**Experiment 1**

**Write a program to perform the empirical analysis of iterative algorithm to find nth Fibonacci number.**

**Theory**

The Fibonacci sequence is a fundamental mathematical sequence where each term is the sum of the two preceding ones, typically starting with 0 and 1 (or 1 and 1). While there exist various methods to compute the nth Fibonacci number, one commonly employed approach is the iterative algorithm. This algorithm involves a step-by-step computation by iteratively adding the last two Fibonacci numbers until the desired nth number is reached. The purpose of this lab assignment is to perform an empirical analysis of this iterative algorithm to gain insights into its time complexity, execution efficiency, and how it scales with the input size. Through this analysis, we aim to evaluate the practical performance of the algorithm and understand its suitability for computing large Fibonacci numbers in real-world applications.

**Code**

#include<stdio.h>

#include<time.h>

int main()

{

int n, i;

double first=0, second=1, temp, time;

clock\_t start, end;

printf("Enter the position of fibonacci number:");

scanf("%d", &n);

start = clock();

printf("%f, %f", first, second);

i = 3;

while(i <= n)

{

temp = first + second;

first = second;

second = temp;

printf("%f \n", temp);

i++;

}

printf("\n\n\n");

end = clock();

printf("The nth fibonacci number is: %lf \n",temp);

time = ((double)(end-start)\*1000) / CLOCKS\_PER\_SEC;

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

}

**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 program to find nth Fibonacci number has been implemented and executed for various values of n. The different values measured during the experiment are tabulated in the table.

|  |  |
| --- | --- |
| **Input Size** | **Time** |
| 1000 | 126 |
| 1500 | 189 |
| 2500 | 257 |
| 3500 | 313 |
| 4500 | 401 |
| 5500 | 424 |
| 6400 | 455 |
| 7500 | 486 |
| 8500 | 659 |
| 9500 | 1153 |
| 10000 | 1181 |

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 seen that the input number(n) has linear relationship with the time taken.

**Conclusion**

In this experiment it has been found that the size of input (n) has linear relationship with the time taken by the system to find the nth Fibonacci number. This is equivalent with the asymptotic time complexity of the algorithm. Hence, this experiment proves complexity of the algorithm to find nth Fibonacci number is O (n).

**Experiment 2**

**Write a program to perform the empirical analysis of iterative algorithm for linear search.**

**Theory**

The linear search algorithm, also known as sequential search, is a simple and intuitive method used to locate a specific element within a list or array. It involves scanning each element in the list one by one until the target element is found or the entire list has been traversed. The efficiency of this algorithm is a crucial aspect to consider, especially when dealing with large datasets, as it directly impacts the time required to find a particular item. In this lab assignment, we aim to perform an empirical analysis of the iterative linear search algorithm. Through this analysis, we will investigate how the algorithm's performance scales with varying input sizes, enabling us to evaluate its efficiency in practical scenarios. By measuring and analyzing the execution times and comparing them for different input sizes, we will gain valuable insights into the algorithm's time complexity and its suitability for specific search tasks.

**Code**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int LinearSearch(int \*arr, int n,int key);

int main()

{

clock\_t start, end;

double time;

int arr[500000],n,key, num;

printf("-----Linear Searching--------\n");

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

scanf("%d",&n);

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

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

{

num = (rand()%10000);

arr[i] = num;

}

printf("Enter the key\n");

scanf("%d",&key);

start = clock();

int ans = LinearSearch(arr,n,key);

if(ans!=-1)

{

printf("\n%d is in %d index.\n",key,ans);

}

else

{

printf("\nNot Found\n");

}

end = clock();

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

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

}

int LinearSearch(int \*arr, int n,int key)

{

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

{

if(arr[i]== key)

{

return i+1;

}

}

return -1;

}

**Result** **Analysis**

This experiment has been conducted in a 64-bit system with 16 GB RAM a 12th Gen Intel® Core™ i5-12500H 3.10 GHz. The algorithm is implemented in C programming language in Visual Studio Code 1.81.1 Code Editor.

In this experiment, the program to find nth Fibonacci number has been implemented and executed for various values of n. The different values measured during the experiment are tabulated in the table.

|  |  |
| --- | --- |
| **Input Size** | **Time (Microsecond)** |
| 10000 | 20 |
| 20000 | 20 |
| 30000 | 30 |
| 40000 | 10 |
| 50000 | 20 |
| 60000 | 30 |
| 70000 | 20 |
| 80000 | 20 |

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 the size of array n has linear relationship with the time taken by the system to find the key in it.

**Conclusion**

In this experiment it has been found that the size of array(n) has linear relationship with the time taken by the system to find the key in it. This is equivalent with the asymptotic time complexity of the algorithm. Hence, this experiment proves complexity of the algorithm to linear search is O (n).