ASSIGNMEMT

COURSE CODE: CSE 214

COURSE NAME: Algorithm

Submitted To

Subroto Nag Pinku

Department of CSE

Daffodil International University

Submitted By

Md. Foysal Ahmed

ID: 191-15-12486

Section: O-14

TASK 5 (Recursive Approach)

IMPLEMENTATION

```
#include <iostream>

using namespace std;

int fibo(int n)
{
   if(n<=1)
   {
      return n;
   }
   else
   {
      return fibo(n-1)+fibo(n-2);
   }

}
int main()
{
   int n,number;
   cin >> n;

   number = fibo(n);

   cout<<number<<endl;

}
```

ANALYSIS

	Cost	Time
`fib(n):`		
`if n <= 1`	c1	1
`return 1`	c2	1
`return fib(n - 1) + fib(n - 2)`	T(n-1)+T(n-2)	

For n>1 there occurs 1 comparison, 2 subtractions, 1 addition.
Therefore, the time function can be written as,
$T(n) = T(n-1) + T(n-2) + 4$ where 4 is some constant that can be replaced with c

$T(n) = T(n-1) + T(n-2) + c$
Let's try to establish a lower bound by approximating that $T(n-1) \sim T(n-2), though\, T(n-1) \geq T(n-2)$, hence lower bound

Md.Foysal Ahmed
ID:191-15-12486

$T(n) = 2T(n-2) + c$ [from the approximation T(n-1) ~ T(n-2)]

Now, $T(n-2) = 2T(n-4) + c$

Substituting $T(n)$,
$T(n) = 2 * (2T(n-4) + c) + c$
$T(n) = 4T(n-4) + 2c + c$
$T(n) = 4T(n-4) + 3c$
Again,
$T(n-4) = 4T(n-4-4) + 3c$
$T(n-4) = 4T(n-8) + 3c$
Now substitute,
$T(n) = 4 * (4T(n-8) + 3c) + 3c$
$T(n) = 16T(n-8) + 12c + 3c$
$T(n) = 16T(n-8) + 15c$
…………………………………………….
…………………………………………….
…………………………………………….
$T(n) = 2^k T(n-2k) + (2^k - 1)c$
Let's find the value of k for which: $n - 2k = 0$
$k = n/2$

Therefore,
$T(n) = 2^{\frac{n}{2}}\, T(0) + (2^{\frac{n}{2}} - 1)c$
$T(n) = 2^{\frac{n}{2}}\, T(0) + 2^{\frac{n}{2}} * c - c$
$= 2^{\frac{n}{2}}(1 + c) - c$

$i.e. T(n) = 2^{n/2}$

now for the upper bound, we can approximate $T(n-2) \sim T(n-1)\ as\ T(n-2) \le T(n-1)$
$T(n) = 2T(n-1) + c$ [from the approximation $T(n-1) \sim T(n-2)$]

Now, $T(n-1) = 2T(n-2) + c$

Substituting $T(n)$,
$T(n) = 2 * (2T(n-2) + c) + c$
$= 4T(n-2) + 3c$
$Again,$
$T(n-2) = 4T(n-4) + 3c$
Now substitute,
$T(n) = 4 * (4T(n-4) + 3c) + 3c$
$T(n) = 16T(n-4) + 15c$

Md.Foysal Ahmed
ID:191-15-12486

……………………………………………
……………………………………………
……………………………………………

$T(n) = 2^k T(n-k) + (2^k - 1)c$
Let's find the value of k for which: $n - k = 0$
$k = n$
Therefore,
$T(n) = 2^n T(0) + (2^n - 1)c$
$\quad = 2^n * (1 + c) - c$

$i.e. T(n) = 2^n$

Hence the time taken by recursive Fibonacci is $\boldsymbol{O(2^n)}$ or exponential.

Md.Foysal Ahmed
ID:191-15-12486

TASK 5 (Iterative Approach)

IMPLEMENTATION

```
#include <iostream>

using namespace std;

int fibo(int n)
{
    int i,fib[n];

    fib[0] = 0;
    fib[1] = 1;


    for(i=2; i<=n; i++)
    {
        fib[i] = fib[i-1]+fib[i-2];
    }
    return fib[n];
}
int main()
{
    int n,number;

    cin >> n;

    number = fibo(n);

    cout << number << endl;

}
```

ANALYSIS

	Cost	Time
fib(n):		
1.fib[0] <- 0	c1	1
2.fib[0] <- 1	c2	1
3.for i<-2 to n	c3	n
4. fib[i] = fib[i-1]+fib[i-2]	c4	(n-1)
5.return fib[n]	c5	1

Md.Foysal Ahmed
ID:191-15-12486

We start analyzing the Insertion Sort procedure with the time "cost" of each statement and the number of times each statement is executed.

In line 3 the loop will execute from 2 to n. Therefore, it will run for n times as the test is executed one time more than the loop body.

The running time of the algorithm is the sum of running times for each statement executed; a statement that takes c_i steps to execute and executes n times will contribute $c_i * n$ times to the total running time.

$$T(n) = c1 * 1 + c2 * 1 + c3 * n + c4 * (n - 1) + c5$$
$$T(n) = c3 * n + c4 * n - c4 + c1 + c2 + c5$$
$$T(n) = (c3 + c4)n + c1 + c2 - c4 + c5$$

This equation can be written as,

$T(n)$ = $an + b$ (where a & b are some constant) **which is linear function.**

Therefore, The time complexity of this Fibonacci Iterative method is $O(n)$.

Md.Foysal Ahmed
ID:191-15-12486

TASK 6 (Last Digit of a Large Fibonacci Number)

Implementation

```
#include <iostream>
using namespace std;

int fibo(int n)
{
   int i,fib[n];

   fib[0] = 0;
   fib[1] = 1;


   for(i=2; i<=n; i++)
   {
      fib[i] = (fib[i-1]+fib[i-2])%10 ;
   }
   return fib[n];
}

int main()
{
   int n,number;

   cin >> n;

   number = fibo(n);

   cout << number << endl;

}
```

The complexity of this program is $O(n)$ which is as same as Task 5 (Fibonacci iterative method).

Md.Foysal Ahmed
ID:191-15-12486

TASK 7 (Euclidean GCD)

IMPLEMENTATION

```
#include<bits/stdc++.h>
using namespace std;

 int gcd(int a,int b)
{
   if(b==0)
   {
      return a;
   }
   else
   {
      if(a>b)
      {
         gcd(b,a%b);
      }
      else
      {
         gcd(a,b%a);
      }
   }

}

int main()
{
   int a,b,res;
   cin >> a >> b;

   res = gcd(a,b);

   cout << res << endl;
}
```

Md.Foysal Ahmed
ID:191-15-12486

ANALYSIS

Let's consider a case where $gcd(a, b)$ is $gcd(56,21)$ where a=56, b=21

Every time we are calling the function recursively as $gcd(b, a\%b)$ and the base is if b=0 then we are returning a as the answer of $gcd(a, b)$.

The Euclidean Algorithm is working as follows:

$gcd(56,21) = gcd(21,14) = gcd(14,7) = gcd(7,0)$ here a is changing every time with the previous b as it is calling its self recursively. The result is 7 as we have reached our base case that is $b = 0$.

At each recursive step, gcd will cut one of the arguments in half (at most)

Let, $gcd(a, b) \leq T(n)$ where $n = a, b$ and d is the decreasing factor with $d = \frac{a}{a\%b}$

$$T(n) = T\left(\frac{n}{d}\right) + c$$

$$T(n) = T\left(\frac{n}{d^2}\right) + 2c$$

$$T(n) = T\left(\frac{n}{d^3}\right) + 3c$$

……………………………………

……………………………………

$$T(n) = T\left(\frac{n}{d^k}\right) + kc$$

When $\frac{n}{d^k} = 1$

$n = d^k$

$k = \log_d(n)$

Therefore,

$T(n) = T(1) + c\log_d(n)$

$T(a, b) = 1 + c\log_d(x, y)$

Therefore we can say that the time complexity of gcd$(\boldsymbol{a}, \boldsymbol{b})$ is $\boldsymbol{O(log\ n)}$.

If one number is multiple of another number, then it is the best case of GCD. Best case time complexity is $\boldsymbol{O(1)}$.

Md.Foysal Ahmed
ID:191-15-12486

TASK 8 (LCM)

Implementation

```
#include<bits/stdc++.h>

using namespace std;

 long long int gcd(int a,int b)
{
    if(b==0)
    {
        return a;
    }
    else
    {
        if(a>b)
        {
            gcd(b,a%b);
        }
        else
        {
            gcd(a,b%a);
        }
    }

}

int main()
{
   long long int a,b,res,lcm;
    cin >> a >> b;

    res = gcd(a,b);
    lcm = (a*b)/res;

    cout << lcm << endl;
}
```

The complexity of this program is $O(log\ n)$ as same as Euclidean GCD algorithm written in task 7.

Md.Foysal Ahmed
ID:191-15-12486