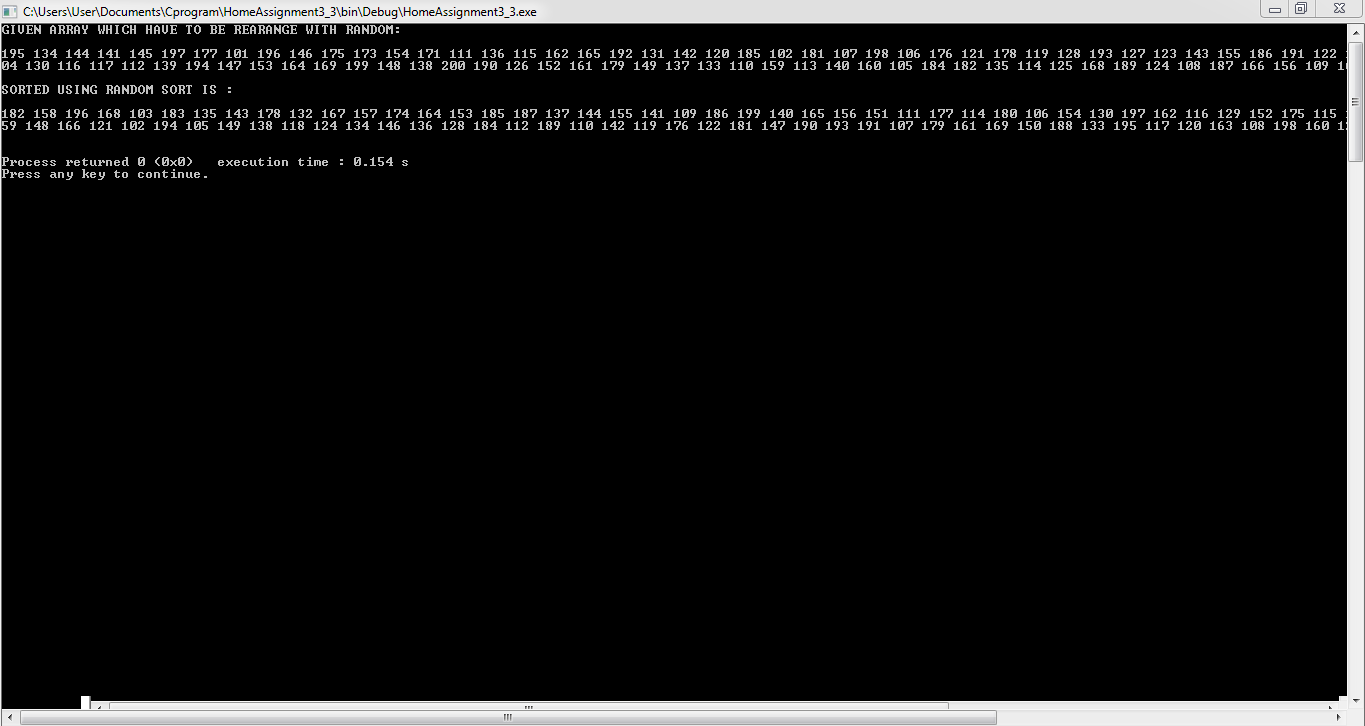
printf("\n");

return 0;

}

# **OUTPUT :**

**Fig :6- [101-200] Array Sort In Random Order**



# **MATERIALS :**

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

# 

# **PROCEDURE :**

1. C Programming
2. C++ Programming
3. Java Programming

# **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**)

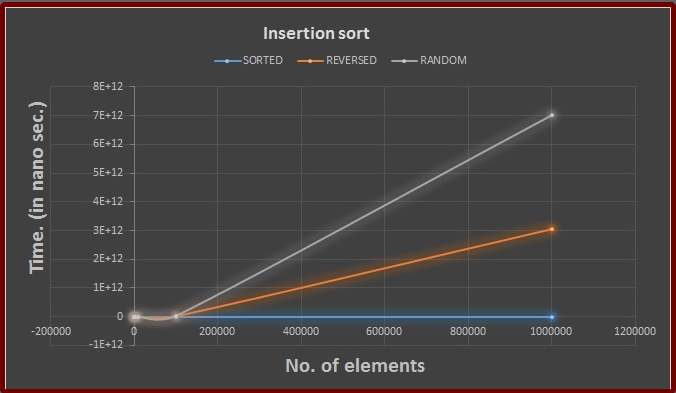
Insertion sort gets penalized if comparison or copying is slow. In other words, the maximum array size which is faster to sort with insertion sort compared to O(*n*log*n*) algorithms gets smaller if comparison or copying of the array elements is slow.

**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.

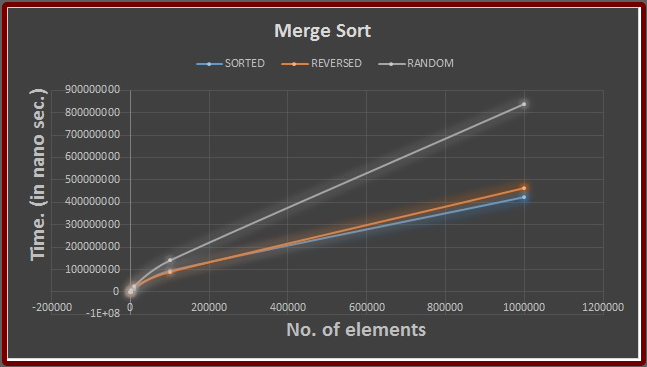
**Time complexity:**

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.

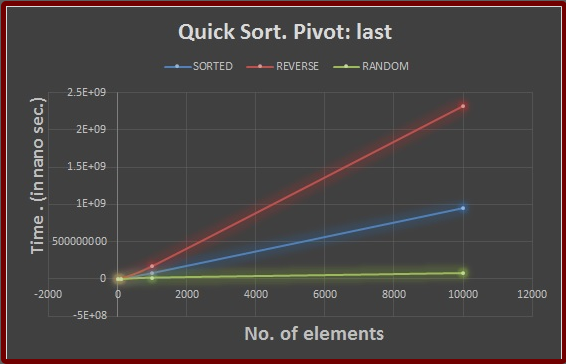
**Time complexity:**

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**



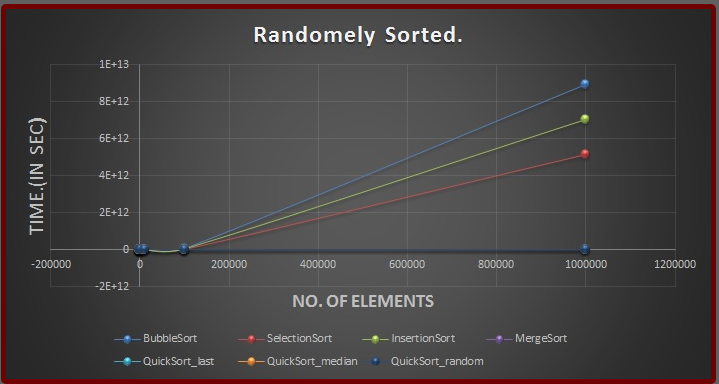
1. For **Random sort** approach,Most of the times this is going to be the approach since we are going to sort random number elements

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-codes***
2. ***Stack Overflow***
3. ***GeeksforGeeks***
4. ***Other Blog Sites***
5. ***Books Help***

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