Fibonacci Series

0 1 1 2 3 5 8 . . . . → Values increase exponentially

F(0) = 0
F(1) = 1
F(n) = F(n-1) + F(n-2)

→ fib1 recursive algo

$O(2^n)$ calls

F(n) : the value is $O(2^n)$ → store in $O(n)$ bits

n = 60

$F(n) \cong 2^{60}$

addition cannot be assumed to be O(1)

$O(2^n)$ calls × O(n) per addition

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

exponential

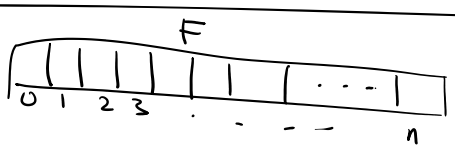
F(10000) : it take n bits to store 10000

addition on large n costs O(n) time

F(10000) = F(9999) + F(9998)

---

fib2(n):



F[0] = 0
F[1] = 1
for i = 2 to n
    F[i] = F[i-1] + F[i-2]

O(n) calls × O(n) per addition

$= O(n^2)$

Polynomial : quadratic

---

fib3(n): Matrix Multiplication based

Theory vs Practical

Matrix Multiplication based

$$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = 1 \cdot F_1 + 0 \cdot F_0$$
$$= 1 \cdot F_1 + 1 \cdot F_0$$

$$\begin{pmatrix} F_1 \\ F_2 \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

$$\begin{pmatrix} F_2 \\ F_3 \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{pmatrix} F_1 \\ F_2 \end{pmatrix}$$

$F_2 = F_2$

$F_3 = F_1 + F_2 = F_2 + F_1$

$$= \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \left[\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix} \right]$$

$$\begin{pmatrix} F_{②} \\ F_3 \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^{②} \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$

M

$F_0 = 0$
$F_1 = 1$ base case

Theory vs Practical

$O(n^{\log_2 3})$ time

$= O(n^{1.585})$ vs $O(n^2)$

asymptotically $n^{1.585}$ is way faster than n^2

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{pmatrix} F_0 \\ F_1 \end{pmatrix}$$ ← show by induction

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

original formula

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix} \xrightarrow{\text{algo}} M^n = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

2x2 matrix

fib3(n):

O(1) → given $M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$

Compute the ... power ... of M

given $M = \begin{bmatrix} \cdot & \cdot \\ 1 & 1 \end{bmatrix}$

Compute the n-th power of M

$$M^n = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

?

$a, b, c, d \approx O(2^n)$ in value

very large integers

8 multiplications
4 additions
$O(n)$ time

$$\begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{pmatrix} 0 \\ 1 \end{pmatrix}$$

F_0, F_1

$$\rightarrow \begin{pmatrix} F_n \\ F_{n+1} \end{pmatrix} = \begin{pmatrix} b \\ d \end{pmatrix}$$

return b

$$M = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$$

how to compute $M^n = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}^n$

$$M^2 = \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}}_{2\times 2} \times \underbrace{\begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}}_{2\times 2} = \begin{bmatrix} 1 & 1 \\ 1 & 2 \end{bmatrix}$$

8 multiplications
4 additions

$0 \cdot 0 + 1 \cdot 1$
mul mul
add

adding 2 n-bit numbers → $O(n)$ time

multiplying 2 n-bit numbers → $O(n^2)$ time

typically these are $O(1)$ time for 64 bits

Each matrix multiplication takes $O(n^2)$ time

n steps × $n(n^2)$:

M

$M^2 = M \times M$ — $O(n^2)$

$M^3 = M^2 \times M$ — $O(n^2)$

:

naive !

n steps
×
$O(n^2)$ time per step
=
$O(n^3)$

$M^5 = M^2 \times M \quad - O(n^2)$
⋮
$M^n = M^{n-1} \times M \quad - O(n^2)$

Naive

loop : 1 ... n
linear

M ← powers of the matrix

logarithmic progression

$M^n = M^{n/2} \times M^{n/2}$

$n = 2^k$: n is a power of 2

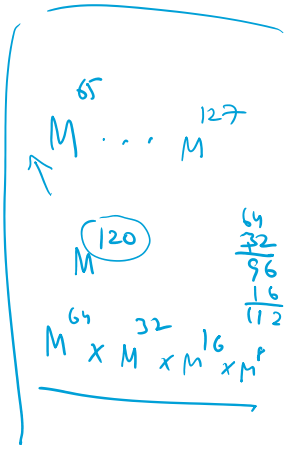
$M^{64} = M^{32} \times M^{32} \quad - O(n^2)$
$M^{32} = M^{16} \times M^{16}$
$M^{16} = M^8 \times M^8$
$M^8 = M^4 \times M^4$
$M^4 = M^2 \times M^2$
$M^2 = M \times M$

# of steps
$\log_2 n$

$\log n = k$

```
M = M
for i = 1 .. log n
    M = M x M
```

$\log_2 n$ vs n steps



$\log_2 n$ steps × $O(n^2)$ multiplication time

$= O(n^2 \log n)$ vs $O(n^3)$

log progression

linear progression

naive analysis

$O(n^2)$

→ rigorous analysis of running time

Recursive formula for running time

$T(n) \equiv$ time the algo takes on input n

$$T(n) = T(n/2) + O\left(\left(\frac{n}{2}\right)^2\right)$$

Compute M^n

Compute $M^{n/2}$

multiply $M^{n/2} \times M^{n/2}$

(2×2) (2×2)

8 multiplications $\rightarrow O(n^2)$

4 adds

$$T(n) = \boxed{T(n/2)} + O\left(\frac{n^2}{4}\right)$$

$$= \left[T\left(\frac{n}{4}\right) + O\left(\frac{n^2}{16}\right)\right] + O\left(\frac{n^2}{4}\right)$$

$$= T\left(\frac{n}{4}\right) + O\left(\frac{n^2}{16}\right) + O\left(\frac{n^2}{4}\right)$$

$$= T\left(\frac{n}{8}\right) + O\left(\frac{n^2}{64}\right) + O\left(\frac{n^2}{16}\right) + O\left(\frac{n^2}{4}\right)$$

$$= O\left(\frac{n^2}{4}\right) + O\left(\frac{n^2}{16}\right) + O\left(\frac{n^2}{64}\right) + \cdots + T(1)$$

$$= \frac{n^2}{4}\left(O(1) + O\left(\frac{1}{4}\right) + O\left(\frac{1}{16}\right) + \cdots\right)$$

$$= \frac{n^2}{4}\left(\boxed{1 + \frac{1}{4} + \frac{1}{16} + \cdots}\right)$$

log steps

geometric series $r = 1/4$

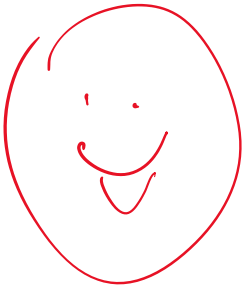
geometric series $r = 1/4$

$$\sum_{i=0}^{\infty} r^i = \frac{1}{1-r} \quad \text{if } |r| < 1$$

$$\frac{1}{1-1/4} = \frac{1}{3/4} = 4/3$$

$$\leq \frac{n^2}{4} \cdot \frac{4}{3}$$

$$= O(n^2)$$



$O(n^2)$ time to multiply 2 n-bit numbers

$\Omega(\sqrt{n} \cdot n) \approx$ lower bound

$O(n^{1.585})$ upper bound

we can multiply 2 n-bit numbers in $O(n^{\log_2 3})$ time !

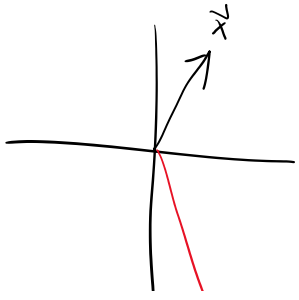
divide-and-conquer

powers of a matrix

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

2x2 symmetric matrix

↳ real eigenvalues
orthogonal eigenvectors



→ real eigenvalue
orthogonal eigenvectors

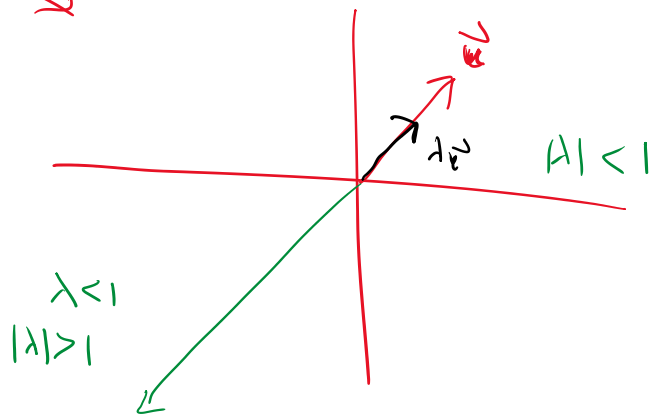
$M\vec{u} = \lambda\vec{u}$ eigenequation

matrix · vector eigen · scalar eigenvalue · eigenvector

$\vec{x}' = M\vec{x}$ → $\vec{x}'$

linear transformation

$\vec{u}$ is an invariant direction



$Mu_1 = \lambda_1 u_1$
$Mu_2 = \lambda_2 u_2$

M^n → written in terms of eigenvectors & eigenvalues

$$M^n = \begin{bmatrix} | & | \\ u_1 & u_2 \\ | & | \end{bmatrix} \begin{bmatrix} \lambda_1^n & \\ & \lambda_2^n \end{bmatrix} \begin{bmatrix} - u_1 - \\ - u_2 - \end{bmatrix}$$

$$F_n = \frac{1}{\sqrt{5}}\left(\lambda_1^n - \lambda_2^n\right)$$

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

mathematically elegant

mathematically elegant ☺

$O(n^2)$ ☹

$\sqrt{5}$ is irrational ⟶ precision issue

Big-Oh Notation

$[5n^2 + 100n] + 1000$ operations

analysis of algorithms ⟵ asymptotic analysis

focus is on leading terms

$n \to \infty$

as n becomes large

"big-oh of $g(n)$"

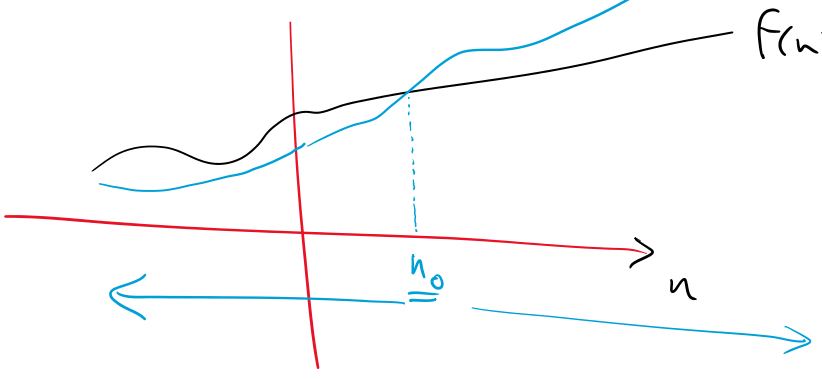
$f(n)$ is $O(g(n))$ if $\exists$ constant c and $\exists$ an integer n_0 such that for all $n > n_0$

$$f(n) \leq c \cdot g(n)$$

n is a variable

eventually $g(n)$ dominates $c \cdot g(n)$

or the graph of $f(n)$ lies below graph of $g(n)$



If $f(n)$ is $O(g(n))$ then we say that $g(n)$ is $\Omega(f(n))$ big-omega

$f(n)$ is $\Omega(g(n))$ if $\exists c, \exists n_0$, such that $\forall n > n_0$

$$f(n) \geq c\, g(n)$$

If $f(n) = O(g(n))$ and $g(n) = O(f(n))$

$f(n) = \Theta(g(n))$ big-theta

$f(n) = 5n^2 + 100n + 1000$

$g(n) = 0.1n^2 + 5$

$f(n) = \Theta(g(n))$

---

how do these functions behave: $n \to \infty$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \begin{cases} \infty & \text{then } f(n) = \Omega(g(n)) \\ 0 & \text{then } f(n) = O(g(n)) \\ a & \text{then } f(n) = \Theta(g(n)) \end{cases}$$

a: constant

$\lim_{n\to\infty}$ $f(n) = (n^2 + 5n)/n^2$

$g(n) = (10n^2 + 2n + 100)/n^2$

$$= \frac{1 + 5/n}{10 + 2/n + 100/n^2} \to \frac{1}{10} = 0.1$$

$5/n \to 0$, $2/n \to 0$, $100/n^2 \to 0$

$f(n) = \Theta(g(n))$

$f(n) = \log_2 n$

$g(n) = \sqrt{n}$

$$\lim_{n \to \infty} \frac{\log_2 n}{n^{1/2}} \equiv \lim_{x \to \infty} \frac{x}{(2^x)^{1/2}} = \frac{x}{2^{x/2}}$$

$$\log_2 n = x \quad = 0$$

$$n = 2^x$$

$f(n) = O(g(n))$

$O(1)$ ← Constant time

$O(\log n)$ ← logarithmic

$O(n)$ ← linear

$O(n \log n)$ ← linearithmic

$O(n^2)$ ← quadratic

$\vdots$

$O(n^k)$ ← polynomial

$O(2^n)$ ← exponential

$O(3^n)$ ←

$O(n!)$ ← factorial