**Data Structures and Algorithms**

**Lab-10**

Name: Ahmad Amjad Mughal

Reg No: 121672

Class: BSCS-6C

**Task 1:**

//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);

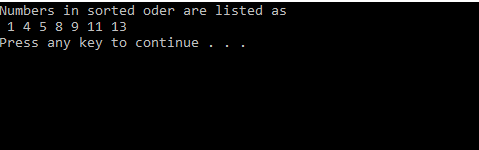
cout << "Numbers in sorted oder are listed as " << endl;

display(array, n);

return 0;

**}**

**Output:**

****

**Task 2+3:**

//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();

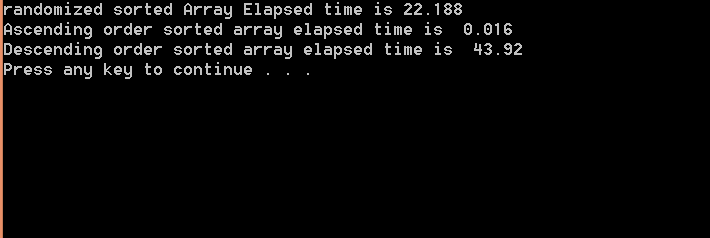
ascending();

descending();

return 0;

}

**Output:**

****

**Best case:** The best case input is an array that is already sorted .If we take a closer look at the insertion sort code, we can notice that every iteration of while loop reduces one inversion. The While loop executes only if **arr[i] < arr[j].** Therefore overall time complexity of the insertion sort is **O(n + f(n))** where f(n) is inversion count. If the inversion count is **O(n),** then the time complexity of insertion sort is O(n). In this case insertion sort has a linear running time (i.e., O(*n*)).During each iteration, the first remaining element of the input is only compared with the right-most element of the sorted subsection of the array.

**Average Case:** Average case is applied when array list comprises of numbers in random order. On average, we'd expect that each element is less than half the elements to its left. The running time would be half of the worst-case running time. Average case would still be O(n2).

**Worst Case:** The worst simplest worst case input is an array sorted in reverse order. The set of all worst case inputs consists of all arrays where each element is the smallest or second-smallest of the elements before it. In these cases, inner loop shift the sub sorted list of array before new sorting. This gives insertion sort a quadratic running time (i.e., O(*n*2)).