Lecutre 1 Introduction

1 What is this course

- Programming and problem solving, with applications.
- **Algorithms**: method for solving a problem.
- **Data structure**: method to store information.

2 Algorithms

- **Definition**: A well-defined computational procedure to solve a computational ***problem*** (to transform some input into a desired output).
- Statement of the ***problem*** specifies the desired ***input/output relationship***.
- Algorithm describes a specific computational procedure for achieving that input/output relationship.

3 Why study algorithms and performance?

- Algorithms help us to understand ***scalability***.
- Performance often draws the line between what is feasible and what is impossible.
- Algorithmic mathematics provides a ***language*** for talking about program behavior.
- Performance is the ***currency***of computing.
- The lessons of program performance generalize to other computing resources.
- Speed is fun!
- For intellectual stimulation.
    - ***" An algorithm must be seen to be believed. "*** *— Donald Knuth*
    - " *For me,* ***great algorithms are the poetry of computation***. Just like verse, they can be terse, allusive, dense, and even mysterious. But once unlocked, they cast a brilliant new light on some*

      aspect of computing. " — Francis Sullivan
- To become a proficient programmer.
    - *" I will, in fact, claim that the difference between a bad programmer and a good one is whether he considers his code or his data structures more important. Bad programmers worry about the code. Good programmers worry about data structures and their relationships. "*

      — Linus Torvalds (creator of Linux)
    - *" Algorithms + Data Structures = Programs. " — Niklaus Wirth*
- Thinking and solving problems like a computer scientist.

4 Fibonacci Numbers

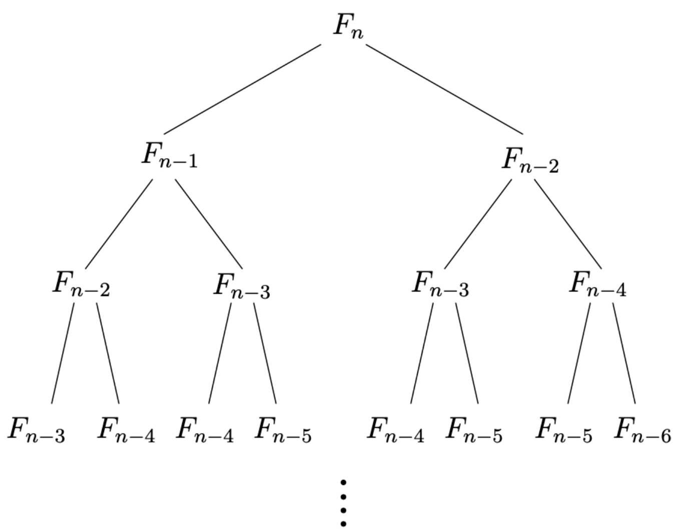
- Fibonacci is most widely known for his famous sequence of numbers

  $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots,$

  $$F_n = \begin{cases} 0 & \text{n=0} \\ 1 & \text{n=1} \\ F_{n-1} + F_{n-2} & \text{n>=2} \end{cases}$$
- In fact, the Fibonacci numbers **grow** almost as fast as the powers of 2. $F_n \approx 2^{0.694n}$

```
function fib(n):
  if n=0
    return 0
  if n=1
    return 1
  return fib(n-1) + fib(n-2)
```

- Whenever we have an algorithm, there are three questions we always ask about it:
  - Is it correct?
  - How much time does it take, as a function of n?
  - And can we do better?
- Is it correct?
  - this algorithm is precisely Fibonacci's definition of F_n.
- How much time does it take, as a function of n?
  - Let $T(n)$ be the number of *computer steps* need to compute `fib(n)`.
  - If n is less than 2, the procedure halts $T(n) \leq 2$.
  - For large values of n, there are two recursive invocation of `fib(n)`, taking time $T(n-1) + T(n-2)$.
    - Therefore, $T(n) = T(n-1) + T(n-2) + 3$
    - $T(n) \geq F_n$.
    - The running time of the algorithm grows as fast as the Fibonacci numbers!
    - $T(n)$ is exponetial in n.
- Can we do better ?
  - The following figure shows the cascade of recursive invo- cations triggered by a single call to `fib(n)`.



   - *Notice that many computations are repeated!*

```
function fib2(n)
  if n = 0 return 0
  create an array f[0...n]
  f[0] = 0
  f[1] = 1
  for i=2 to n
    f[i] = f[i-1] + f[i-2]
  return f[n]
```

- How long does it take?
   - The loop consists of a single computer step and is executed $n-1$ times.
   - The running time of `fib2(n)` is *linear* in n.

5 Challenge

- Will my program be able to solve a large practical input ?
- Use **scientific method** to understand performance.
   - *Observe* some feature of the natural world.
   - *Hypothesize* a model that is consistent with the observations.
   - **Predict* events using the hypothesis.
   - *Verify* the predictions by making further observations.
   - *Validate* by repeating until the hypothesis and observations agree.
- Principles
   - Experiments must be *reproducible*.
   - Hypotheses must be *falsifiable*.