

# Agenda

1

- The worst algorithm in the history of humanity
- Asymptotic notations: Big-O, Big-Omega, Theta
- An iterative solution
- A better iterative solution
- The repeated squaring trick

# Fibonacci sequence

2

**THE WORST ALGORITHM IN HISTORY**

# Fibonacci sequence

3

- 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, ...
- $F[0] = 0$
- $F[1] = 1$
- $F[2] = F[1] + F[0] = 1$
- $F[3] = F[2] + F[1] = 2$
- $F[4] = F[3] + F[2] = 3$
- $F[n] = F[n-1] + F[n-2]$

# Recursion – fib1()

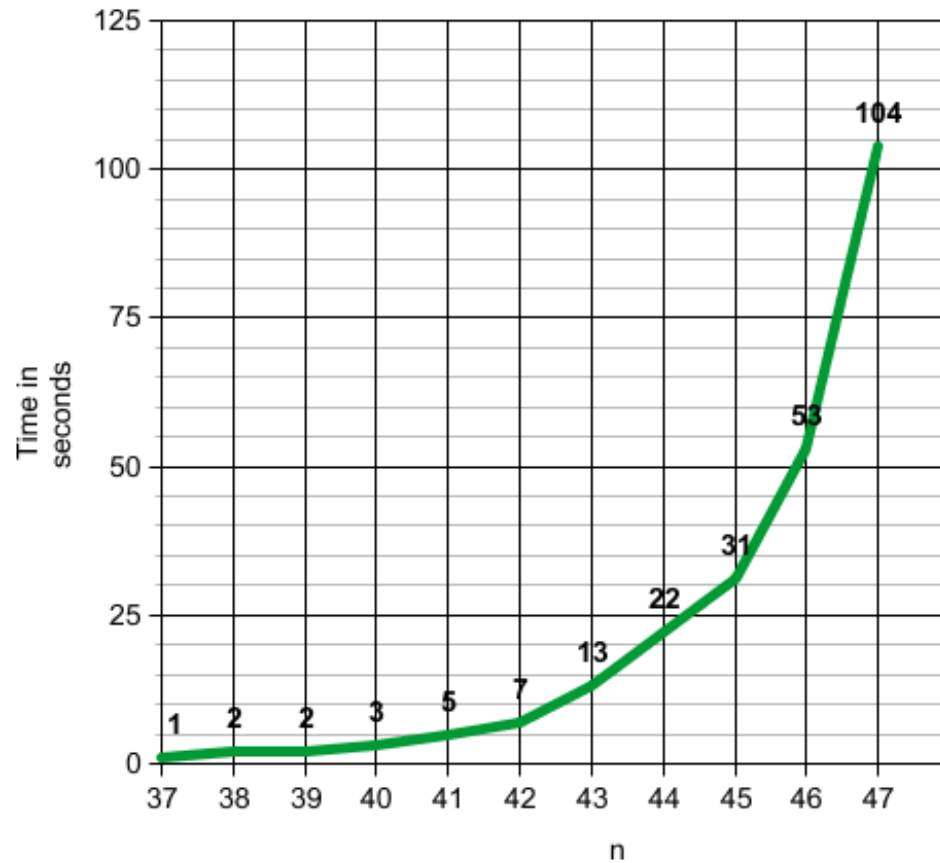
4

```
/**  
 *-----  
 * the most straightforward algorithm to compute F[n]  
 *-----  
 */  
unsigned long long fib1(unsigned long n) {  
    if (n <= 1) return n;  
    return fib1(n-1) + fib1(n-2);  
}
```

# Run time on my laptop

5

Fib1 run time



# On large numbers

6

- Looks like the run time is doubled for each  $n++$
- We won't be able to compute  $F[120]$  if the trend continues
- The age of the universe is 15 billion years  $< 2^{60}$  sec
- The function looks ... exponential
  - Is there a theoretical justification for this?

# Functions

7

- Sometimes we mean a C++ function
- Sometimes we mean a mathematical function like  $F[n]$
- A C++ function can be used to compute a mathematical function
  - But not always
- What we mean should be clear from context

# Analysis of fib1()

8

**GUESS AND INDUCT STRATEGY**

**THINKING ABOUT THE MAIN BODY**



# Guess and induct

9

- For  $n > 1$ , suppose it takes  $c$  mili-sec in  $\text{fib}_1(n)$  not counting the recursive calls
- For  $n=0, 1$ , suppose it takes  $d$  mili-sec
- $T[n] =$  time  $\text{fib}_1(n)$  takes
- $T[0] = T[1] = d$
- $T[n] = c + T[n-1] + T[n-2]$  when  $n > 1$
  
- To estimate  $T[n]$ , we can
  - Guess a formula for it
  - Prove by induction that it works

# The guess

10

- Bottom-up iteration
  - $T[0] = T[1] = d$
  - $T[2] = c + 2d$
  - $T[3] = 2c + 3d$
  - $T[4] = 4c + 5d$
  - $T[5] = 7c + 8d$
  - $T[6] = 12c + 13d$
- Can you guess a formula for  $T[n]$ ?
  - $T[n] = (F[n+1] - 1)c + F[n+1]d$

# The Proof

11

- The base cases:  $n=0,1$
- The hypothesis: suppose
  - $T[m] = (F[m+1] - 1)c + F[m+1]d$  for all  $m < n$
- The induction step:
  - $$\begin{aligned} T[n] &= c + T[n-1] + T[n-2] \\ &= c + (F[n] - 1)c + F[n]d \\ &\quad + (F[n-1] - 1)c + F[n-1]d \\ &= (F[n+1] - 1)c + F[n]d \end{aligned}$$

# How does this help?

12

$$F[n] = \frac{\phi^n - (-1/\phi)^n}{\sqrt{5}}$$

$$\phi = \frac{1 + \sqrt{5}}{2} \approx 1.6$$

The golden ratio

So, there are constants  $C, D$  such that

13

$$C\phi^n \leq T[n] \leq D\phi^n$$

This explains the exponential-curve we saw

# Asymptotic analysis

14

- **BACK OF THE ENVELOPE TIME/SPACE ESTIMATION**
- **INDEPENDENT OF WHETHER OUR COMPUTER IS FAST**
  - **BIG-O, BIG-OMEGA, THETA**

# From intuition to formality

15

- If we ran fib1() on a computer with  $C = 10^{-9}$ :

$$10^{-9} (1.6)^{140} \geq 3.77 \cdot 10^{19} > 100 \cdot \text{age of univ.}$$

- We need a formal way to state that  $(1.6)^n$  is the “correct” measure of fib1()’s runtime
  - How fast the target computer runs shouldn’t concern us

# Big-O

16

$$f, g : \mathbb{N} \rightarrow \mathbb{R}^+$$

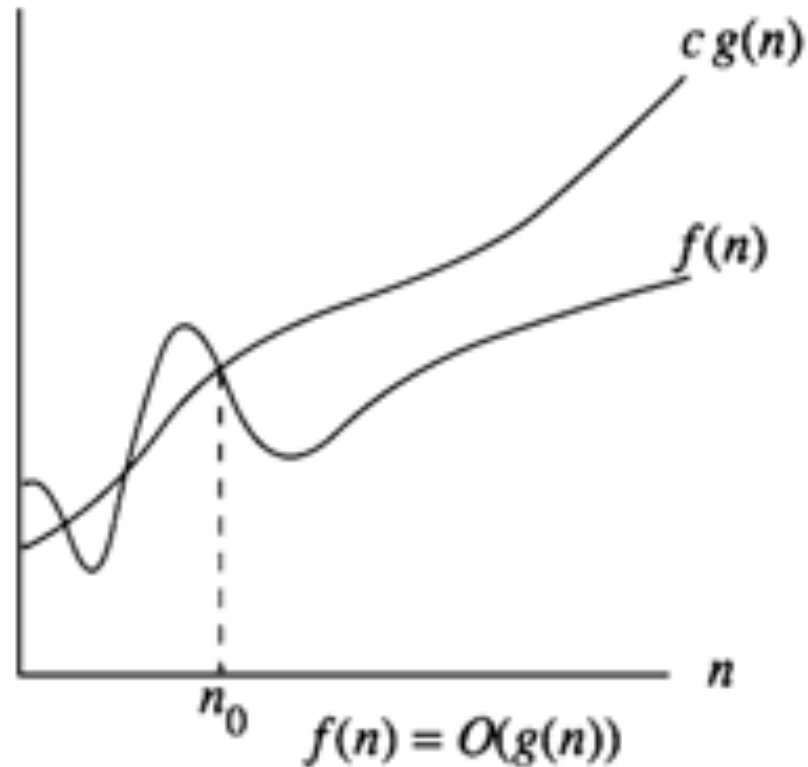
$$f(n) = O(g(n)) \text{ iff } \exists \text{ constants } C, n_0 > 0$$

$$\text{such that } f(n) \leq Cg(n), \forall n \geq n_0$$



# Intuition

17



in our case  $T[n] = O(\phi^n)$

# In English

18

- $f(n) = O(g(n))$  means: for  $n$  sufficiently large,  $f(n)$  is bounded above by a constant scaling of  $g(n)$ 
  - Does the “English translation” make things worse?
- An algorithm with runtime  $f(n)$  is at least as good as an algorithm with runtime  $g(n)$ , asymptotically

# Examples

19

$$n^2 = O(n^2)$$

$$n^2 = O(n^2 / 10^6)$$

$$n = O(n^2)$$

# Big-Omega

20

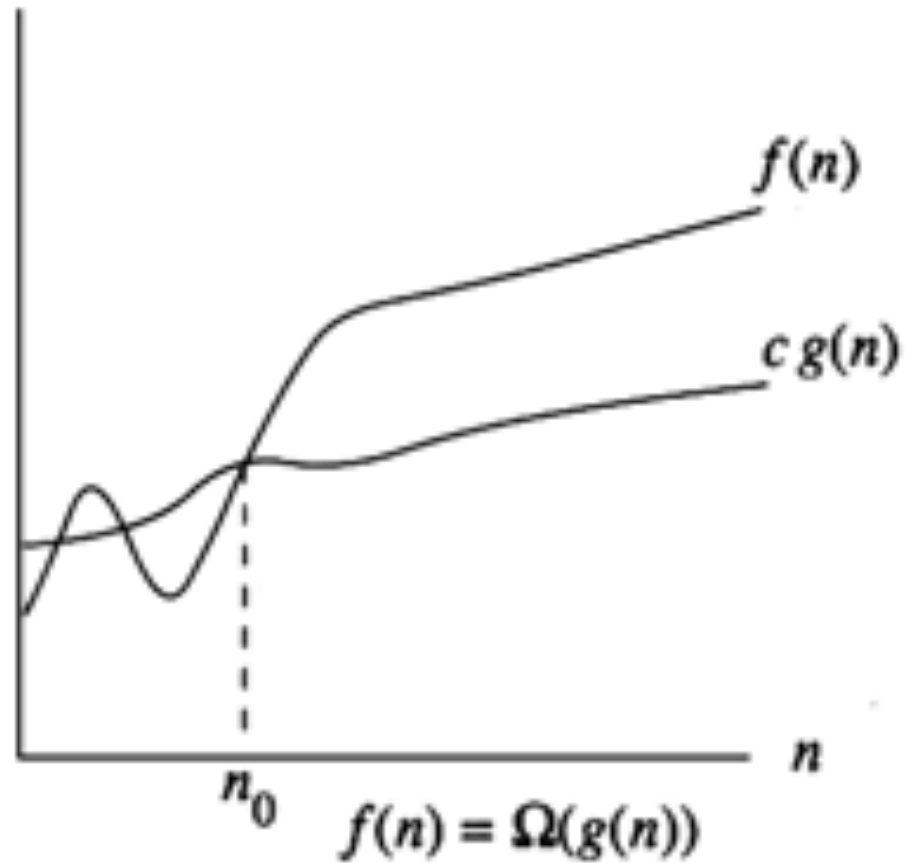
$$f, g : \mathbb{N} \rightarrow \mathbb{R}^+$$

$$f(n) = \Omega(g(n)) \text{ iff } \exists \text{ constants } C, n_0 > 0$$

$$\text{such that } f(n) \geq Cg(n), \forall n \geq n_0$$

# In picture

21



# Examples

22

$$n \log n = \Omega(n)$$

$$2^n / 10^6 = \Omega(n^{100})$$

# Equivalence

23

$$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$$

# Theta

24

$$f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \text{ and } g(n) = O(f(n))$$

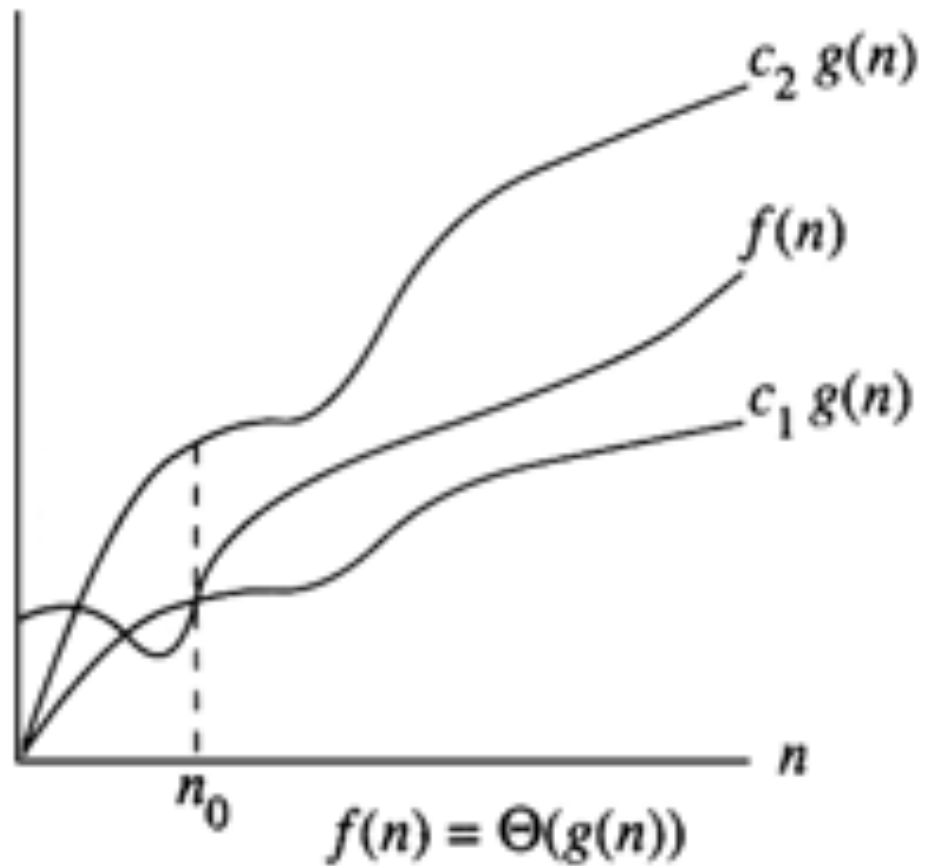
We say they “have the same growth rate”

in fib1() example:  $T[n] = \Theta(\phi^n)$



# In picture

25



# Better algorithms for $F[n]$

26

- **A LINEAR TIME ALGORITHM USING VECTORS**
- **A LINEAR TIME ALGORITHM USING ARRAYS**
- **A LINEAR TIME ALGORITHM WITH CONSTANT SPACE**

# An algorithm using vector

27

```
unsigned long long fib2(unsigned long n) {  
    // this is one implementation option  
    if (n <= 1) return n;  
    vector<unsigned long long> A;  
    A.push_back(0); A.push_back(1);  
    for (unsigned long i=2; i<=n; i++) {  
        A.push_back(A[i-1]+A[i-2]);  
    }  
    return A[n];  
}
```

Guess how large an n we can handle this time?

# Data

28

<b>n</b>	<b><math>10^6</math></b>	<b><math>10^7</math></b>	<b><math>10^8</math></b>	<b><math>10^9</math></b>
<b># seconds</b>	<b>1</b>	<b>1</b>	<b>9</b>	<b>Eats up all my CPU/ RAM</b>

# How about an array?

29

```
unsigned long long fib2(unsigned long n) {  
    if (n <= 1) return n;  
    unsigned long long* A = new unsigned long long[n];  
    A[0] = 0; A[1] = 1;  
    for (unsigned long i=2; i<=n; i++) {  
        A[i] = A[i-1]+A[i-2];  
    }  
    unsigned long long ret = A[n];  
    delete [] A;  
    return ret;  
}
```

# Data

30

<b>n</b>	<b><math>10^6</math></b>	<b><math>10^7</math></b>	<b><math>10^8</math></b>	<b><math>10^9</math></b>
# seconds	1	1	1	Segmentation fault

Data structure matters a great deal!

Some assumptions we made are false if too much space is involved: computer has to use hard-drive as memory

# Dynamic programming!

31

```
unsigned long long fib3(unsigned long n) {
    if (n <= 1) return n;
    unsigned long long a=0, b=1, temp;
    unsigned long i;
    for (unsigned long i=2; i<= n; i++) {
        temp = a + b; // F[i] = F[i-2] + F[i-1]
        a = b;        // a = F[i-1]
        b = temp;     // b = F[i]
    }
    return temp;
}
```

# Data

32

<b>n</b>	<b><math>10^8</math></b>	<b><math>10^9</math></b>	<b><math>10^{10}</math></b>	<b><math>10^{11}</math></b>
# seconds	1	3	35	359

The answers are incorrect because  $F[10^8]$  is greater than the largest integer representable by unsigned long long

But that's ok. We want to know the runtime



# An even faster algorithm

33

- **THE REPEATED SQUARING TRICK**

# Math helps!

34

- We can re-formulate the problem a little:

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \quad \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} F[n-1] \\ F[n-2] \end{bmatrix} = \begin{bmatrix} F[n] \\ F[n-1] \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} \quad \begin{bmatrix} F[n+1] \\ F[n] \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

# How to we compute $A^n$ quickly?

35

- Want

$$\begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n$$

- But can we even compute  $3^n$  quickly?

# First algorithm

36

```
unsigned long long power1(unsigned long n) {  
    unsigned long i;  
    unsigned long long ret=1;  
    for (unsigned long i=0; i<n; i++)  
        ret *= base;  
    return ret;  
}
```

When  $n = 10^{10}$  it took 44 seconds

# Second algorithm

37

```
unsigned long long power2(unsigned long n) {  
    unsigned long long ret;  
    if (n == 0) return 1;  
    if (n % 2 == 0) {  
        ret = power2(n/2);  
        return ret * ret;  
    } else {  
        ret = power2((n-1)/2);  
        return base * ret * ret;  
    }  
}
```

When  $n = 10^{19}$  it took  $< 1$  second

Couldn't test  $n = 10^{20}$  because that's  $> \text{sizeof}(\text{unsigned long})$

# Runtime analysis

38

- First algorithm  $O(n)$
- Second algorithm  $O(\log n)$
- We can apply the second algorithm to the Fibonacci problem: `fib4()` has the following data

<b>n</b>	<b><math>10^8</math></b>	<b><math>10^9</math></b>	<b><math>10^{10}</math></b>	<b><math>10^{19}</math></b>
# seconds	1	1	1	1