Project

#1) **Problem**: Write an iterative C++ function that inputs a non negative integer n and returns the nth Fibonacci number.

**Source Code:**

#include <iostream>

using namespace std;

int fibonacci ( int x) {

int i = 0, j = 1, k = 1, sum=0;

while (i < x-1){

sum=j+k;

j=k;

k=sum;

i++;

}

return j;

}

int main()

{

int number;

clock\_t t;

t = clock();

cout << "Enter the Fibonacci Term: " ;

cin >> number;

cout << "The " << number << "- th Fibonacci Number is: " << endl<< fibonacci(number) << endl ;

t = clock()-t;

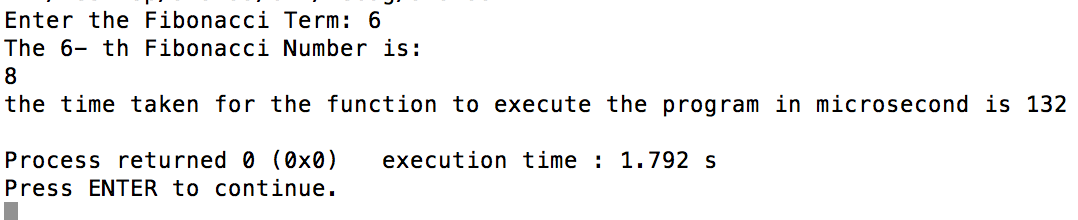
double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

cout<<"the time taken for the function to execute the program in microsecond is " <<time\_taken\*1000000<< endl;

return 0;

}

**Output:**

****

#2) **Problem**: Write a recursive C++ function that inputs a nonnegative integer n and returns the nth Fibonacci number.

**Source Code:**

#include <iostream>

using namespace std;

int fibonacci ( int x) {

if (x==1 || x==2)

return 1;

else

return fibonacci(x-1) +fibonacci(x-2);

}

int main()

{

int number;

clock\_t t;

t = clock();

cout << "Enter the Fibonacci Term: " ;

cin >> number;

cout << "The " << number << "- th Fibonacci Number is: " << endl<< fibonacci(number) << endl ;

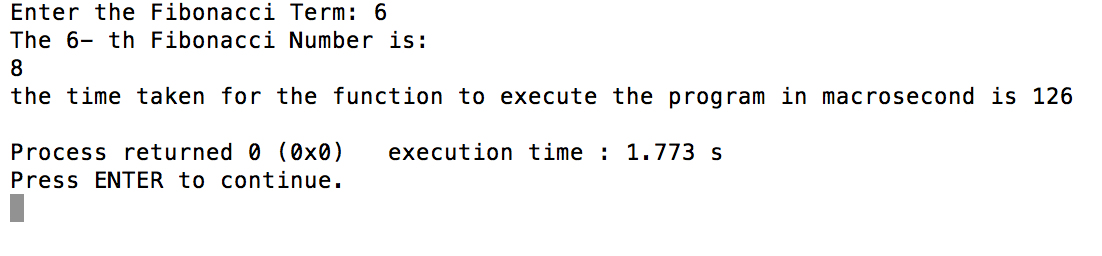
t = clock()-t;

double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

cout<<"the time taken for the function to execute the program in macrosecond is " << time\_taken\*1000000 << endl;

return 0;

}

**Output:**

**Conclusion/ reflection:**

I didn't face much difficulties while writing the code for both problems (1 &2 ) because I had practiced this program a lot in my mac 101 class. But Since, I wanted the to show the time taken for the execution of the program, I did get some syntax errors when I was trying to write the code. I also got the but put in the format like 0.000132 seconds, so I converted it into microseconds by multiplying it by 1000000. Rather than that, it wasn't that challenging for me.

**#3) Problem:** Compare the number of operations and time taken to compute Fibonacci numbers recursively versus that needed to compute them iteratively.

**Source Code:**

#include<iostream>

#include <time.h>

using namespace std;

//two variables are defined to store the number of operations for Iterative and Recursive call of Fibonacci

int Recursive\_Operation = 0;

int Iterative\_Operation = 0;

// recursive function to compute fibonacci number

int fibRec(int n)

{

Recursive\_Operation++; // increasing count

if (n <= 1)

return n;

return fibRec(n-1) + fibRec(n-2);

}

// iterative function to compute fibonacci number

int fibIter(int n)

{

int a = 0, b = 1, c, i;

Iterative\_Operation++;

if( n == 0)

return a;

for (i = 2; i <= n; i++)

{

Iterative\_Operation++;

c = a + b;

a = b;

b = c;

}

return b;

}

int main ()

{

int n = 6;

// calling recursive method

int f1 = fibRec(6);

cout<<"Number of operation taken by recursive method : "<<Recursive\_Operation<<endl;

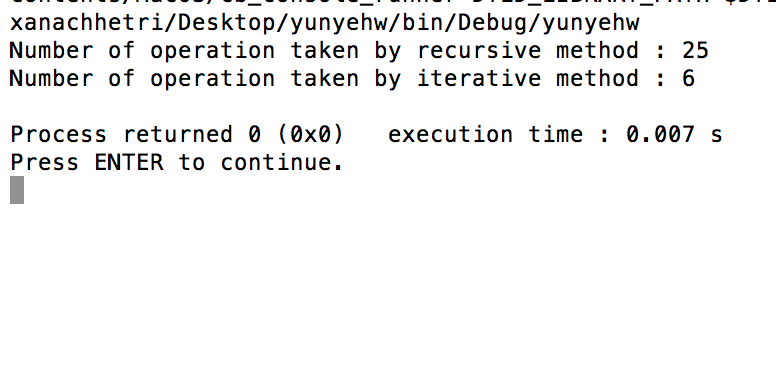
// calling iterative method

int f2= fibIter(6);

cout<<"Number of operation taken by iterative method : "<<Iterative\_Operation<<endl;

return 0;

}



**Output:**

**Conclusion:**

I found this problem little bit challenging for me. I was getting so many errors when I wrote this program because I forgot to call the recursive and iterative method due to which it wasn't showing the output at all. But later I declared variable n=6 with int data type and fixed the problem.

When comparing the time and number of operation take by recursive and iterative method to find f(6) i.e. Fibonacci number 6, I found that iterative method gave output in 132 macro second while that of recursive gave output in 126 macro seconds. And, also number of operation taken by recursive method is 25 while that of iterative is 6. This shows that the number of iterations required in recursive function is more than compared to iterative calls.Also the speed of recursive function is slower than iterative approach and reason is in first approach we are calling function which means keeping the current value & parameters in stack and once function call is done then return values is pushed to stack and calling function gets its state by loading state and its value from stack. Lots of store and load instructions are required.

**#4) I. Problem:** Find the exact value of f100, f500, and f1000, where fn is the nth Fibonacci number. What are times taken to find out the exact values?

**Source code:**

#include <iostream>

using namespace std;

double fibonacci ( double x) {

double i = 0, j = 1, k = 1, sum=0;

while (i < x-1){

sum=j+k;

j=k;

k=sum;

i++;

}

return j;

}

int main()

{

double number;

clock\_t t;

t = clock();

cout << "Enter the Fibonacci Term: " ;

cin >> number;

cout << "The " << number << "- th Fibonacci Number is: " << endl<< fibonacci(number) << endl ;

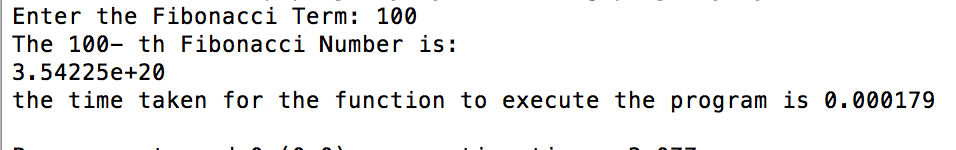
t = clock()-t;

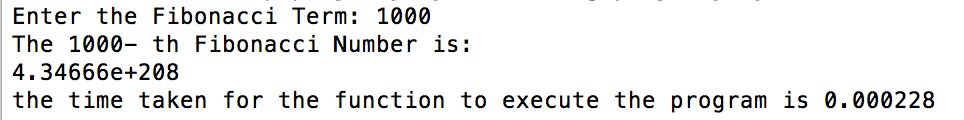
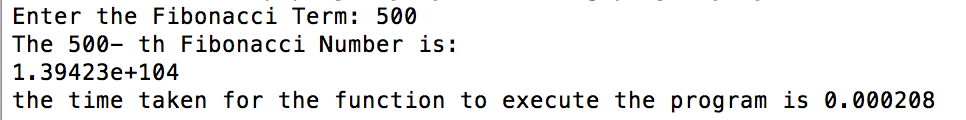
double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

cout<<"the time taken for the function to execute the program is " << time\_taken<< endl;

return 0;

}

**Output:**



**Conclusion:**

This problem was similar to the first problem due to which I thought I would get the correct result but every time I inputted the number greater than 50, the output came in negative and gave the wrong answer. Later, I figured out that I used the data type int due to which I was getting the errors. So, I changed it to double and finally got correct answer.

From the output, I came to conclusion that number of operation is directly proportional to time taken. It is because the time taken to execute the 1000th Fibonacci number is 0.000228\*1000000= 228 micro seconds which is more than that of 500 (i.e., 208) and that of 100(i.e. 179 micro second).

**4) II. Problem:** Find the smallest Fibonacci number (1) greater than 1,000,000

**Source code:**

#include <iostream>

using namespace std;

int main()

{

int j = 0, k = 1, sum=0;

while (sum < 1000000){

sum=j+k;

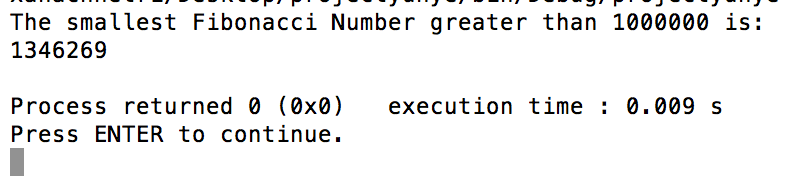
j=k;

k=sum;

}

cout << "The smallest Fibonacci Number greater than 1000000 is: " << endl<< sum << endl ;

}

**Output:**

**and (2) greater than 1,000,000,000.**

**Source code:**

#include <iostream>

using namespace std;

int main()

{

int j = 0, k = 1, sum=0;

while (sum < 1000000000){

sum=j+k;

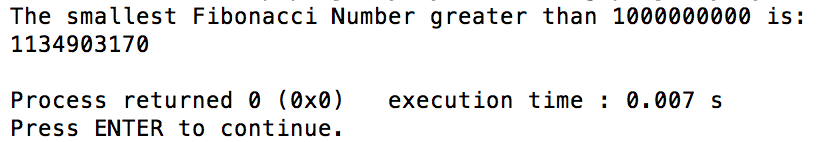
j=k;

k=sum;

}

cout << "The smallest Fibonacci Number greater than 1000000000 is: " << endl<< sum << endl ;

}

**Output:**

**Conclusion:**

This program was bit tricky for me. It is because at first I tried to find the Fibonacci number greater than 1000000 and 1000000000 by using the code of the first problem. I kept on inputting the number greater than 1000000 and 1000000000 one by one and it was tedious and not working at all. But later, I realized that I have to change the code and I could finally get the desired output. Rather than that, I didn't get much errors.

**#4) III. Problem:** Find as many prime Fibonacci numbers as you can. It is unknown whether there are infinitely many of these. Find out the times taken to find first 10, 20, 30, 40...up to 200 and draw a graph and see the pattern.

**Source Code:**

#include <iostream>

#include<cmath>

using namespace std;

bool prime(int sum);

double fibonacci ( double x) {

int i = 0;

double j = 1, k = 1, sum = 1;//the next fib. sum=j+k

while(i!=x){

sum = j + k;

j = k;

k = sum;

while (true) {

if (prime(sum)){

i= i+1;

cout<< sum<< endl;

break;

}

else

break;

}

}

return k;

}

bool prime(int sum) {

int i;

for (i = 2; i <=(sqrt(sum)); i++) {

if (sum % i == 0) // If i divides n evenly,

return false; // n is not a prime.

}

return true; // If no divisor found, n is prime.

}

int main()

{

double number;

clock\_t t;

t = clock();

cout << "Enter the Fibonacci Term: " ;

cin >> number;

cout << "The " << number << "- th Fibonacci Number is: " << endl<< fibonacci(number) << endl ;

t = clock()-t;

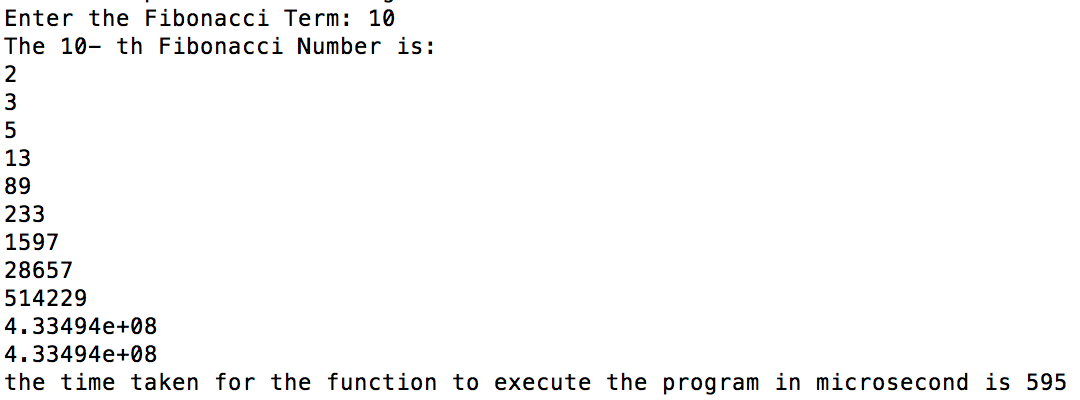
double time\_taken = ((double)t)/CLOCKS\_PER\_SEC;

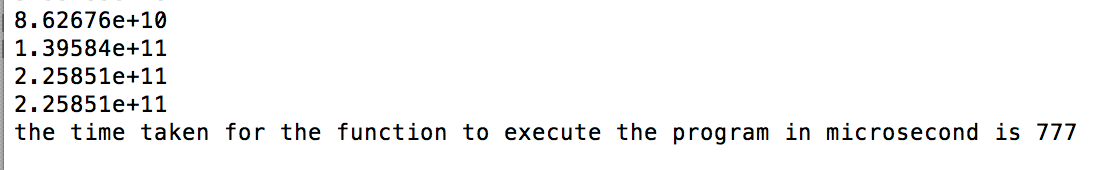
cout<<"the time taken for the function to execute the program in microsecond is " << time\_taken\*1000000<< endl;

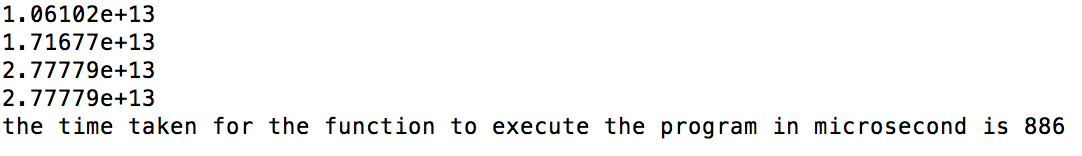
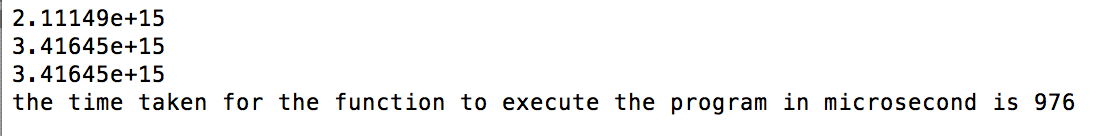
return 0;

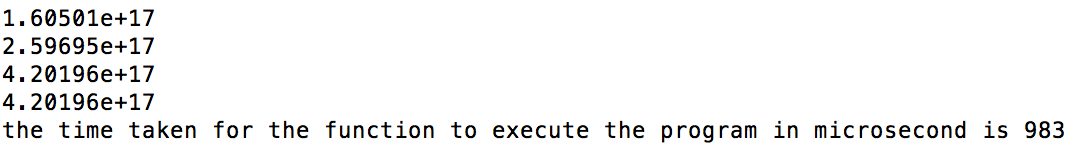
}

**Output:**

**first 10:**

**first 20:**

**first 30:**  **first 40:**

**first 50:**

**Table for time taken f**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| f(n) | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |
| time taken (µs) | 595 | 777 | 886 | 976 | 983 | 1012 | 1086 | 1125 | 1133 | 1164 |
| f(n) | 110 | 120 | 130 | 140 | 150 | 160 | 170 | 180 | 190 | 200 |
| time taken (µs) | 1198 | 1247 | 1265 | 1289 | 1302 | 1388 | 1398 | 1431 | 1455 | 1474 |

**or computing Fibonacci prime numbers Conclusion:**

**time in microseconds (µs)**(µs)

**f(n)**

I struggled a lot while writing this program because I didn't write the boolean part first and tried to find the prime numbers by different method. But it didn't work out. Therefore, I got help from the book and figured out the code for how to find prime numbers and used it here. And also, the time taken for the execution of the program was coming in huge decimal numbers due to which I multiplied the seconds by 1000000 and turned it into micro seconds. The graph and the table shows that the time for the execution of the program is increasing when the f(n) is increasing. Hence, F(n) and time in microseconds is directly promotional.

**2)**

Fibonacci number is the series of numbers which is formed by adding up the two numbers before it. Fibonacci numbers were first introduced in 1202. There are many different applications of Fibonacci numbers to sciences and real word. One of the amazing mathematical problems Fibonacci investigated in Liber Abaci was about how fast rabbits could breed in ideal lifestyle. Lets suppose a new born pair of rabbits, one male and one female are put in a field. Rabbits are usually able to mate at the age of one month and the end of its second month a female can produce another pair of rabbits. Lets assume that our rabbits never die and that the female always produces one new pair (one male and one female) every month from the second month on. The question that was running on the Fibonacci’s mind was, how many pairs will their be in one year?

He observed that rabbits mated at the end of the first month and there was still only 1 pair.

Then, female produces a new pair at the end of the second month. Hence, now there are 2 pair of rabbits now. Again, at the end of the third month, the original female again produces a second pair of rabbits making 3 pairs in all. Similarly, the original female rabbit produced another new pair at the end of the fourth month. And, the female born two months ago also produced her first pair making total 5 pairs of rabbits

Hence, if we do it mathematically, let there are  pairs of rabbits after  months. The number of pairs in month  will be  (since rabbits never die in this problem) plus the number of new pairs are also born. But new pairs are only born to pairs after at least 1 month. Therefore, there will be  new pairs. So we have



 which is simply the rule for generating the Fibonacci numbers.

The fibonacci numbers not only occur in rabbits but it also appears in plants and flowers. Botanists have shown that flowers often have a Fibonacci number of petals. For example, lets consider a sunflower plant. If we carefully watch the picture of the sunflower below and count those curves of seeds spiraling to the left as you go outwards, we will see that there are 55 spirals. At the same point, we can also see that there are 34 spirals of seeds spiraling to the right.Similarly, a little bit further towards the centre, we can count 34 spirals to the left and 21 spirals to the right. Hence, the pair of numbers (counting spirals curving left and curving right) are numbers of the Fibonacci series.



Also, many plants show the Fibonacci numbers in the arrangements of the leaves around their stems. For example, in the plant in the picture above, we have 3 clockwise rotations before we meet a leaf directly above the first, passing 5 leaves on the way. If we go anti-clockwise, we need only 2 turns. Notice that 2, 3 and 5 are consecutive Fibonacci numbers.

For the lower plant in the picture, we have 5 clockwise rotations passing 8 leaves, or just 3 rotations in the anti-clockwise direction. This time 3, 5 and 8 are consecutive numbers in the Fibonacci sequence.

Fibonacci numbers can be used in [Fibonacci heaps](http://en.wikipedia.org/wiki/Fibonacci_heap), which is a data structure that can be used to speed up some very practical algorithms. A real life problem where I think I can apply the concept of Fibonacci numbers is it can be used for searching a sorted list of entries in an array to find a particular one. People usually use the [Binary search algorithm](http://en.wikipedia.org/wiki/Binary_search_algorithm) but that is optimal assuming that each time you look at an array element it takes the same amount of time. Hence, I think [Fibonacci search](http://en.wikipedia.org/wiki/Fibonacci_search_technique) works better. This can also apply to situations like searching through arrays that don't fit entirely in our computer's cache.