Illinois Institute of Technology

Department of Computer Science

CS430 - Section: 04 (live)

Introduction to Algorithms

IIT Assignment 3 - Report

Authors:
Bastien LEDUC
Ashik SYED SHAFFIULLAH
Kachikwu NWIKE
Hongbo WANG

Student Number:
A20520860
A20516459
A20517226
A20524770

November 19, 2022

Instructions

1. The Fibonacci sequence *0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...* is defined for $n \geqslant 2$ by the recurrence $F(n) = F(n-1) + F(n-2)$ with $F(0) = 0$ and $F(1) = 1$.

   (a) **(2 points)** Calculate recursively $F(n)$ by applying the top-down procedure $Fib()$ which calls itself. Use your favorite programming language to write $Fib()$. Test $Fib()$ for different numerical values of n to make sure that $Fib(n)$ correctly calculates $F(n)$. You can use the formula for $F(n)$ given in item **(f)** below to check your numerical results.

   (b) **(2 points)** Draw the recursion tree for $Fib(5)$. Let's denote by $T(n)$ the total number of calls to $Fib()$ needed to calculate $F(n)$ (note that $T(n)$ includes the original call to $Fib(n)$ at the root). What is the value of $T(5)$?

   (c) **(4 points)** What is the recurrence for $T(n)$? What are the initial conditions of this recurrence?

   (d) **(2 points)** Use $Fib()$ to empirically verify that $T(n) = 2F(n+1) - 1$.

   (e) **(4 points)** Given the recurrence for $T(n)$ found in item **(c)** above, prove by induction that the formula $T(n) = 2F(n+1) - 1$ is correct.

   (f) **(2 points)** Use the formula given in item **(d)** to find the asymptotic complexity $T(n) = O(?)$ of $Fib()$. The closed-form expression for the Fibonacci numbers $F(n) = \dfrac{\phi^n - \psi^n}{\sqrt{5}}$ where $\phi = \dfrac{1+\sqrt{5}}{2}$ and $\psi = \dfrac{1-\sqrt{5}}{2}$, may be helpful.

   (g) **(2 points)** Implement in your favorite programming language the bottom-up iterative procedure $Better\_Fib()$ with "memoization" for better performance. Test $Better\_Fib()$ for different numerical values of n to make sure that it correctly calculates $F(n)$. You can use the formula for $F(n)$ given in item **(f)** above to check your numerical results.

   (h) **(1 points)** What is the asymptotic complexity $T_{BF}(n) = O(?)$ of $Better\_Fib()$?

   Submit source codes of all the programs you need to produce your results.

Solutions

1. (a) *(See details in the Jupyter notebook).*

N	**F(n)**	**Fib(n)**
0	0	0
1	1	1
2	1	1
3	2	2
4	3	3
5	5	5
6	8	8
7	13	13
8	21	21
9	34	34
10	55	55
11	89	89
12	144	144
13	233	233

Table 1: Fib(n) vs F(n)

(b) Draw the recursion tree for $Fib(5)$:

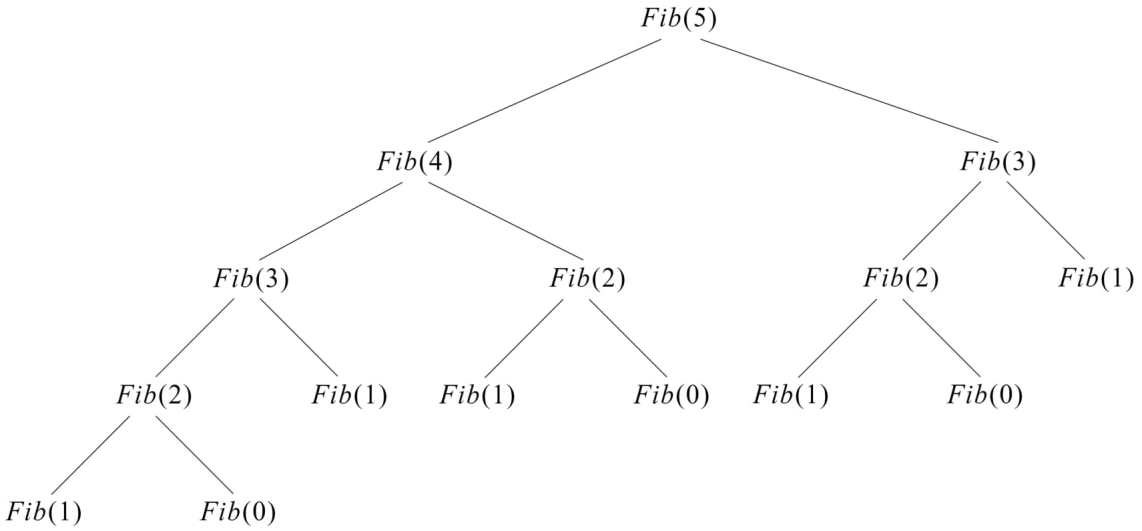

Figure 1: Fibonacci's recursion tree for $Fib(5)$.

By using a counter to count the total number of calls $T(n)$ of the $Fib()$ function, we obtained for $Fib(5)$: $T(5) = 15$, which is the number of nodes in the recursive tree for $Fib(5)$.

(c) The recurrence for $T(n)$ is:

$$T(n) = \underbrace{T(n-1)}_{\#\text{ of nodes in LST}} + \underbrace{T(n-2)}_{\#\text{ of nodes in RST}} + \underbrace{1}_{\text{root node}}$$

where the initial conditions from the recurrence diagram are $T(0) = 1$ and $T(1) = 1$. Starting with $n = 2$:

$$\begin{aligned} T(2) &= T(1) + T(0) + 1 = 3 \\ T(3) &= T(2) + T(1) + 1 = 5 \\ T(4) &= T(3) + T(2) + 1 = 9 \\ T(5) &= T(4) + T(3) + 1 = 15 \\ &\vdots \\ \text{Therefore: } T(n) &= T(n-1) + T(n-2) + 1 \end{aligned}$$

(d) *(See details in the Jupyter notebook).*

N	**T(n)_recursion**	**T(n)_closedForm**
0	1	1
1	1	1
2	3	3
3	5	5
4	9	9
5	15	15
6	25	25
7	41	41
8	67	67
9	109	109
10	177	177

Table 2: T(n) recursion vs. T(n) closed-form

(e) We need to prove:

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

using the recurrence:

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

with base cases $T(0) = 1$ and $T(1) = 1$.

Solution:

- Base case:

  For $n = 0$ and $n = 1$ (see Table 1):

$$\begin{aligned} T(0) &= 2F(0+1) - 1 \\ &= 2F(1) - 1 \\ &= 2 \times 1 - 1 \\ &= 1 \\ T(1) &= 2F(1+1) - 1 \\ &= 2F(2) - 1 \\ &= 2 \times 1 - 1 \\ &= 1 \end{aligned}$$

- Induction method:

  We assume that the formula is true for $T(n-1)$ and $T(n-2)$, let's prove it for $T(n)$:

$$\begin{aligned} T(n) &= T(n-1) + T(n-2) + 1 \\ &= (2F(n) - 1) + (2F(n-1) - 1) + 1 \\ &= 2F(n) + 2F(n-1) - 1 \\ &= 2(F(n) + F(n-1)) - 1 \\ &= 2(F(n+1)) - 1 \\ &= \mathbf{2F(n+1) - 1} \end{aligned}$$

  Therefore, $\mathbf{T(n) = 2F(n+1) - 1}$.

(f) Use the formula given in item **(d)** to find the asymptotic complexity $T(n) = O(?)$ of $Fib()$.

The closed-form Fibonacci expression is given by:

$$F(n) = \frac{\phi^n - \psi^n}{\sqrt{5}} = \frac{\phi^n}{\sqrt{5}} - \frac{\psi^n}{\sqrt{5}}$$

where $\phi = \dfrac{1+\sqrt{5}}{2}$ and $\psi = \dfrac{1-\sqrt{5}}{2}$.
Since $|\psi| < 1$, we have:

$$\frac{|\psi^n|}{\sqrt{5}} < \frac{1}{\sqrt{5}} < \frac{1}{2}$$

So $\dfrac{|\psi^n|}{\sqrt{5}}$ is small and can be neglected as we know that $F(n)$ is going to be an integer. Therefore, $\dfrac{\phi^n}{\sqrt{5}}$ is closest to $F(n)$, which means that when we round $\dfrac{\phi^n}{\sqrt{5}}$ to the nearest integer, we get $F(n)$. To do that, we round it upwards if it is a half-integer. Therefore, $\left\lfloor \dfrac{\phi^n}{\sqrt{5}} + \dfrac{1}{2} \right\rfloor$ is one way of expressing $F(n)$.
We now have:

$$F(n) = \left\lfloor \frac{\phi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor$$

$$\begin{aligned}
T(n) &= 2F(n+1) - 1 \\
&= 2\left\lfloor \frac{\phi^{n+1}}{\sqrt{5}} + \frac{1}{2} \right\rfloor - 1 \\
T(n) &\leqslant \frac{2}{\sqrt{5}}\phi^{n+1} \\
T(n) &\leqslant \frac{2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{n+1} \\
T(n) &\leqslant \frac{2}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)\left(\frac{1+\sqrt{5}}{2}\right)^{n} \\
T(n) &\leqslant \left(\frac{1+\sqrt{5}}{\sqrt{5}}\right)\left(\frac{1+\sqrt{5}}{2}\right)^{n} \\
T(n) &= \mathbf{1.447}\phi^{\mathbf{n}}
\end{aligned}$$

where $\mathbf{0 \leqslant T(n) \leqslant 1.447}\phi^{\mathbf{n}} \quad \forall\, n \geqslant n_0$ where $\mathbf{c = 1.447}$ and $\mathbf{g(n)} = \left(\dfrac{\mathbf{1+\sqrt{5}}}{\mathbf{2}}\right)^{\mathbf{n}}$.

Therefore, $\mathbf{T(n) = O}(\phi^{\mathbf{n}})$.

(g) The bottom-up approach uses the Dynamic Programming. In here, it recognizes the order in which we solved the sub-problems (functions). This allows us to compute the solution for each function only once, rather than calling it multiple times, like we do in the recursive approach. The advantage of this bottom-up approach is less memory space consumption in our script.

In bottom-up approach we calculate the Fibonacci number in order until we reach $F(n)$ and we use two temporary variables to store the intermediate values.

N	F(n)	Better_Fib(n)
0	0	0
1	1	1
2	1	1
3	2	2
4	3	3
5	5	5
6	8	8
7	13	13
8	21	21
9	34	34
10	55	55
11	89	89
12	144	144
13	233	233

Table 3: F(n) vs Better_Fib(n)

(h) Implementation of `Better_Fib()`:

```
1   # Implement the Bottom-up approach
2   # of Fibonacci called Better_Fib(n)
3   def Better_Fib(n):                                  # Cost      Times
4       # Error handling
5       if n == 0:                                      # c1          1
6           return 0                                    # c2          1
7       if n == 1:                                      # c3          1
8           return 1                                    # c4          1
9       # Initialize the cache
10      # with the initial values
11      # for memoization where memo[0]
12      # and memo[1] are the results for
13      # Fib(0) and Fib(1) respectively.
14      memo = [0]*(n+1)                                # c5          1
15      memo[0] = 0                                     # c6          1
16      memo[1] = 1                                     # c7          1
17      for i in range(2, n+1):                         # c8          n
18          memo[i] = memo[i-1] + memo[i-2]             # c9          n-1
19      return memo[n]                                  # c10         1
```

For the time complexity of `Better_Fib()`:

$$\begin{aligned} T_{BF}(n) &= c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 + c_8 \times n + c_9 \times (n-1) + c_{10} \\ &= n \times (c_8 + c_9) + (c_1 + c_2 + c_3 + c_4 + c_5 + c_6 + c_7 - c_9 + c_{10}) \end{aligned}$$

We can express this running time as $an + b$ for constants a and b that depend on the statement costs c_i. It is thus a linear function of n. Therefore, $\mathbf{T_{BF}(n) = O(n)}$.

Team contribution

Bastien: (25%) coding parts, algorithms, plots, report.

Ashik: (25%) algorithms, mathematical proofs, report.

Kachikwu: (25%) mathematical induction proof, asymptotic complexity.

Hongbo: (25%) Fibonacci recursive tree, algorithms, report.