Comparison of Different Sorting Techniques

INTRODUCTION

The program compares Bubble, Insertion and Selection sorting technique with different sets of input and different ranges of array.

The program aims to demonstrate when to use the right sorting technique for faster speed.

Bubble Sort - This is the slowest technique amongst others. Bubble sort is almost useless to implement for any given range of input and array size. For larger arrays(with random elements of the order 10^5), it might take several minutes to sort.

Selection Sort - Better than bubble sort. But performs Selection Sort performs the same. In a selection sort, for each element that is to be added to the sorted section, you have to scan the *entire* unsorted part of the list to find the minimum remaining element.

Insertion Sort - this actually pretty good on **nearly sorted collections**. But otherwise, selection and insertion require same amount of time.Usually, insertion sort will perform less comparisons than selection sort, depending on the degree of "sortedness" of the array.

In an insertion sort, you have to search the *sorted* section to find out where the next element goes but the search ends once you have found the insertion point (about half way through the sorted section on average).

In this program, user is given two choice in the beginning:

1. **<u>To compare bubble, insertion and selection sort independently for different range of values and a generate multiple sets of results showing time taken as output.</u>**

here the user can try with smaller range of elements in the array and conclude that there is no significant difference between the time taken by each technique. But as the array gets bigger, there is more time difference between bubble and the other two sorts. Hence bubble should not be used when the array is big. Between other two, the time difference is not that much.

2. **To compare between insertion and selection sort for different degree of 'sortedness' .**

here, the user will notice that for completely unsorted array, the time taken by selection and insertion is nearly the same. But as degree of sortedness increases, insertion sort is performed much faster than selection sort.

Degree of sortedness:It is the percentage of number of elements which already sorted to the total number of elements.

In the program, for 80% sortedness, user shall enter 80 without the percentage sign.

Description of the code:

Header file used.
Time.h : to use clock() & CLOCK_PER_SEC.
stdlib.h: to use srand function, malloc function.

Class comparisons: has,
Private:
1.length: data member of type int to store the range of the array.
2.*list1, *list2, *list3: pointer of int type later dynamically allocated.
2. Void selection: a member function which has the selection sort algorithm.
3. Void insertion: a member function which has the insertion sort algorithm.
4. Void bubble: a member function which has the bubble sort algorithm.
5. Option1,option2: member function which perform the analysis on the sorting algorithms.
Public: constructor of the class to call the function menu.

Member function description:
1.**menu()**: gives user choice between option1() and option2() and calls the required member function respectively.

2. **Insertion(), selection() and bubble()**: Contain their respective sorting technique code. To find the time difference clock() function is used. Before the sorting code, t1,t2 of type double(which can hold 64 bit floating point) is used and assigned to clock(). Which checks the time at that instance. And after the sorting code, t2 is again assigned to clock(). The time difference between t1 and t2 gives the time taken for performing the sort.

3. **Option1()**: user input of number of elements and the number of sets of results to be obtained is taken. Memory allocation for the variables list1,2 and 3 is done accordingly. Srand with parameter of time is taken to generate the required random numbers and stored in all three arrays. It further calls all three sorting functions.

4**.option2()**: range of array is set to 1 lakh elements and the degree of sortedness is taken as input. Using srand function, selective no. of random elements are generated and inserted. Rest are inserted using for loop. It further calls all three sorting functions.

5. **main()**: used to create the class-type variable z.

OUTPUTS:

```
                                TIME COMPARISON BETWEEN DIFFERENT SORTING TECHNIQUES


Enter the choice:
1. Comparison between 3 sorts
2.Between insertion and selection based on degree of sortedness.
0.exit
$$$$


_
enter the no of elements to generate in the array: 80000

How many sets of results do you want to obtain? 2
Insertion Sort  : 3.69972 sec
selection Sort  : 5.37937 sec
bubble Sort     : 16.7561 sec

Insertion Sort  : 3.6859 sec
selection Sort  : 5.38607 sec
bubble Sort     : 16.721 sec
```

```
Enter the choice:
1. Comparison between 3 sorts
2.Between insertion and selection based on degree of sortedness.
0.exit
2
comparison between insertion and selection sort using array of 1 lakh elements of various degree of sortedness:enter the degree of sortedness in %
90


Insertion Sort  : 2.26895 sec
selection Sort  : 8.50187 sec
```

```
Enter the choice:
1. Comparison between 3 sorts
2.Between insertion and selection based on degree of sortedness.
0.exit
2
comparison between insertion and selection sort using array of 1 lakh elements of various degree of sortedness:enter the degree of sortedness in %
80


Insertion Sort  : 4.10917 sec
selection Sort  : 8.50247 sec



Enter the choice:
1. Comparison between 3 sorts
2.Between insertion and selection based on degree of sortedness.
0.exit
2
comparison between insertion and selection sort using array of 1 lakh elements of various degree of sortedness:enter the degree of sortedness in %
70


Insertion Sort  : 5.47488 sec
selection Sort  : 8.49655 sec
```

```
Enter the choice:
1. Comparison between 3 sorts
2.Between insertion and selection based on degree of sortedness.
0.exit
1
enter the no of elements to generate in the array: 50000

How many sets of results do you want to obtain? 2
Insertion Sort  : 1.50252 sec
selection Sort  : 2.10174 sec
bubble Sort     : 6.4082 sec

Insertion Sort  : 1.4455 sec
selection Sort  : 2.0991 sec
bubble Sort     : 6.40171 sec
```

Reference:

https://stackoverflow.com/questions/15799034/insertion-sort-vs-selection-sort

https://www.codingunit.com/c-reference-stdlib-h-function-srand-initialize-random-number-generator

https://www.tutorialspoint.com/c_standard_library/time_h.htm

https://pediaa.com/what-is-the-difference-between-bubble-sort-and-insertion-sort/