Applied R Programming 2022

-- Nikhil Vidhani
Finance and Accounting,
IIM Bangalore

Course Objective

- At the end of 7 weeks:
  - You should be able to perform basic programming tasks in R
    - Irrespective whether they are related to data analysis or your work
    - Appreciate programming as a means to accomplish huge no. of smaller tasks
    - Build complex logic and work-flow through small and concise functions
  - Have a good understanding of how to approach an empirical project
    - Data sources, merging, cleaning etc
  - Perform data analysis
    - Summaries, plots, regressions
    - Scale up your project

- What should you do?
  - Practice, practice and practice. There is no other way to learn programming.
    - Programming nuances
    - Experiment

Course Outline

- Module I
  - Basics of Computer Architecture and Programming
  - Intro to Programming through R
  - Introduction to data.frame
  - Functions
  - useful R methods
  - Loop Functions

- Module II
  - Introduction to *data.table* R package: syntax, usage and benefits
  - Merging datasets
  - Long form and wide form
  - Plotting in R

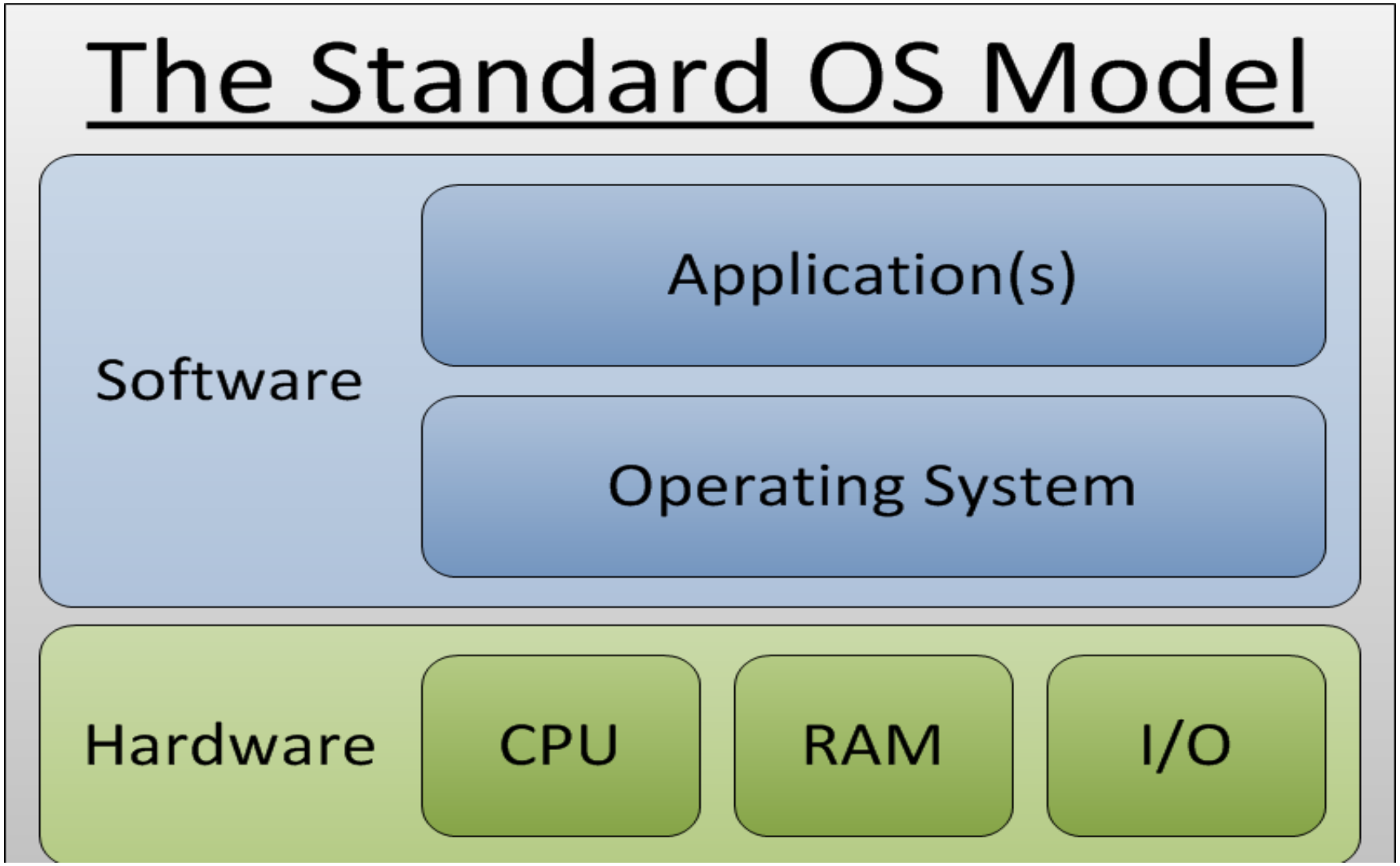
- Module III
  - Introduction to Data Analysis
    - Steps in a Data analysis project
    - Nuances: missing values, repeating data and extremes

Session-1

Module - I

- Basics of Computer Architecture and Programming
- Intro to Programming through R
- Introduction to data.frame
- Functions
- useful R methods
- Loop Functions

What's a computer look like?



What does a Computer do?

What does a Computer do?

- Perform Calculations!
  - Billions of them every second.
  - Cores, threads, clock speed

What does a Computer do?

- Perform Calculations!
  - Billions of them every second.
  - Cores, threads, clock speed
- Stores data
  - Cache vs RAM vs HDD
    - SSD
  - Speed vs storage cost

What does a Computer do?

- Perform Calculations!
  - Billions of them every second.
  - Cores, threads, clock speed
- Stores data
  - Cache vs RAM vs HDD
    - SSD
  - Speed vs storage cost
- Runs Software
  - System (OS): Linux, Windows and Mac-OS
  - Application: R, RStudio, Excel

What is a program?

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish

- An algorithm takes input, perform some operations and gives output
  - Executes in finite time
  - E.g. sorting, searching, reading, copying!

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish

- An algorithm takes input, perform some operations and gives output
  - Executes in finite time
  - E.g. sorting, searching, reading, copying!

- Complexity of a Program
  - Time and space!
  - E.g. Fibonacci series! (next slide)

What is a program?

- Translation of an algorithm into a language that computer understands
  - Think of it like a recipe. With the right ingredients and right procedure, you get the right dish

- An algorithm takes input, perform some operations and gives output
  - Executes in finite time
  - E.g. sorting, searching, reading, copying!

- Complexity of a Program
  - Time and space!
  - E.g. Fibonacci series! (next slide)

- Programming Paradigms
  - Iterative vs Recursive
  - Procedural vs Object Oriented
    - No need to understand OOP concepts unless you wish to build a software (like an R package)

Fibonacci Time Complexity Example

Fibonacci Time Complexity Example

- Fibonacci numbers are defined as:

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.

Fibonacci Time Complexity Example

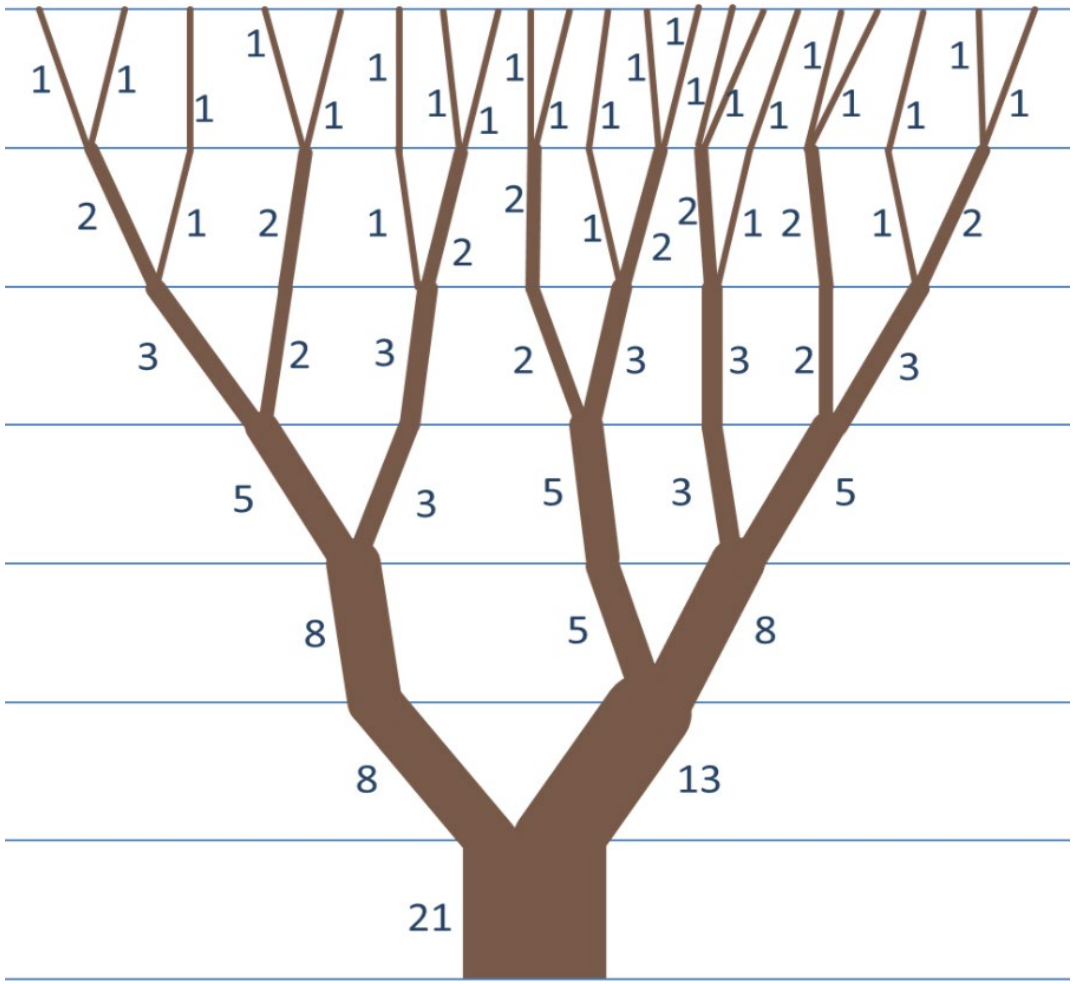
- Fibonacci numbers are defined as:

$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.



Fibonacci Time Complexity Example

- Fibonacci numbers are defined as:

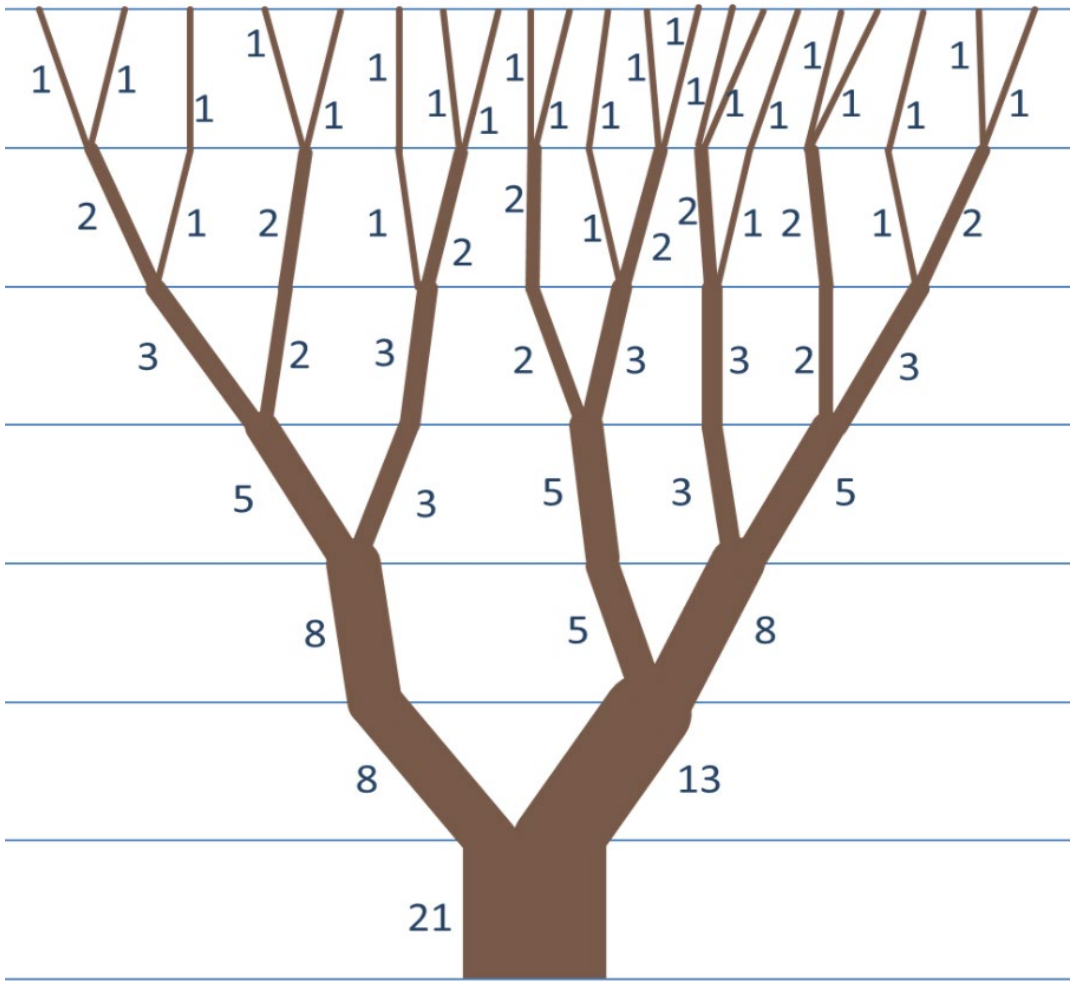
$$F_0 = 0, \quad F_1 = 1,$$

and

$$F_n = F_{n-1} + F_{n-2}$$

for $n > 1$.

- Thus, finding a Fibonacci number is pretty straight-forward.



3 ways to find Fib(n)

3 ways to find Fib(n)

- Method-1,
  return 1
  - Fib(n-1) + Fib(n-2)
  - Recursion
  - Base case:
    - If $n \leq 2$,
      return 1

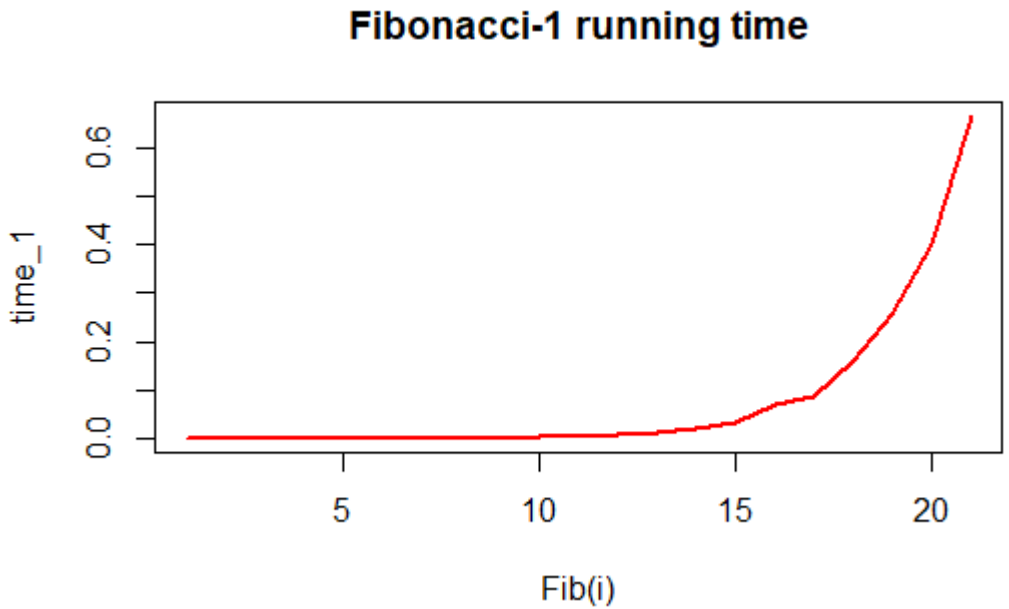
3 ways to find Fib(n)

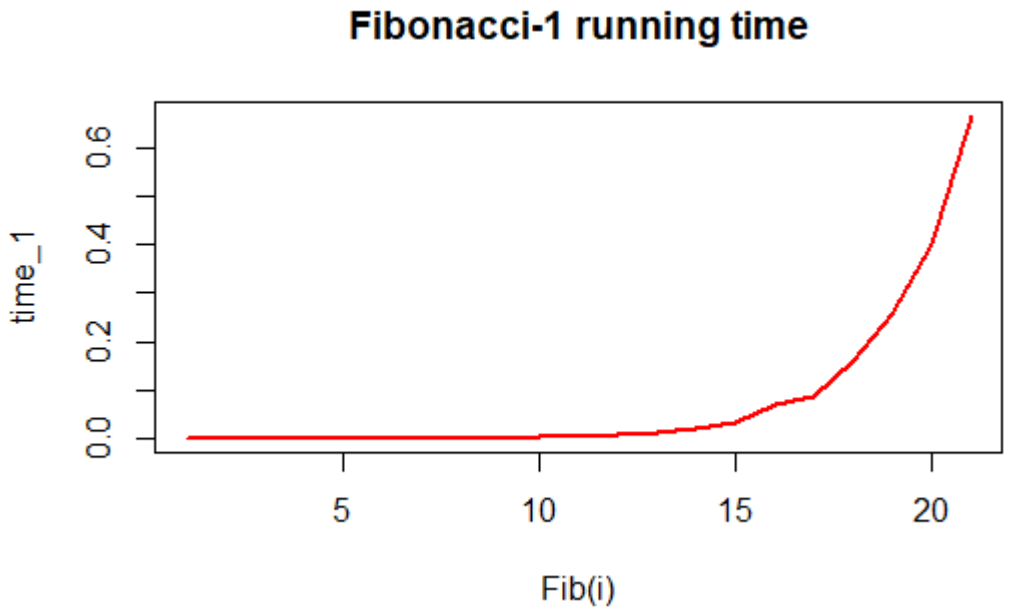
- Method-1,
  return 1
  - Fib(n-1) + Fib(n-2)
  - Recursion
  - Base case:
    - If $n \leq 2$,
      return 1

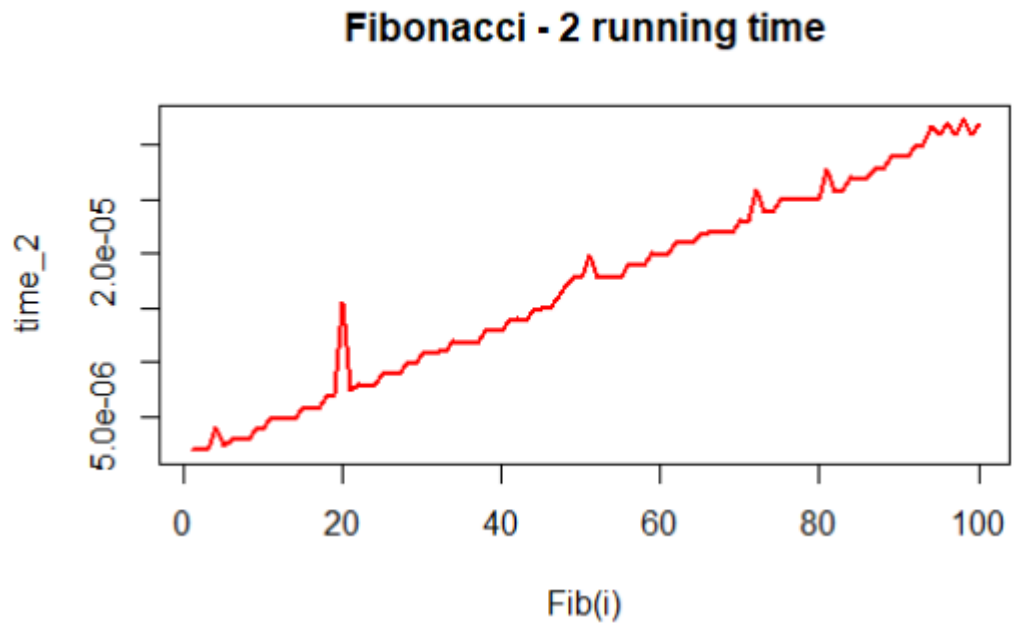
- Method-2
  - a = b = 1
  - for $i \in 3 \ldots n$
    - f = a + b
    - a = b
    - b = f
  - Iterative
  - Base Case:
    - If $n \leq 2$,
      return 1

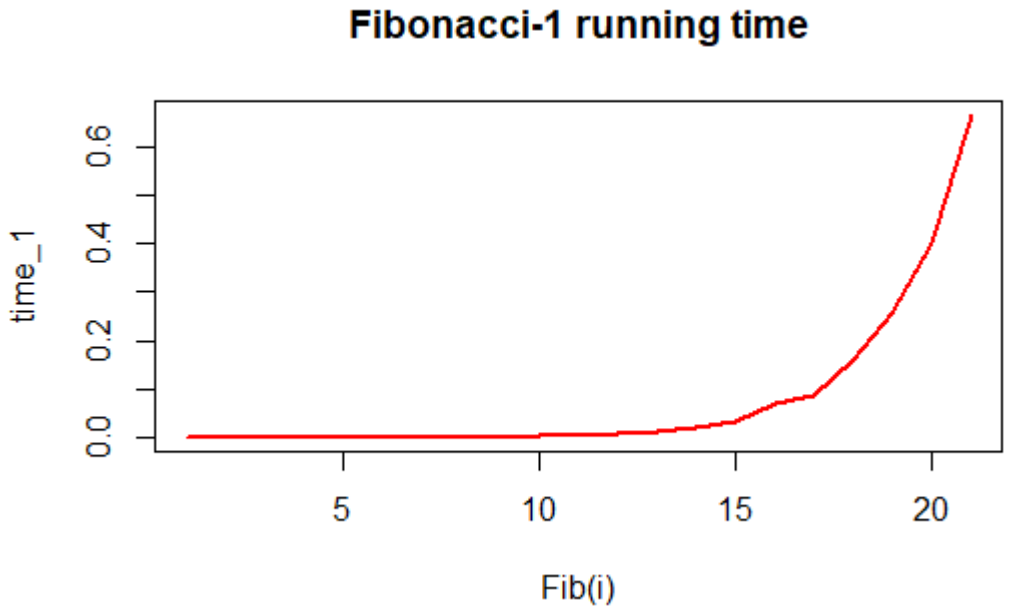
3 ways to find Fib(n)

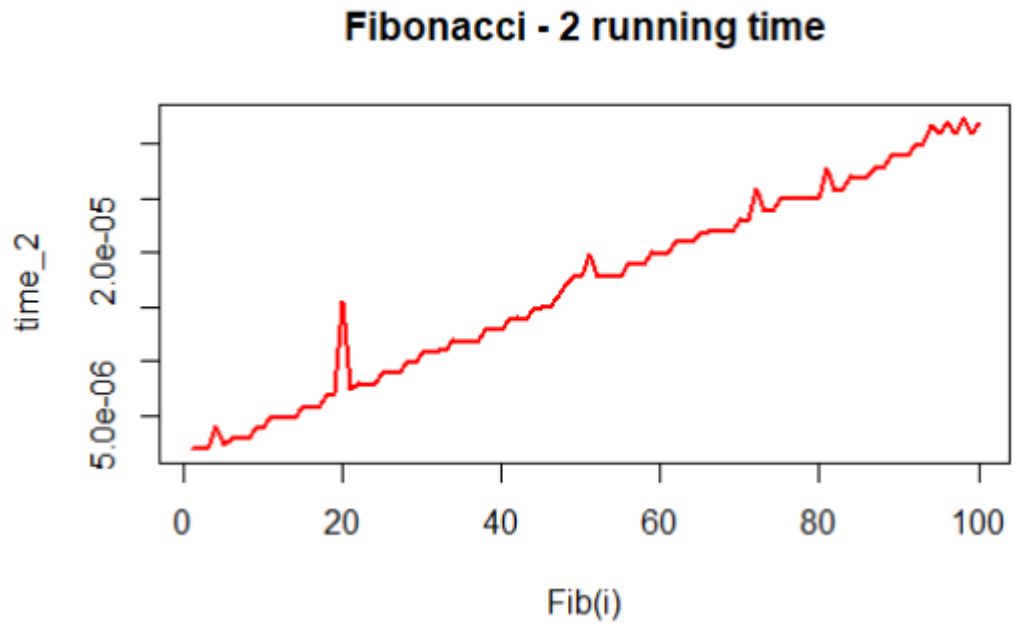
- Method-1, return 1
  - Fib(n-1) + Fib(n-2)
  - Recursion
  - Base case:
    - If $n \leq 2$, return 1
- Method-2
  - a = b = 1
  - for $i \in 3 \ldots n$
    - f = a + b
    - a = b
    - b = f
  - Iterative
  - Base Case:
    - If $n \leq 2$, return 1
- Method-3
  - $\phi = \frac{1+\sqrt{5}}{2}$
  - $\hat{\phi} = \frac{1-\sqrt{5}}{2}$
  - $Fib(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}}$
  - Constant time
    - Really?

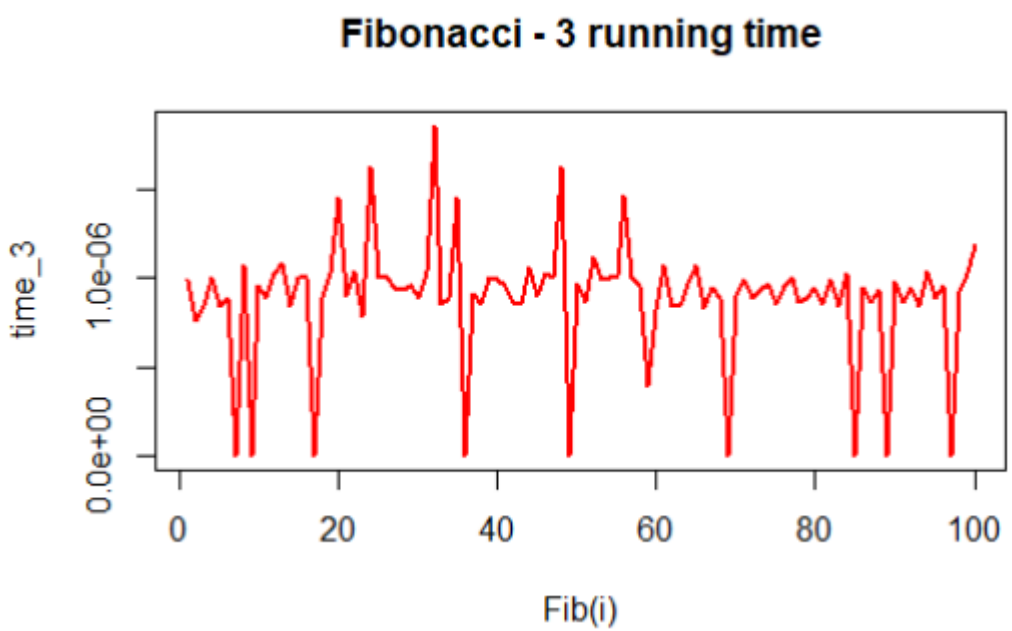

Fibonacci-1 running time
time_1
0.0
0.2
0.4
0.6
5
10
15
20
Fib(i)

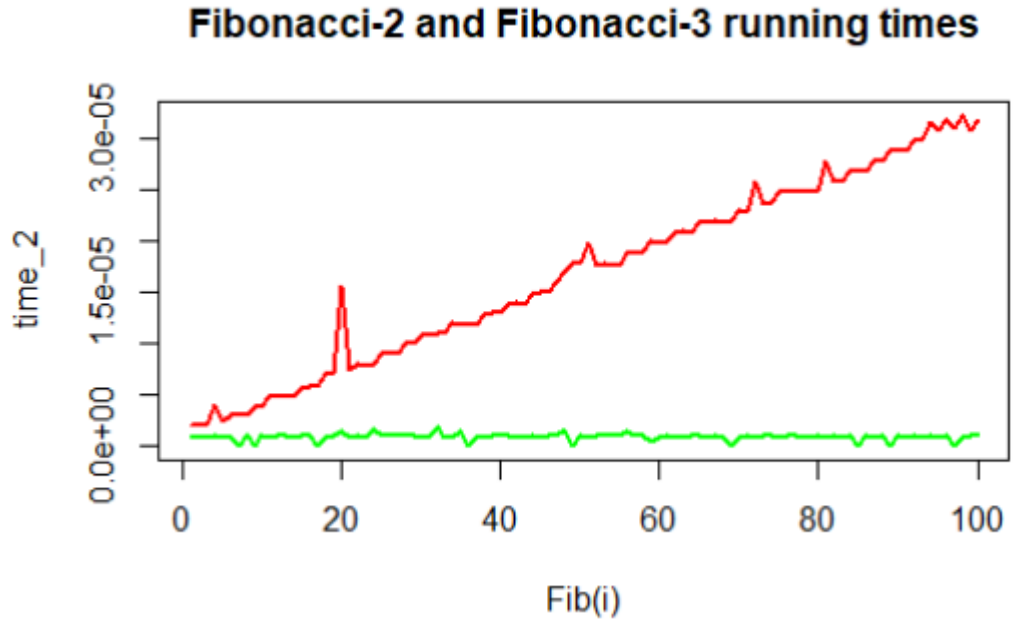

Fibonacci-1 running time
time_1
0.0
0.2
0.4
0.6
5
10
15
20
Fib(i)


Fibonacci - 2 running time
time_2
5.0e-06
2.0e-05
0
20
40
60
80
100
Fib(i)


Fibonacci-1 running time
time_1
0.0
0.2
0.4
0.6
5
10
15
20
Fib(i)


Fibonacci - 2 running time
time_2
5.0e-06
2.0e-05
0
20
40
60
80
100
Fib(i)


Fibonacci - 3 running time
time_3
0.0e+00
1.0e-06
0
20
40
60
80
100
Fib(i)


Fibonacci-2 and Fibonacci-3 running times
time_2
0.0e+00
1.5e-05
3.0e-05
0
20
40
60
80
100
Fib(i)

What makes a good program?

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn’t worth pursuing

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn’t worth pursuing
- Readability
  - Make your code succinct and comments verbose
    - You will forget your code within weeks
    - The key to writing good code is the ability to quickly recap previously written code

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn't worth pursuing
- Readability
  - Make your code succinct and comments verbose
    - You will forget your code within weeks
    - The key to writing good code is the ability to quickly recap previously written code
- Organized
  - Writing code is an art. Much like a story!
  - Make small modules which can be reused later. Make it like a building: Bricks → Wall → floor → Building

What makes a good program?

- Efficient
  - Think about complexity
    - Will your code scale when input is 10x bigger? 100x?
  - Identify parts of your program that runs a million times
    - And optimize those the best you can
    - A project that takes forever to run isn't worth pursuing
- Readability
  - Make your code succinct and comments verbose
    - You will forget your code within weeks
    - The key to writing good code is the ability to quickly recap previously written code
- Organized
  - Writing code is an art. Much like a story!
  - Make small modules which can be reused later. Make it like a building: Bricks → Wall → floor → Building
- Error Handling
  - Identify potential errors and print messages so that you know what and where problem occurred
  - For time consuming code, print regular messages (logs) in a file
    - You don't want to run 4 hours of code just to find a small bug! Instead look at the log file.