Converting Fibonacci To c++

Yhaliff Said Barraza

06/Mayo/2019

1 Fibonacci sequence

Here is 1 of the formulas for finding a number in the **Fibonacci sequence**

$$F(X) = \begin{cases} F(X)_{x-1} + F(X)_{x-2} & n > 1 \\ F(0) = 0 \\ F(1) = 0 \\ F(2) = 1 \end{cases}$$

Now lets convert this to **C++**, First will use recursion to find the number X

1.1 Code Recursion Fibonacci

```
int RecursiveFibonacci(int Index)
{
  //! this is for handling exceptions
  if (Index == 0) { return 0; }

  if (Index == 1)
  {
    return 0;
  }
  else if (Index == 2) {
    return 1;
  }
  return RecursiveFibonacci(Index - 1) + RecursiveFibonacci(Index - 2);
}
```

1.2 Code Iterative Fibonacci

The code above this subsection is just one of many way to find a Fibonacci number, let's showcase how to get fibonacci through Iteration .

Note "n" means input

Algorithm 1 Iterative Fibonacci

```
1: FirstValue = 0
2: SecondsValue = 1
3: Result = 0
4:
5: if n == 0 Or n == 1 then
6:     Return 0
7: end if
8: if n == 2 then
9:     Return 1
10: end if
11:
12: for i < n − 2 do
13:     Result = FirstValue + SecondsValue
14:     FirstValue = SecondValue
15:     SecondValue = Result
16: end for
17: Return Result
```

All that you above you is called *pseudo-code* ,it's a way represent how to program something (algorithms in this case)in such a way that it doesn't matter what programming language your using you can follow along.

Now here is the implementation in **C++**

```
int NonRecursiveFibonacci(int Index) {
  //! this are just the first 2 values of Fibonacci
  int FirstValue = 0;
  int SecondValue = 1;

  int Result = 0;

  if (Index == 1) { return FirstValue; }
  if (Index == 2) { return SecondValue; }

  // I'm subtracting 2 from index because it would give me
  // a correct number, but it would be 2 ahead of the recursive function
  for (int i = 0; i < Index - 2; ++i) {
```

```
    Result = FirstValue + SecondValue;
    FirstValue = SecondValue;
    SecondValue = Result;
  }

  return Result;
}
```

2 Benchmark

Now I have a question for you *Which of the 2 functions is faster?*
Answer : I don't know but, There is a way to Find out
Benchmarking the 2 and making a graph because yes

Now I have another question (yes I do have the answer this time) *How do we benchmark the two Fibonacci functions?*
Answer : Make a function that does that for us plus writes the result to a file .
Here is one of those functions

```
void BenchmarkIterativeFibonacci() {

  // increases in units of 1
  int SimulatedInput = 0;

  std::ofstream OutputFile;
  OutputFile.open("Iterative Times.txt", std::ios::trunc);

  Timer TimerIterative;

  /*!Testing value from 1 to 40 */
  for (int i = 0; i < 400; ++i) {

    if (i % 10 == 0) {
      SimulatedInput++;
      OutputFile << '\n';
    }

    TimerIterative.StartTiming();
    NonRecursiveFibonacci(SimulatedInput);
    TimerIterative.EndTiming();

    OutputFile << TimerIterative.GetResult() << '\t';
  }
```

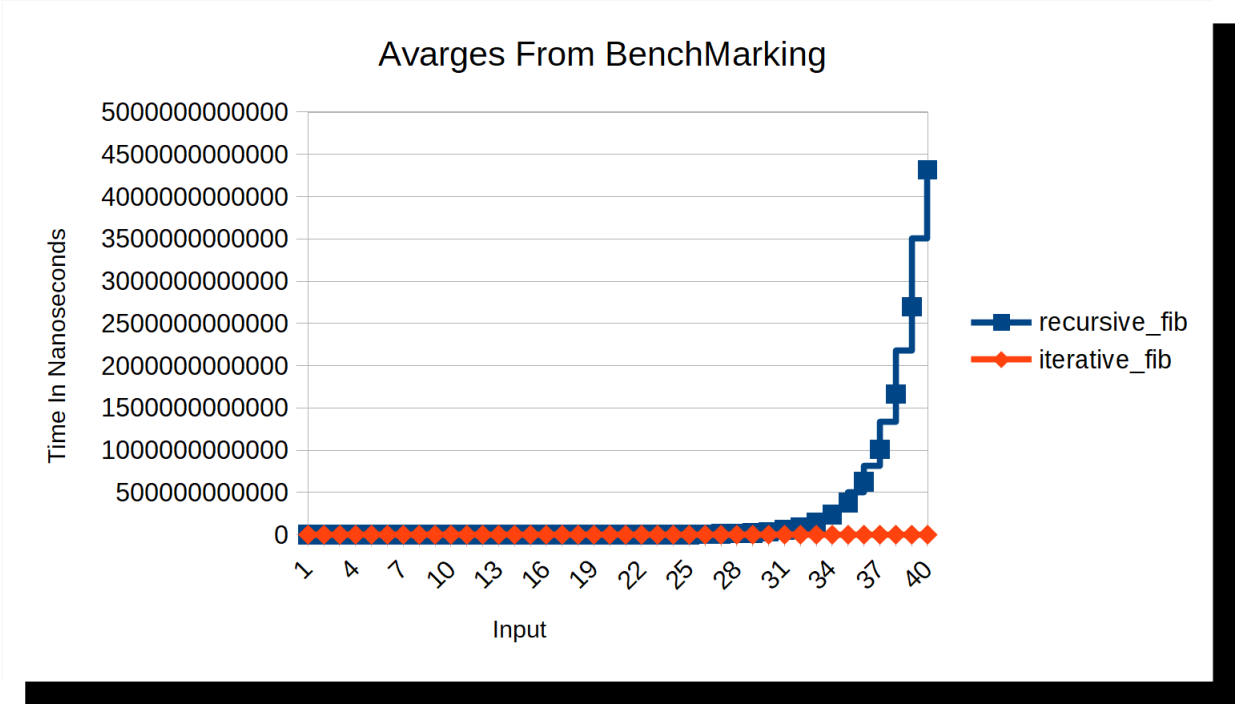
```
  std::cout << "Here is the Simulated INput [" << SimulatedInput << ']';
  OutputFile.close();
}
```

That will get the time for the Iterative Function but, we need to also benchmark the Recursive function too.
Here the code for that.

```
void BenchmarkRecursiveFibonacci() {

  // increases in units of 1
  int SimulatedInput = 0;

  std::ofstream OutputFile;
  OutputFile.open("Recursive Times.txt", std::ios::trunc);

  Timer TimerIterative;

  /*!Testing value from 1 to 40 */
  for (int i = 0; i < 400; ++i) {

    if (i % 10 == 0) {
      SimulatedInput++;
      OutputFile << '\n';
    }
    TimerIterative.StartTiming();
    RecursiveFibonacci(SimulatedInput);
    TimerIterative.EndTiming();

    OutputFile << TimerIterative.GetResult() << '\t';
  }
  std::cout << "Here is the Simulated INput [" << SimulatedInput << ']';
  OutputFile.close();
}
```

3 Graph

Now that we have the data in a text file we just need to graph the data.


Avarges From BenchMarking
Time In Nanoseconds
5000000000000
4500000000000
4000000000000
3500000000000
3000000000000
2500000000000
2000000000000
1500000000000
1000000000000
500000000000
0
1
4
7
10
13
16
19
22
25
28
31
34
37
40
Input
recursive_fib
iterative_fib