**Experiment 3**

**Write a program to the comparative analysis of Iterative Sorting Algorithm (Bubble sort, Insertion sort, Selection sort).**

**Theory**

Sorting algorithms play a pivotal role in computer science and data processing, as they enable the arrangement of data elements in a specific order, such as ascending or descending. This lab assignment focuses on the comparative analysis of three widely used iterative sorting algorithms: Bubble sort, Insertion sort, and Selection sort. Each of these algorithms employs a distinct strategy to sort a list of elements, and their efficiency can vary significantly based on the input data characteristics. The objective of this analysis is to gain a deeper understanding of the performance characteristics of these sorting algorithms by evaluating their time complexity, execution efficiency, and suitability for different data sets. Through empirical analysis, we aim to quantify and compare their relative efficiency, helping us determine the most appropriate sorting algorithm for various real-world scenarios. This investigation will shed light on the strengths and weaknesses of each algorithm, aiding in informed algorithm selection when faced with sorting tasks in practical applications.

**Code**

Bubble Sort

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void BubbleSort(int \*arr,int \*size);

int main()

{

clock\_t start, end;

double time;

printf("-----BUBBLE SORT--------\n");

int size, arr[500000], num;

printf("\nEnter size of array\n");

scanf("%d",&size);

printf("%d numbers\n",size);

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

{

num = (rand()%10000);

arr[i] = num;

}

start = clock();

BubbleSort(arr,&size);

end = clock();

printf("\nSORTED ARRAY\n");

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

{

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

}

printf("\n\n");

time=((double)(end-start)\*10000) / CLOCKS\_PER\_SEC;

printf("Time=%lf micro", time);

}

void swap(int \*first,int \*second)

{

int temp;

temp=\*first;

\*first=\*second;

\*second=temp;

}

void BubbleSort(int \*arr,int \*size)

{

int temp,counter=1;

while (counter<\*size)

{

for (int i = 0; i < \*size-counter; i++)

{

if(arr[i]>arr[i+1])

{

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

}

}

counter++;

}

}

Insertion Sort

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void InsertionSort(int \*arr,int \*size);

int main()

{

clock\_t start, end;

double time;

printf("-----INSERTION SORT--------\n");

int size, arr[500000], num;

printf("\nEnter size of array\n");

scanf("%d",&size);

printf("%d numbers\n",size);

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

{

num = (rand()%10000);

arr[i] = num;

}

start = clock();

InsertionSort(arr,&size);

end = clock();

printf("\nSORTED ARRAY\n");

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

{

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

}

printf("\n\n");

time=((double)(end-start)\*10000) / CLOCKS\_PER\_SEC;

printf("Time=%lf micro", time);

}

void swap(int \*first,int \*second)

{

int temp;

temp=\*first;

\*first=\*second;

\*second=temp;

}

void InsertionSort(int \*arr,int \*size)

{

for (int i = 1; i < \*size; i++)

{

int current = arr[i] ,j = i-1;

while (arr[j] >= current && j>=0)

{

swap(&arr[j],&arr[j+1]);

j--;

}

arr[j+1]=current;

}

}

Selection Sort

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

void selectionSort(int \*arr,int \*size);

int main()

{

clock\_t start, end;

double time;

printf("-----SELECTION SORT--------\n");

int size, arr[500000], num;

printf("\nEnter size of array\n");

scanf("%d",&size);

printf("\n%d numbers\n",size);

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

{

num = (rand()%10000);

arr[i] = num;

}

start = clock();

selectionSort(arr,&size);

end = clock();

printf("\nSORTED ARRAY\n");

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

{

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

}

printf("\n\n");

time=((double)(end-start)\*10000) / CLOCKS\_PER\_SEC;

printf("Time=%lf micro", time);

}

void swap(int \*first,int \*second)

{

int temp;

temp=\*first;

\*first=\*second;

\*second=temp;

}

void selectionSort(int \*arr,int \*size)

{

int temp,counter=0;

while (counter<\*size)

{

for (int i = counter; i < \*size; i++)

{

if(arr[counter]>arr[i])

{

temp=arr[counter];

arr[counter]=arr[i];

arr[i]=temp;

}

}

counter++;

}

}

**Result** **Analysis**

This experiment has been conducted in a 64-bit system with 16 GB RAM a 12th Gen Intel® Core™ i7-1255 1.7 GHz. The algorithm is implemented in C programming language (GCC 4.9.2 64-bit), using Dev-C++ IDE.

In this experiment, the algorithms for Bubble Sort, Insertion Sort and Selection Sort have been implemented and executed for various values of n (Input Size). The different values measured during the experiment are tabulated in the table. The measured time is in microseconds.

|  |  |  |  |
| --- | --- | --- | --- |
| Size of array | Bubble | Insertion | Selection |
| 10000 | 1720 | 780 | 2500 |
| 12500 | 2810 | 1250 | 4060 |
| 15000 | 4370 | 1560 | 5470 |
| 17500 | 5940 | 2190 | 7200 |
| 20000 | 8450 | 2660 | 8920 |

The graph shown below is the plot of input n and time in milliseconds taken by the algorithm.

Based on the above table and graph it is clearly seen that Insertion Sort works efficiently than other Iterative Searching Algorithms.

**Conclusion**

In this experiment it has been found that the Insertion Sort works more efficiently than other Iterative Searching Algorithms. We also verified that the time complexity of Iterative Searching Algorithms in best case is O (n).