*Principles of Computing*  Id: 19000915

Task 5 Week 9-11 Email your work to [csct.uwe@gmail.com](mailto:csct.uwe@gmail.com) before **04-04-2020**

*Recursion and Time Complexity (ref. Handout p41-46, 67-76*) Mark /26

**This is our last worksheet. If possible, please type you answers because they are all “programming” tasks. You can write your program either in pseudocode or in a computer language of your choice. If you do use a computer language, we encourage you to test the correctness of your code and measure the execution time for comparing the efficiency.**

**Q1**. Write two different programs to compute ***mod m*** ).

Note that you are not allowed to use “pow” function from library

(a). powmod1()

It computes by repeated multiplications, i.e. using iteration. [5 marks]

(b). powmod2*(x, n, m)*

It computes by using the following inductive definition [5 marks]

if is even

if is odd

Discuss which the above program is more efficient using big-O notation. If you are not sure about which one is more efficient, test your code on computer, using the following data. Present the timing result to support your conclusion.

For x=29, m = 773, use n = 100,000,000, 500,000,000 and 1,000,000,000 [3 marks]

**Q2**. The ***Fibonacci*** numbers are the numbers in the sequence 1, 1, 2, 3, 5, 8, 13, 21 …, where each number after the first two is computed by adding the preceding two numbers.

Write two different programs fib1(n) and fib2(n) to compute the *n*-th Fibonacci number.

(a). fib1(n) computes the *n*-th Fibonacci number using iteration (i.e. loop). [5 marks]

(b). fib2(n) computes the *n*-th Fibonacci number using induction (i.e. recursion) [5 marks]

Same as the previous question, you need to discuss which the above program is more efficient using big-O notation. If you are not so sure, test your code on computer, using the following data. Present your timing results to support your conclusion.

n = 44, 45, and 46 [3 marks]

**Appendix – Test Data and the Answers**

**29 100000000 mod 773 = 349, 29 500000000 mod 773 = 16, 29 1000000000 mod 773 = 256**

**fib(44) = 701408733, fib(45) = 1134903170, fib(46) = 1836311903**

**Q1.**

**(a). powmod1(x,n,m)**

**int** powmod1(int x, int n, int m)

{

**int** res;

res=1;// Initialize result

x=x%m;// Update x if it is more than or equal to p

cout<<"We'll perform x^n%m using a while-loop"<<endl;

**while**(n>0)

{

// If y is odd, multiply x with result

**if**(n%2==1)

res=(res\*x)%m;

n=n/2;//n is even now

x=(x\*x)%m;

}

**return** res;

}

**int** main()

{

**int** x,n,res,m;

**cout<<**"x=";

**cin>>**x;

**cout<<**"n=";

**cin>>**n;

**cout<<**"m=";

**cin>>**m;

**cout<<**powmod1(x,n,m);

**return** 0;

}

**(b). powmod2(x,n,m)**

**int** powmod2(int x,int n,int m)

{

**if**(n==0)

**return** 1;

**else if**(n%2==0)

**return** (mod(x,n/2,m)\*mod(x,n/2,m))%m;

**else**

**return** ((x%m)\*mod(x,n-1,m))%m;

}

**int** main()

{

**int** x,n,res,m;

**cout<<**"x=";

**cin>>**x;

**cout<<**"n=";

**cin>>**n;

**cout<<**"m=";

**cin>>**m;

**cout<<**powmod2(x,n,m);

**return** 0;

}

**For Q1 (a).**

* **29 100000000 mod 773 = 349 - execution time : 11.043 s**
* **29 500000000 mod 773 = 16 - execution time : 4.866 s**
* **29 1000000000 mod 773 = 256 - execution time : 13.898 s**
* **The complexity of this algorithm is O(log(n)2).**

**For Q1 (b).**

* **29 100000000 mod 773 = 349 - execution time : 7.468 s**
* **29 500000000 mod 773 = 16 - execution time : 12.335 s**
* **29 1000000000 mod 773 = 256 - execution time : 21.434 s**
* **The complexity of this algorithm is O(log(n))**

**The 1st algorithm is more efficient deducing from the data presented above.**

**Q2.**

**(a). fib1(n)**

**int** fib1(**int** n)

**{**

**int** previouspreviousNumber,previousNumber=0,currentNumber=1;

**for(int** i=1;i<n;i++)

{

previouspreviousNumber=previousNumber;

previousNumber=currentNumber;

currentNumber=previouspreviousNumber+previousNumber;

}

**return** currentNumber;

}

**int** main()

{

**int** n;

**cout<<**"n=";

**cin>>**n;

**cout<<**fib1(n);

**return** 0;

}

**(b). fib2(n)**

**int** fib2(**int** n)

**{**

**if**(n==0)

**return** 0;

**if**(n==1)

**return** 1;

**return** fib2(n-1)+fib2(n-2);

}

**int** main()

{

**int** n;

**cout<<**"n=";

**cin>>**n;

**cout<<**fib2(n);

**return** 0;

}

**For Q2 (a).**

* **fib(44) = 701408733 - execution time : 0.945 s**
* **fib(45) = 1134903170 - execution time : 1.046 s**
* **fib(46) = 1836311903 - execution time : 1.168 s**
* **The complexity of this algorithm is O(n).**

**For Q2 (b).**

* **fib(44) = 701408733 - execution time : 6.855 s**
* **fib(45) = 1134903170 - execution time : 10.096 s**
* **fib(46) = 1836311903 - execution time : 16.090 s**
* **The reason for the poor performance is heavy push-pop of the stack memory in each recursive call.**

**So , the 1st algorithm (fib1) is more efficient.**