Experiment-1

WAP to perform empirical analysis of Iterative algorithm to find nth Fibonacci number.

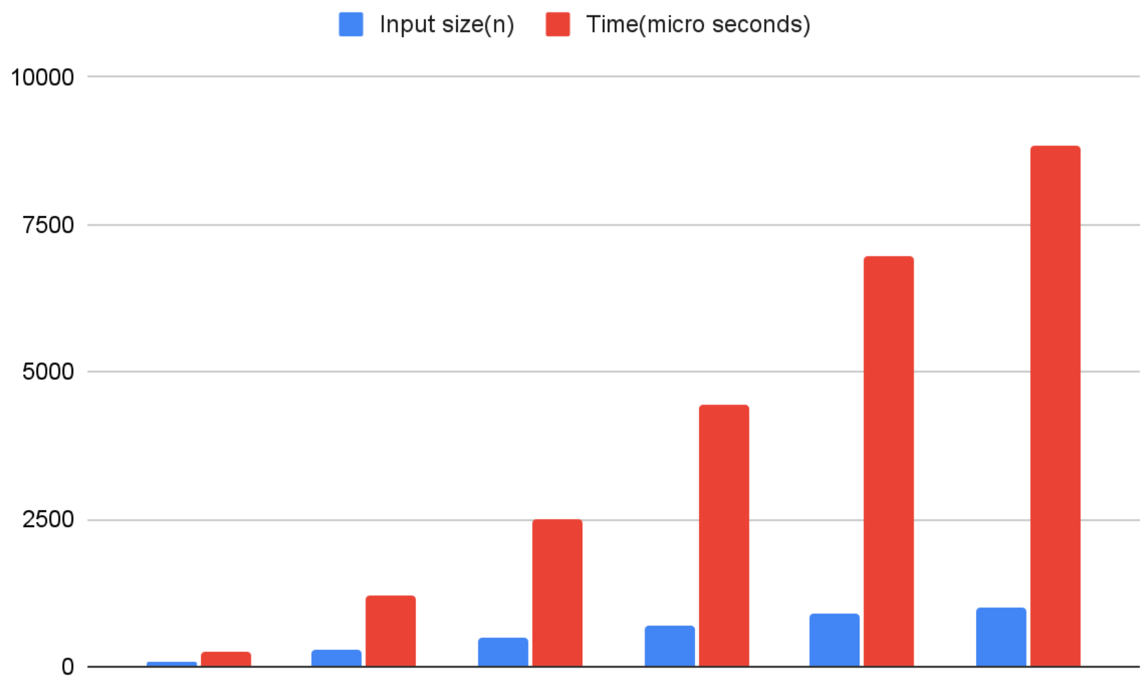
```
/* Program to find the nth Fibonacci number */
#include <stdio.h>
#include <time.h>
int main()
{
   int n, i;
   double first = 0, second = 1, temp, time;
   clock_t start, end;
   printf("Enter the position of fibonacci number:");
   scanf("%d", &n);
   start = clock();
   printf("%f,%f", first, 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) * 1000000) / CLOCKS_PER_SEC;
   printf("Time=%lf microseconds", time);
}
```

Result Analysis and Discussion:

This experiment has been conducted in a 64-bit system with 8 GB RAM and Intel® Core™ i3-10110U × 4. The algorithm is implemented in the C programming language in VS Code IDE. In this experiment the algorithm to find the nth Fibonacci number has been implemented and executed for a different value of n. During this experiment for different values of n the time taken by the algorithm has been measured and tabulated as shown in table below.

Input size(n)	Time(microseconds)
100	253
300	1210
500	2492
700	4442
900	6958
1000	8848

The graph shown below is the plot of input n and the time in microseconds taken by the algorithm while running on a system recorded in table above.



Based on the above table and graph it is clearly seen that the input number n has linear relationship with the time taken by the system to find the nth Fibonacci number of input number n.

Conclusion:

In this experiment it has been found that the size of input (n) has a 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 the complexity of the algorithm to find nth Fibonacci number is O (n).

Experiment-2

WAP to perform empirical analysis of Iterative algorithm for Bubble sort algorithm.

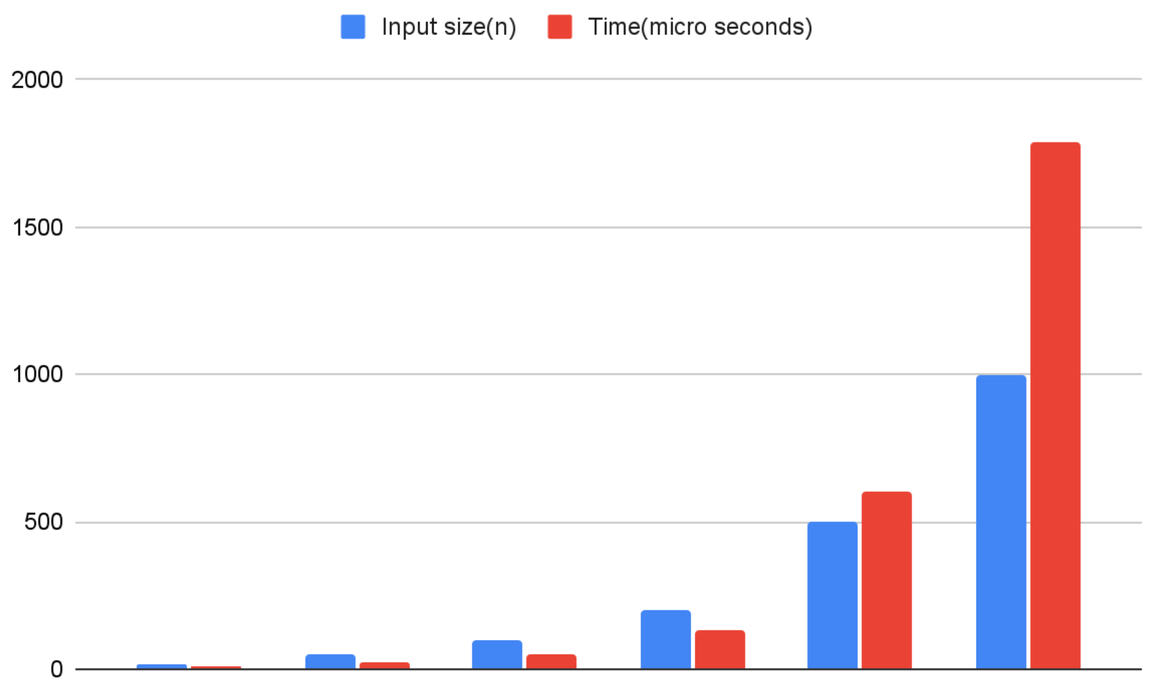
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

// Function to perform Bubble Sort on an array
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap arr[j] and arr[j + 1]
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}

int main() {
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];

    // Generate random integers and populate the array
    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000; // Generate random integers between 0 and 999
    }

    clock_t start, end;
    double time;

    start = clock();

    bubbleSort(arr, n);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    end = clock();

    time = ((double)(end - start) * 1000000) / CLOCKS_PER_SEC;
    printf("\nTime taken for Bubble Sort = %lf microseconds\n", time);

    return 0;
}
```

Result Analysis and Discussion:

This experiment has been conducted in a 64-bit system with 8 GB RAM and Intel® Core™ i3-10110U × 4. The algorithm is implemented in the C programming language in VS Code IDE. In this experiment the algorithm to find the nth Fibonacci number has been implemented and executed for a different value of n. During this experiment for different values of n the time taken by the algorithm has been measured and tabulated as shown in table below.

Input size(n)	Time(microseconds)
20	13
50	22
100	49
200	135
500	605
1000	1785

The graph shown below is the plot of input n and the time in microseconds taken by the algorithm while running on a system recorded in table above.



Based on the above table and graph it is clearly seen that the input number n has linear relationship with the time taken by the system to find the nth Fibonacci number of input number n.

Conclusion:

In this experiment, it's evident that the execution time of the Bubble Sort algorithm grows quadratically with the input size (n), consistent with its expected worst-case time complexity of O(n^2).

Experiment-3

WAP to perform empirical analysis of Iterative algorithm for Insertion sort algorithm.

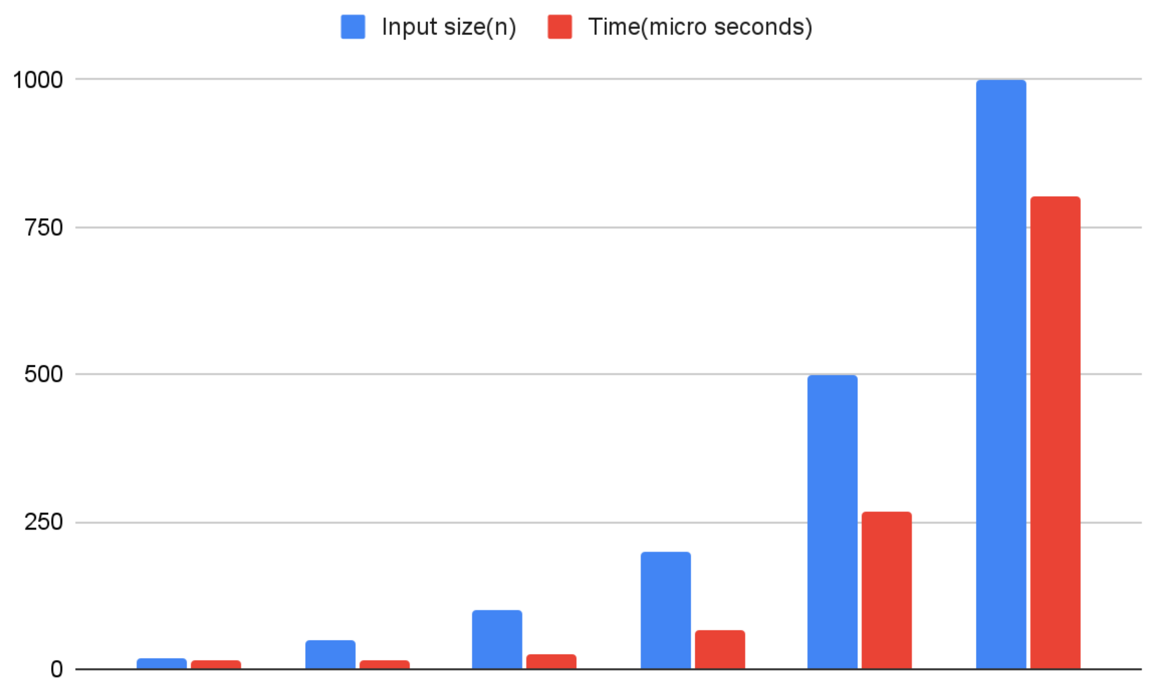
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

void insertionSort(int arr[], int n) {
    int i, key, j;
    for (i = 1; i < n; i++) {
        key = arr[i];
        j = i - 1;

        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}

int main() {
    int n;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];

    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
    }

    clock_t start, end;
    double time;

    start = clock();

    insertionSort(arr, n);
    printf("Sorted array:\n");
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    end = clock();

    time = ((double)(end - start) * 1000000) / CLOCKS_PER_SEC;
    printf("\nTime taken for Insertion Sort = %lf microseconds\n", time);

    return 0;
}
```

Result Analysis and Discussion:

This experiment has been conducted in a 64-bit system with 8 GB RAM and Intel® Core™ i3-10110U × 4. The algorithm is implemented in the C programming language in VS Code IDE. In this experiment the algorithm to find the nth Fibonacci number has been implemented and executed for a different value of n. During this experiment for different values of n the time taken by the algorithm has been measured and tabulated as shown in table below.

Input size(n)	Time(microseconds)
20	14
50	16
100	24
200	65
500	267
1000	802

The graph shown below is the plot of input n and the time in microseconds taken by the algorithm while running on a system recorded in table above.



Conclusion:

In summary, based on the empirical data and the observed trend, we can conclude that the Insertion Sort algorithm demonstrates a near-quadratic time complexity, making it less efficient for sorting large datasets when compared to more efficient sorting algorithms with better average and worst-case time complexities.

Experiment-4

WAP to perform empirical analysis of Iterative algorithm for Insertion sort algorithm.

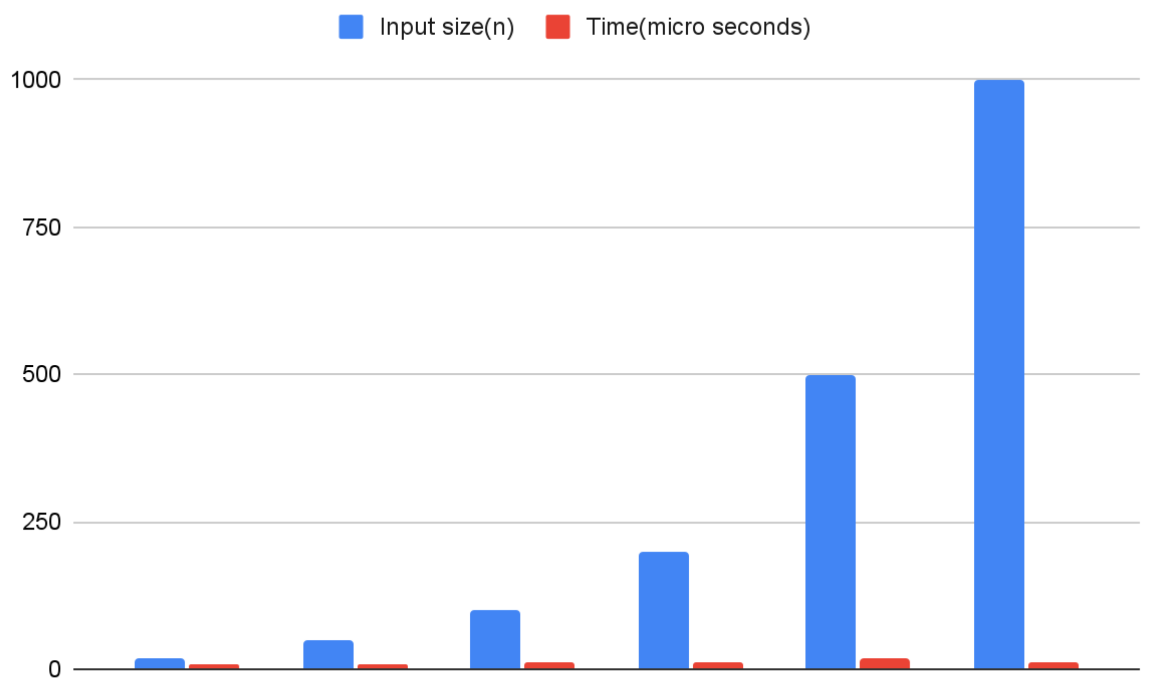
```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int sequentialSearch(int arr[], int n, int x) {
    for (int i = 0; i < n; i++) {
        if (arr[i] == x) {
            return i;
        }
    }
    return -1;
}

int main() {
    int n, x;
    printf("Enter the size of the array: ");
    scanf("%d", &n);

    int arr[n];

    srand(time(NULL));
    for (int i = 0; i < n; i++) {
        arr[i] = rand() % 1000;
    }

    x = rand() % 1000;

    clock_t start, end;
    double time;

    start = clock();

    int result = sequentialSearch(arr, n, x);



    if (result != -1) {
        printf("Element %d found at index %d\n", x, result);
    } else {
        printf("Element %d not found in the array\n", x);
    }
    end = clock();

    time = ((double)(end - start) * 1000000) / CLOCKS_PER_SEC;
    printf("Time taken for Sequential Search = %lf  microseconds\n", time);

    return 0;
}
```

Result Analysis and Discussion:

This experiment has been conducted in a 64-bit system with 8 GB RAM and Intel® Core™ i3-10110U × 4. The algorithm is implemented in the C programming language in VS Code IDE. In this experiment the algorithm to find the nth Fibonacci number has been implemented and executed for a different value of n. During this experiment for different values of n the time taken by the algorithm has been measured and tabulated as shown in table below.

Input size(n)	Time(microseconds)
20	10
50	8
100	13
200	11
500	17
1000	11

The graph shown below is the plot of input n and the time in microseconds taken by the algorithm while running on a system recorded in table above.



Conclusion:

In conclusion, the empirical data supports the theoretical analysis, affirming that the Sequential Search algorithm has a time complexity of O(n), making it an efficient choice for searching in smaller datasets where the overhead of more complex search algorithms may not be justified.