**Batch: B3 Roll. No.: 121**

**Experiment: 11**

**Grade: AA / AB / BB / BC / CC / CD /DD**

|  |
| --- |
| **Title:**  Implementation of sorting algorithms |

**Objective:** To understand various searching methods

**Expected Outcome of Experiment:**

|  |  |
| --- | --- |
| **CO** | **Outcome** |
| **CO3** | Demonstrate sorting and searching methods. |

**Websites/books referred:**

**1. .** Michael T. Goodrich, Roberto Tamassia, and David M. Mount. 2009. Data Structures and Algorithms in C++ (2nd. ed.). Wiley Publishing

**2.** Aaron M. Tenenbaum, Yedidyah Langsam, and Moshe J. Augenstein. Data structures using C.

**3.** [**https://classroom.google.com/u/0/c/NTM5NDM0NzUxMTQ2**](https://classroom.google.com/u/0/c/NTM5NDM0NzUxMTQ2)

**4.** [**https://www.geeksforgeeks.org/stable-and-unstable-sorting-algorithms/#:~:text=What%20is%20a%20stable%20sorting,the%20algorithm%20treats%20equal%20elements**](https://www.geeksforgeeks.org/stable-and-unstable-sorting-algorithms/#:~:text=What%20is%20a%20stable%20sorting,the%20algorithm%20treats%20equal%20elements)**.**

**5.** [**https://www.javatpoint.com/bubble-sort**](https://www.javatpoint.com/bubble-sort)

**6.** [**https://www.javatpoint.com/insertion-sort**](https://www.javatpoint.com/insertion-sort)

**7.** [**https://www.interviewkickstart.com/learn/comparison-among-bubble-sort-selection-sort-and-insertion-sort**](https://www.interviewkickstart.com/learn/comparison-among-bubble-sort-selection-sort-and-insertion-sort)

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Abstract**: -

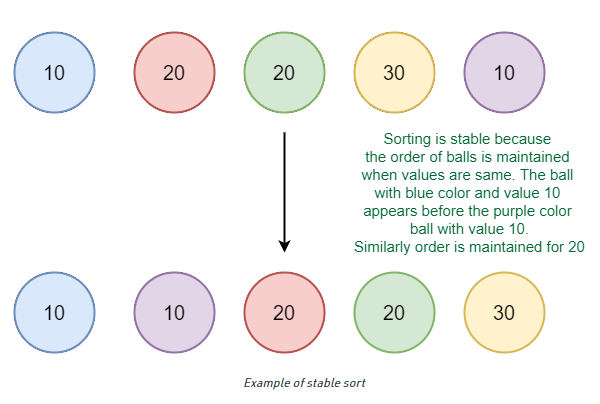
Sorting is any process of arranging items systematically in a particular order.

An Ascending Order sort is one in which n keys are arranged in such a way that keyi < keyj for any i and j such that i<j.

A Descending Order sort is one in which n keys are arranged in such a way that keyi > keyj for any i and j such that i<j.

**Related Theory:** *(Explain stable sort, in place sort, number of passes in a sorting algo)*

Stable Sort: A sorting algorithm is said to be stable if two objects with equal keys appear in the same order in sorted output as they appear in the input data set.



Consider the following data set of student names and their respective class sections.

(Dhanvantari, A)

(Achyut, B)

(Keshav, A)

(Emperuman, B)

(Chaitanya, A)

If this data is sorted according to name only, then it is highly unlikely that the resulting dataset will be grouped according to sections as well.

(Achyut, B)

(Chaitanya, A)

(Dhanvantari, A)

(Emperuman, B)

(Keshav, A)

So the list would have to be sorted again to obtain that list of students section-wise too. But in doing so, if the sorting algorithm is not stable, the result could be something like:

(Chaitanya, A)

(Dhanvantari, A)

(Keshav, A)

(Emperuman, B)

(Achyut, B)

The data set is now sorted according to section, but not according to name. In the name-sorted data set, (Achyut, B) was before (Emperuman, B), but since the sorting algorithm is not stable, the relative order is lost. On the other hand, a stable sorting algorithm would give a result like:

(Chaitanya, A)

(Dhanvantari, A)

(Keshav, A)

(Achyut, B)

(Emperuman, B)

**In place Sort:** An in-place sorting algorithm uses constant space for producing the output (i.e., it modifies the given array only). It sorts the list only by modifying the order of the elements within the list. For example, Insertion Sort and Selection Sorts are in-place sorting algorithms as they do not use any additional space for sorting the list. A typical implementation of Merge Sort and Counting Sort is not in-place.

**Assigned Sorting Algorithm:**

Bubble sort and Insertion sort

**Code and output screenshots for assigned sorting algorithm:**

**Bubble Sort:**

**Code:**

#include<stdio.h>

void display(int a[], int);

void bubble(int a[], int);

void main()

{

int n, a[100];

printf("\nEnter the size of the array, which must not exceed 100: ");

do

{

scanf("%d", &n);

if(n>100)

printf("\nThe array size must not exceed 100.\nEnter again: ");

}while(n>100);

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

{

printf("\nEnter element %d: ", (i+1));

scanf("%d", &a[i]);

}

printf("\nThe array before sorting is - \n");

display(a, n);

bubble(a, n);

printf("\nThe array after sorting is - \n");

display(a, n);

}

void display(int a[], int n)

{

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

printf("%d\t", a[i]);

}

void bubble(int a[], int n)

{

int i, j, temp;

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

{

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

{

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

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

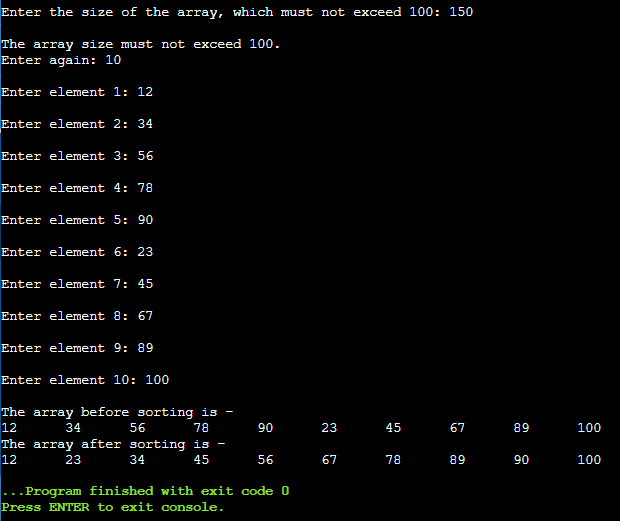
}

}

}

}

**Output:**

****

**Insertion Sort:**

**Code:**

#include<stdio.h>

void display(int a[], int);

void insertion(int a[], int);

void main()

{

int n, a[100];

printf("\nEnter the size of the array, which must not exceed 100: ");

do{

scanf("%d", &n);

if(n>100)

printf("\nThe array size must not exceed 100.\nEnter again: ");

}while(n>100);

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

printf("\nEnter element %d: ", (i+1));

scanf("%d", &a[i]);

}

printf("\nThe array before sorting is - \n");

display(a, n);

insertion(a, n);

printf("\nThe array after sorting is - \n");

display(a, n);

}

void display(int a[], int n){

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

printf("%d\t", a[i]);

}

void insertion(int a[], int n){

int i, j, temp;

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

temp = a[i];

j = i - 1;

while(j >= 0 && temp <= a[j]){

a[j+1] = a[j];

j = j - 1;

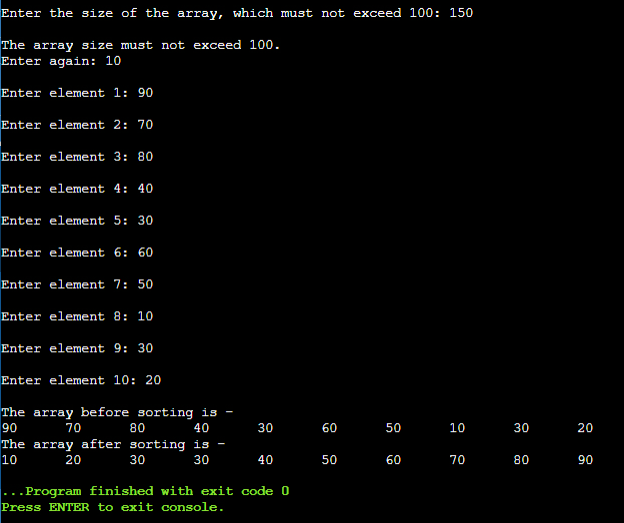
}

a[j+1] = temp;

}

}

**Output:**

****

**Conclusion: -**

Thus, in this experiment, the concept of sorting was learnt and different types of sorting was implemented. The various kinds of sorting algorithms come with their specific advantages and disadvantages. Therefore, there cannot be a reliable general comparison among them. Instead, the application where sorting is required is studied and based on that, a particular sorting algorithm would be used so that there is best utilization of computer memory and time.

**Post lab questions-**

1. **Compare and contrast various sorting algorithms.**

**Ans.**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr. No.** | **Bubble Sort** | **Insertion Sort** | **Counting Sort** |
| **1.** | Best-case time complexity is O(n2). | Best-case time complexity is O(n). | Best-case time complexity is O(n + k). |
| **2.** | Average-case time complexity is O(n2). | Average-case time complexity is O(n2). | Average-case time complexity is O(n + k). |
| **3.** | Worst-case time complexity is O(n2). | Worst-case time complexity is O(n2). | Worst-case time complexity is O(n + k). |
| **4.** | It is stable. | It is stable. | It is stable. |
| **5.** | It is in-place. | It is in-place. | It is not in-place. |
| **6.** | It is the simplest stable in-place sorting method and very easy to code. | It is effective when comparison operations are costly. | It works better than other comparison-based sorting algorithms when many numbers have to be sorted that lie in a comparatively smaller range on the number line. |

1. **Comment on the input (sorted in ascending order/descending order/random) and time required for execution of sorting algorithm.**