Indian Institute of Technology, Kanpur
Computer Science & Engineering Department

Programming Assignment 1

Pratham Sahu 210755
Aditya Bangar 210069

ESO207A – Data Structures and Algorithms

Does the efficiency of algorithms really matter ?

7th August, 2022

The Objective of the Assignment

The followings are the objectives of this assignment.

1. To investigate whether the efficiency of an algorithm really matters in real world ?
2. To verify how well the RAM model of computation captures the time complexity of an algorithm.

Background of the Assignment

Fibonacci numbers have many applications in theoretical as well as applied computer science. They are used in pseudo-random number generators, Fibonacci heap data structure, analyzing efficiency of Euclids algorithm etc. Fibonacci numbers are also found in natural patterns like flower petals. Let F(n) denote nth Fibonacci number. Each of you would have written a code for Fibonacci numbers during the course ESC101. Consider a related computational problem whose input is an integer n, and output is (F(n) mod 2022). In the lecture, three algorithms for solving this problem were discussed. The first algorithm was recursive and denoted by Rfib. The second algorithm was iterative and denoted by Ifib. And the third algorithm, CleverFib was quite complex – it involved repeated squaring to compute some power of a cleverly defined 2×2 matrix.

Table of Contents

Problem 1

Contents

minted

1.1 Problem Statement

The implementation of each algorithm is supposed to work for each possible value of n upto 10^{18} . However, it will turn out that some of these algorithms will start taking too much time as n increases. In order to observe this on your own, for each algorithm, you have to experimentally determine the largest possible value of n for which the corresponding implementation gives the output within the time limit (in seconds) from the set 0.001, 0.1, 1, 5, 60, 600. You need to fill up the table given below with these values. If value of n exceeds 1018 for a specific time interval from the set, write $> 10^{18}$ in the corresponding entry in the table.

Time $\rightarrow$	0.001 sec	0.1 sec	1 sec	5 sec	60 sec	600 sec
RFib						
Ifib						
CleverFib						

1.2 RFib Algorithm

The RFib Algorithm is implemented using Recursion with the fibonacci of n-1 and n-2 called recursively. We make use of the following recurrence relation:

$$F(n) = F(n-1) + F(n-2)$$

However, this algorithm is not very efficient with time complexity:

$$O(n) > 2^{(n-2)/2}$$

The algorithm implemented using C Programming Langauge is given below

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

int Rfib(int n, int m){
    if(n==0){
        return 0;
    }
    else if(n==1){
        return 1;
    }
    else{
        return (Rfib(n-1,m)+Rfib(n-2,m))%m;
    }
}

int main(){
    int n=0;
    int m=0;
    printf("ENTER n,m :");
    scanf("%i, %i", &n, &m);
     clock_t start, end;
    double cpu_time_used;
    start = clock();

    int c=Rfib(n,m);
    printf("Output: %d",c);
    end = clock();

    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;

    printf("\nTime taken: %f\n", cpu_time_used);

    return 0;
}
```

1.3 IFib Algorithm

The Ifib algorithm is implemented by using Iterative techniques. We run a loop that keeps adding the fibonacci of previous two numbers that are stored in two variables 'a' and 'b'. These variables are updated in each iteration.

This algorithm fares better than the RFib algorithm with a time complexity:

$$O(n) = n$$

The algorithm implemented using C Programming Langauge is given below

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

int main(){
    double time_taken=0;
    long long int n;
    long long int m=2022;
    printf("Enter n : ");
    scanf("%lld", &n);
    long long int a=0,b=1,c;
    clock_t start = clock();
    for(long long int i=2;i<=n;i++){
        c=(a+b)%2022;
        a=b;
        b=c;
    }
    clock_t end = clock();
    time_taken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("output: %lld \n", c);
    printf("time taken: %lf \n", time_taken);
    return 0;
}
```

1.4 CleverFib Algorithm

The CleverFib algorithm is the most efficient of the three algorithms with a much better time complexity. This is implemented by using the concepts of linear algebra. We know that

$$F(n) = F(n-1) + F(n-2)$$

Now using this, we can state the following:

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F(1) \\ F(0) \end{bmatrix}$$

Now solving this recursively, we achieve the following result:

$$\begin{bmatrix} F(n) \\ F(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \begin{bmatrix} F(n-1) \\ F(n-2) \end{bmatrix}$$

Thus this has effectively reduced to a problem of finding powers, since we already know F(1) and F(0). This algorithm has a time complexity much better than the other two algorithms discussed before with:

$$O(n) = log_2 n$$

This is the best time complexity that can be achieved for calculating powers.
The algorithm implemented using C Programming Language is given below

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

long long int m=0;

struct mat{
    long long int a11;
    long long int a12;
    long long int a21;
    long long int a22;
   };

struct mat multiply(struct mat a, struct mat b){
    struct mat c;
    c.a11=(a.a11*b.a11+a.a12*b.a21)%m;
    c.a12=(a.a11*b.a12+a.a12*b.a22)%m;
    c.a21=(a.a21*b.a11+a.a22*b.a21)%m;
    c.a22=(a.a21*b.a12+a.a22*b.a22)%m;
    return c;
}


struct mat power(struct mat a, long long int n){
    if(n==0){
        struct mat i;
        i.a11=1;
        i.a12=0;
        i.a21=0;
        i.a22=1;
        return i;
    }

    else{
        struct mat temp=power(a,n/2);
        temp=multiply(temp,temp);
        if(n%2==1){
            temp=multiply(temp,a);
        }
        return temp;
    }

}

int main(){
    long long int n=0;
    printf("ENTER n,m: ");
    scanf("%lld, %lld", &n, &m);
    struct mat a;
    a.a11=1;
    a.a12=1;
    a.a21=1;
    a.a22=0;

    clock_t start, end;
    double cpu_time_used;
    start = clock();
    struct mat c=power(a, n-1);
    printf("Output: %lld",(c.a11));
    end = clock();
    cpu_time_used = ((double) (end - start)) / CLOCKS_PER_SEC;
    printf("\nTime taken: %f\n", cpu_time_used);
    return 0;
}
```

1.5 Experimental determination table

The readings for this table were achieved by implementing trial and error over a range of values to find the number that gave the closest reading to the time intervals given in the table. The time readings are subjective and depend on various factors which include:

- Processor(Most dependent)
- Temperature
- Other weather conditions like Humidity,etc
- Factors like number of processes running, etc

The readings for this assignment were taken on a laptop with the Apple M1 Chip and 8GB of RAM.

Time $\rightarrow$	0.001 sec	0.1 sec	1 sec	5 sec	60 sec	600 sec
RFib	23	35	40	44	49	54
Ifib	$57 \cdot 10^4$	$65 \cdot 10^5$	$85 \cdot 10^6$	$44 \cdot 10^7$	$52 \cdot 10^8$	$54 \cdot 10^9$
CleverFib	$> 10^{18}$	$> 10^{18}$	$> 10^{18}$	$> 10^{18}$	$> 10^{18}$	$> 10^{18}$

Problem 2

Contents

2.1 Problem Statement

Plot the following graphs.
(i) $log_2(Time\ taken\ by\ RFib)$ as a function of n.
(ii) $Time\ taken$ by IFib as a function of n.
(iii) $Time\ taken$ by CleverFib as a function of $log_2(n)$

(a) Provide precise and concise justification for the shapes of the graphs you obtain.
(b) If a graph is a line, what is the value of its slope ? If each graph is a line, can you provide an explanation for the difference in their slopes ?
(c) In each call of CleverFib, the number of instructions executed (excluding the recursive call invoked) are significantly more than RFib. Moreover, CleverFib involved multiplication operations whereas the other two algorithms involved addition operation only. We know that multiplying a pair of numbers takes more time than adding a pair of numbers on a computer.

2.2 Solution

2.2.1 (i), (ii), (iii)

(i) $log_2(Time\ taken\ by\ RFib)$ **as a function of** n

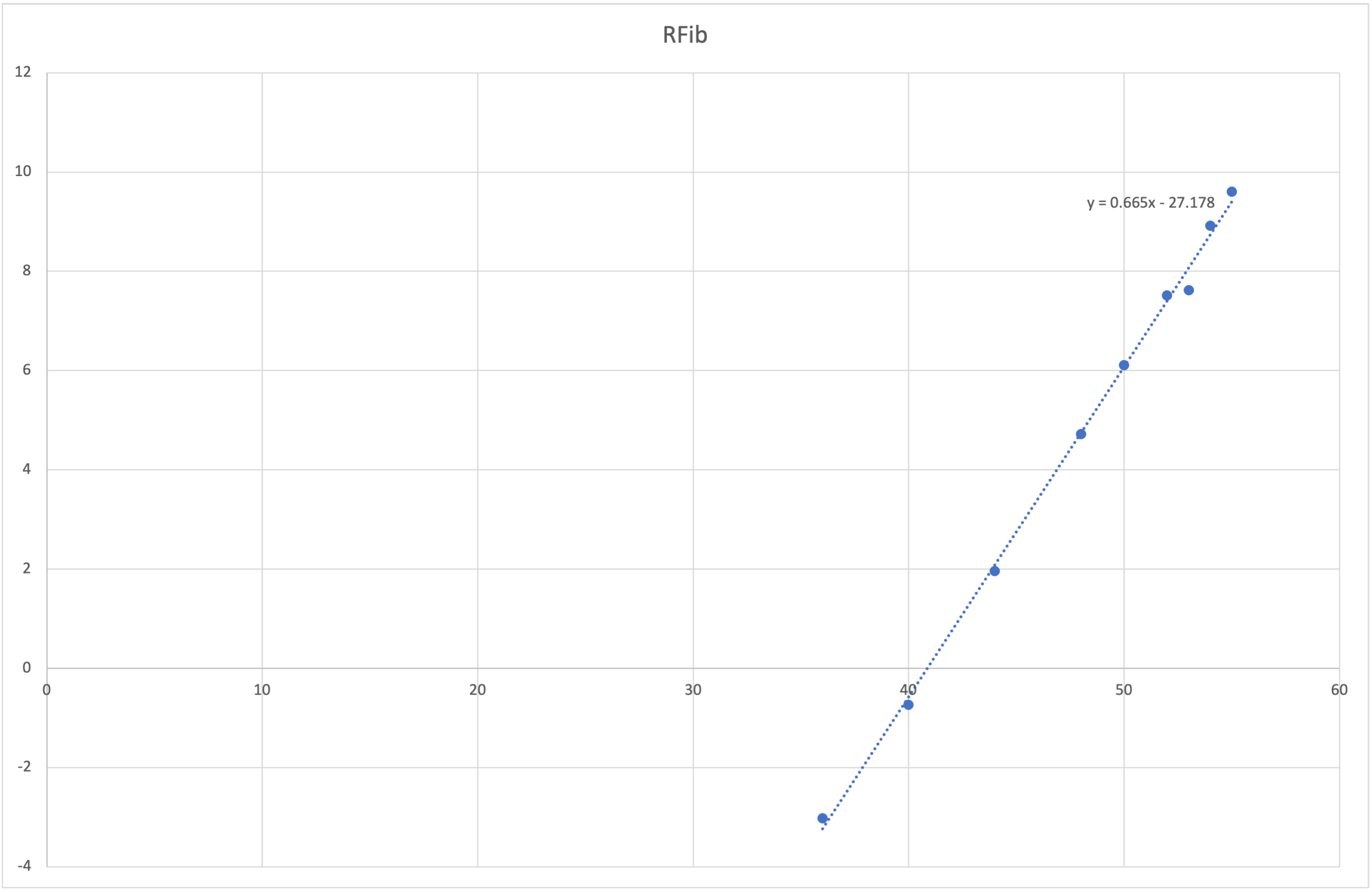

Figure 2.1: y-axis: $log_2(time)$ vs x-axis: n

(ii) *Time taken* by IFib as a function of n

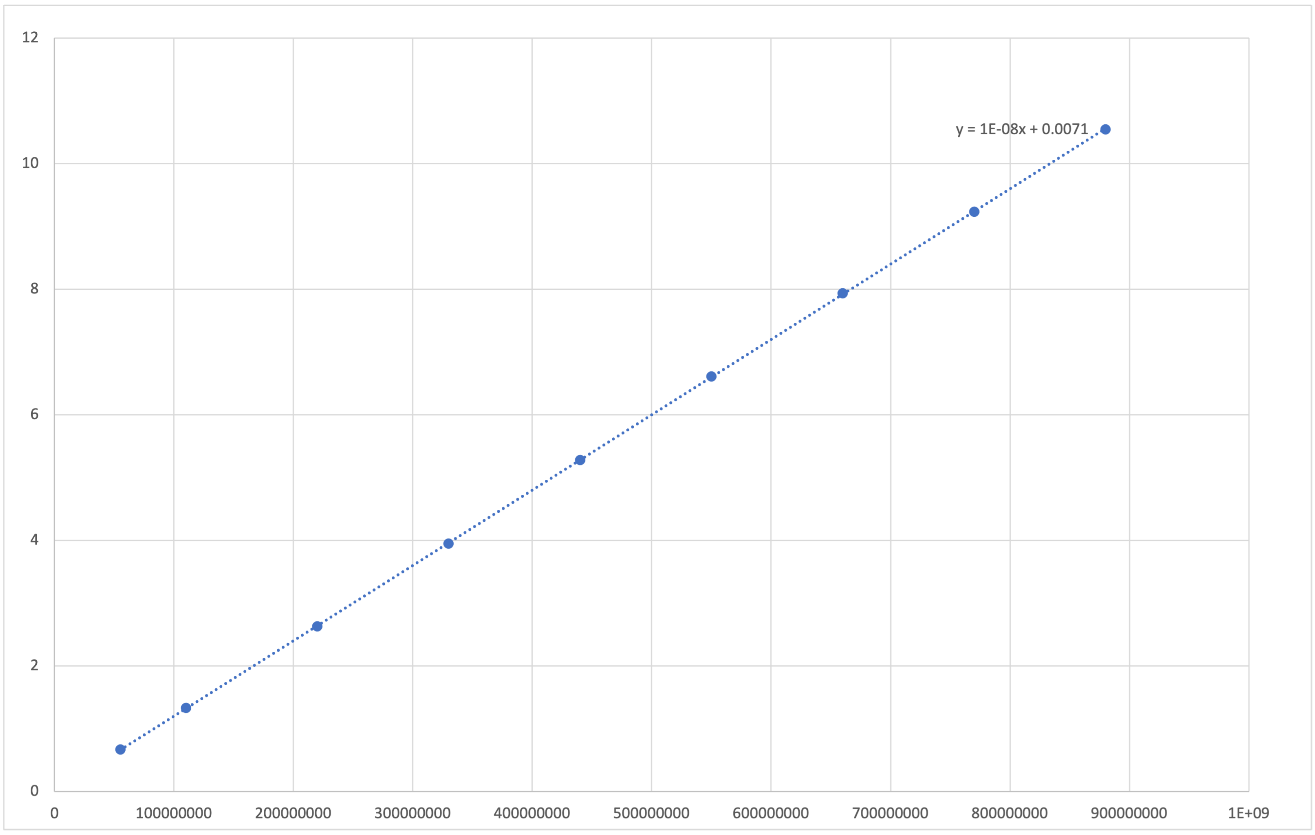

Figure 2.2: y-axis: time(s) vs x-axis: n

(iii) *Time taken* **by CleverFib as a function of** $log_2(n)$

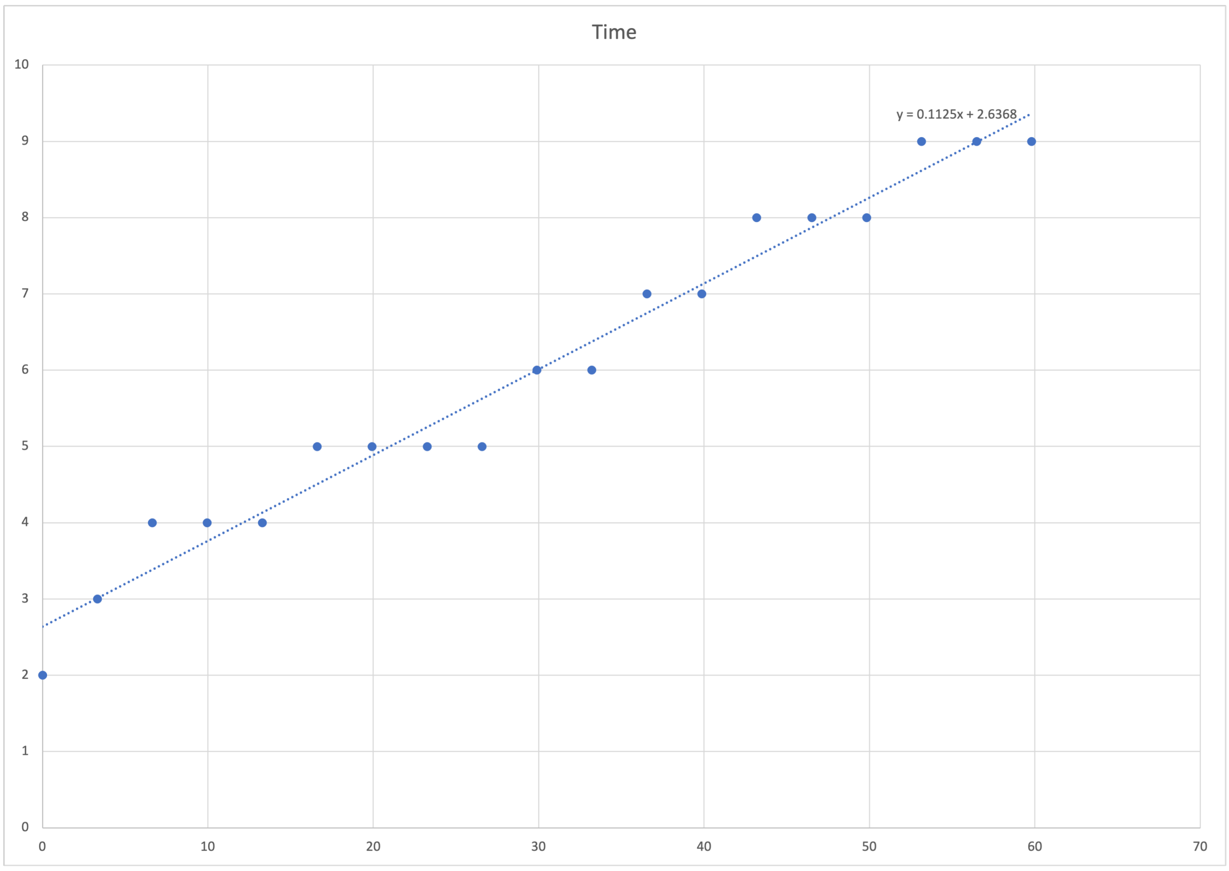

Figure 2.3: y-axis: time ($\times 10^{-6}$) vs x-axis: $log_2 n$

2.2.2 (a), (b), (c)

(a) Provide precise and concise justification for the shapes of the graphs you obtain.

The graph of the various functions are quantitatively plotted by taking readings by running the algorithms, however they can be qualitatively approximated using the word RAM model.

[Note: O(n) always takes the largest or most contributing element in the time complexity calculated by word RAM model.]

- The RFib model has a time complexity $O(n) > 2^{(n-2)/2}$ as stated before.From, the graph we can qualitatively state that the time complexity is of approximately the same form.
- The IFib model has a time complexity $O(n) = n$ as computed from the word RAM model. Thus the graph of time vs n is straight line.
- The CleverFib Algorithm has a time complexity of $O(n) = log_2 n$, thus the graph of time vs $log_2 n$ is a straight line.

(b) If a graph is a line, what is the value of its slope ? If each graph is a line, can you provide an explanation for the difference in their slopes ?

Slope of RFib: 0.665
Slope of IFib: 10^{-8}
Slope of CleverFib: 0.1125×10^-6

The expressions for the time complexity as computed by word RAM model are:

$$RFib(n) > 2^{(n-2)/2}$$

$$IFib(n) = 3n$$

$$CleverFib(n) = 56log_2(n) + 11$$

When we calculate the required quantities which are plotted,

$$log_2(RFib(n)) > (n-2)/2$$

$$IFib(n) = 3n$$

$$CleverFib(n) = 56log_2(n) + 11$$

Thus the graph's are straight lines.

The slopes are different because of the number of underlying operations in each iteration. Also the time complexity is indicative of the relative time taken by numbers for the same algorithm and we cannot compare different algorithms that way.

(c) In each call of CleverFib, the number of instructions executed (excluding the recursive call invoked) are significantly more than RFib. Moreover, CleverFib involved multiplication operations whereas the other two algorithms involved addition operation only. We know that multiplying a pair of numbers takes more time than adding a pair of numbers on a computer.

(i) Did these fact influence the running time of CleverFib ? (You might like to refer to the graph of CleverFib and the graphs of RFib and IFib to answer this question).

These factors do affect the time taken, with each recursive call taking more time to execute. **However** the number of recursive calls in CleverFib are much lower than RFib, thus time taken even for much larger numbers is much lower. Thus on referring to the graphs also we can conclude that time taken is much more in RFib due to higher number of recursive calls dominating over the time taken to execute each call.

(ii) Did these facts affect the relative speed of CleverFib compared to the other 2 algorithms ? If not, then state the reason.

No these facts did not affect the relative time taken to execute these functions, because their effect is neglegible as compared to the change in number of times these statements are called. The number of statements called in RFib»IFib»CleverFib, thus it doesn't affect their relative rates of execution.

Problem 3

Contents

3.1 Problem Statement

Based on the observations and inferences from 1 and 2 above, how accurate did you find the RAM model of computation in measuring the running time of an algorithm ? How accurate did you find the RAM models of computation in comparing the efficiency of a pair of algorithms ?

3.2 Solution

>Using the RAM model for computation we can find the number of steps that the algorithm would require till it's completion, but it is not possible to determine the exact time it will take to compute as the efficiency of the processor varies, also for a given processor many other random factors affect the computation.

>The RAM model of computation provides a very good approximation for the comparison of 2 algorithms, as we can approximately get the function for the time taken according to the number of steps taken in the RAM model for computation in terms of the number of inputs.

Resources Used

The following resources were used in the assignment:

- Visual Studio Code(Text Editor)
- Microsoft Excel(version: 16.63.1)
- LaTeX

C Programming Language was used to code the algorithms throughout the Assignment.