**Data Structures and Algorithms**

**Lab-11**

**Name:** Ahmad Amjad Mughal

**Reg No:** 121672

**Class:** BSCS-6C

**Task 1**

**Insertion sort:**

//Ahmad Amjad Mughal

//BSCS-6C

//121672

#include <iostream> //input output library

using namespace std; //for prompt commands

void insertionSort(int array[], int n) //function that performs sorting

{

int i, key, j;

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

{

key = array[i]; //sets second element of array as a key

j = i - 1; //assign first element j

while (j >= 0 && array[j] > key) //comparsion of numbers

{

array[j + 1] = array[j];

j = j - 1;

}

array[j + 1] = key;

}

}

void display(int array[], int n) //function that displays the sorted array

{

int i;

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

cout <<" "<< array[i];

cout << endl;

}

int main()

{

int array[] = {13,11,8,5,4,1,9 };

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

//function calls

insertionSort(array, n);

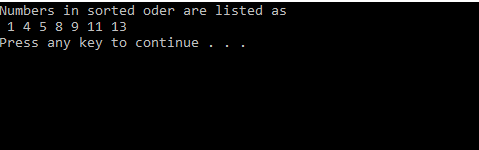
cout << "Numbers in sorted oder are listed as " << endl;

display(array, n);

return 0;

**}**

**Screenshot:**



**Merge Sort:**

#include <iostream>

using namespace std;

void Merge(int \*a, int low, int high, int mid)

{

// We have low to mid and mid+1 to high already sorted.

int i, j, k, temp[10- 0 + 1];

i = low;

k = 0;

j = mid + 1;

// Merge the two parts into temp[].

while (i <= mid && j <= high)

{

if (a[i] < a[j])

{

temp[k] = a[i];

k++;

i++;

}

else

{

temp[k] = a[j];

k++;

j++;

}

}

// Insert all the remaining values from i to mid into temp[].

while (i <= mid)

{

temp[k] = a[i];

k++;

i++;

}

// Insert all the remaining values from j to high into temp[].

while (j <= high)

{

temp[k] = a[j];

k++;

j++;

}

// Assign sorted data stored in temp[] to a[].

for (i = low; i <= high; i++)

{

a[i] = temp[i - low];

}

}

// A function to split array into two parts.

void MergeSort(int \*a, int low, int high)

{

int mid;

if (low < high)

{

mid = (low + high) / 2;

// Split the data into two half.

MergeSort(a, low, mid);

MergeSort(a, mid + 1, high);

// Merge them to get sorted output.

Merge(a, low, high, mid);

}

}

int main()

{

int n, i;

cout << "\nEnter the number of data element to be sorted: ";

cin >> n;

int arr[10];

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

{

cout << "Enter element " << i + 1 << ": ";

cin >> arr[i];

}

MergeSort(arr, 0, n - 1);

// Printing the sorted data.

cout << "\nSorted Data ";

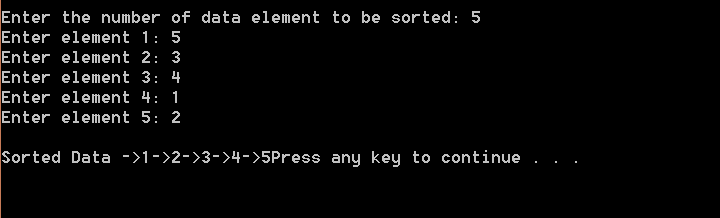
for (i = 0; i < n; i++)

cout << "->" << arr[i];

return 0;

}

**Screenshot:**

****

**Task 2:**

**Insertion sort:**

//Ahmad Amjad Mughal

//BSCS-6C

//121672

#include <ctime>

#include <iostream> //input output library

using namespace std; //for prompt commands

void insertionSort(int array[], int n) //function that performs sorting

{

int i, key, j;

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

{

key = array[i]; //sets second element of array as a key

j = i - 1; //assign first element j

while (j >= 0 && array[j] > key) //comparsion of numbers

{

array[j + 1] = array[j];

j = j - 1;

}

array[j + 1] = key;

}

}

void display(int array[], int n) //function that displays the sorted array

{

int i;

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

cout << array[i];

}

void random()

{

int random = 100000;

int arrayr[100000];

srand(time(NULL));

for (int number = 0; number < 10000; number++)

arrayr[number] = rand() % 100000 + 1;

clock\_t start = clock();

insertionSort(arrayr, random);

clock\_t stop = clock();

double iteration\_time = double(stop - start) / CLOCKS\_PER\_SEC;

cout << "randomized sorted Array Elapsed time is " << iteration\_time << endl;

}

void ascending()

{

int random = 100000;

int arrayr[100000];

srand(time(NULL));

for (int number = 0; number < 100000; number++)

arrayr[number] = rand() % random + 1;

insertionSort(arrayr, random);

clock\_t start = clock();

insertionSort(arrayr, random);

clock\_t stop = clock();

double iteration\_time = double(stop - start) / CLOCKS\_PER\_SEC;

cout << "Ascending order sorted array elapsed time is " << iteration\_time << endl;

}

void descending(){

int random = 100000;

int arrayr[100000];

int arraydesc[100000];

srand(time(NULL));

for (int number= 0; number < random; number++) {

arrayr[number] = rand() % random + 1;

}

insertionSort(arrayr, random);

for (int i = 100000, j = 0; i>0, j <= 100000; i--, j++){

arraydesc[j] = arrayr[i];

}

clock\_t start = clock(); // start timer

insertionSort(arraydesc, random);

clock\_t stop = clock();

double iteration\_time = double(stop - start) / CLOCKS\_PER\_SEC; //time in seconds

cout << "Descending order sorted array elapsed time is " << iteration\_time << endl;

};

int main()

{

random();

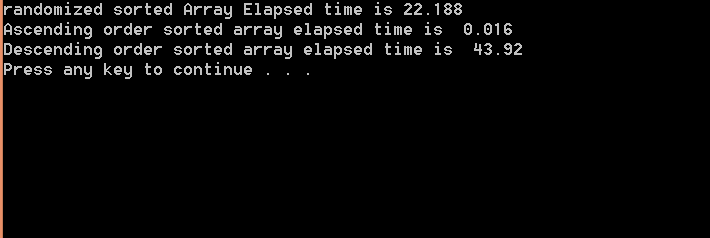
ascending();

descending();

return 0;

}

**Output:**



**Merge sort:**

#include <iostream>

#include <stdlib.h>

#include <ctime>

using namespace std;

void merge(int arr[], int p, int q, int r)

{

int i, j, k;

int\* L;

int\* R;

int n1 = q - p + 1;

int n2 = r - q;

L = new int[n1];

R = new int[n2];

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

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

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

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

i = 0;

j = 0;

k = p;

while (i < n1 && j < n2)

{

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

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

void mergeSort(int arr[], int p, int r)

{

if (p < r)

{

int q = p + (r - p) / 2;

mergeSort(arr, p, q);

mergeSort(arr, q + 1, r);

merge(arr, p, q, r);

}

}

/\* Function to print an array \*/

void printArray(int A[], int size)

{

int i;

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

cout << A[i] << " ";

cout << endl;

}

void randomFunction(){

int arrayr[100000];

srand(time(NULL));

for (int number = 0; number < 100000; number++) {

arrayr[number] = rand() % 100000 + 1;

}

double time\_iteration;

clock\_t start = clock(); // start timer

mergeSort(arrayr, 0, (100000 - 1));

clock\_t stop = clock();

time\_iteration = double(stop - start) / CLOCKS\_PER\_SEC; //time in seconds

cout << "Time consumed for sorting random numbers: " << time\_iteration << endl;

}

void ascendingFunction(){

int arrayr[100000];

srand(time(NULL));

for (int number = 0; number < 100000; number++) {

arrayr[number] = rand() % 100000 + 1;

}

mergeSort(arrayr, 0, (100000 - 1));

double time\_iteration;

clock\_t start = clock(); // start timer

mergeSort(arrayr, 0, (100000 - 1));

clock\_t stop = clock();

time\_iteration = double(stop - start) / CLOCKS\_PER\_SEC; //time in seconds

cout << "Time consumed for already sorted numbers: " << time\_iteration << endl;

}

void descendingFunction(){

int arrayr[100000];

int arraydec[100000];

srand(time(NULL));

for (int number = 0; number < 100000; number++) {

arrayr[number] = rand() % 100000 + 1;

}

mergeSort(arrayr, 0, (100000 - 1));

for (int i = (100000 - 1), j = 0; i >= 0, j < 100000; i--, j++){

arraydec[j] = arrayr[i];

}

double time\_iteration;

clock\_t start = clock(); // start timer

mergeSort(arraydec, 0, (100000 - 1));

clock\_t stop = clock();

time\_iteration = double(stop - start) / CLOCKS\_PER\_SEC; //time in seconds

cout << "Time consumed when numbers are sorted in descending order: " << time\_iteration << endl;

};

int main()

{

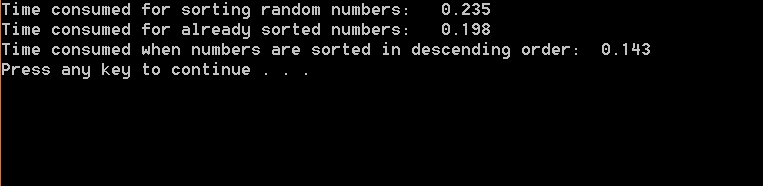
randomFunction();

ascendingFunction();

descendingFunction();

}

**Output:**

****

**Comparison:**

* Merge sort is much better than insertion sort as time complexity of merge sort is O(nlogn) and that of insertion sort is O(n^2).
* Merge sort has same time consumption for best, average and worst cases while in case of insertion sort, time complexity is different for different case.
* merge sort is better in time complexity when there is random array.
* Inserion sort is better for measuring time complexity of ascending numbers. Merge sort creates two sub arrays and calculate the time for each comparsion.
* Merge sort is better for measuring time complexity of descending numbers. Time complexity of worst case of insertion is 43.92 sec and for merge it is 0.143 sec.
* In merge sort, for every array we make two sub arrays and then merge them so we get same time for all cases. In insertion sort, we pick an element, compare it with the element at its left, and then sort accordingly so time depends on the previous arrangement of elements.