

**Reading:** Chapter 1 & 2.

**Announcements:**

- Canvas (vs. Piazza?)
- grading:
  - homework: 50%
  - participation: 10%
  - midterm: 15% (10/27)
  - final: 25% (12/3)
- new labs.
  - Monday: 10, 11, 4, 5.
  - Tuesday: 10, 11.
- homework partners (must be same lab)
- Homework plan:
  - assigned thursday, due thursday, work in pairs, graded for accuracy and quality.
  - peer review (mandatory and extra credit).
  - automatic extension to sunday (for 25% of grade)
- TA: Sam Taggart.
- office hours
  - hartline: Tues, 1-2pm, Ford 3-329.
  - taggart: Wed, 10:30-12pm, TBA.

## Algorithms

- algorithms are everywhere. examples:
  - digital computers,
  - parliamentary procedure,
  - scientific method,
  - biological processes.
- algorithms design and analysis governs everything.
- good algorithms are closest things to magic.
- course philosophy: no particular algorithm is important.
- course goals: how to design, analyze, and think about algorithms.
- we will not cover anything you could figure out on your own.

# Algorithms for Fibonacci Numbers

“0, 1, 1, 2, 3, 5, 8, 13, 21, ...”

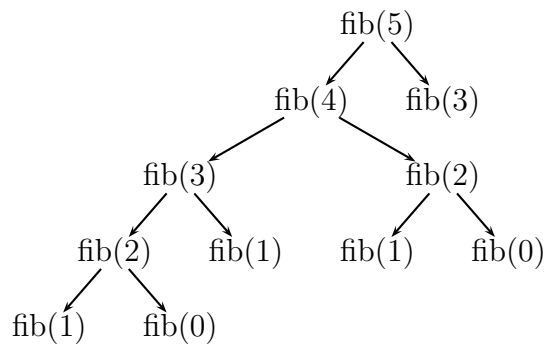
**Question:** recursive alg?

**Algorithm:** Recursive Fibonacci

fib(k):

1. if  $k \leq 1$  return  $k$
2. (else) return  $\text{fib}(k-1) + \text{fib}(k-2)$

**Example:**



## Analysis

“what is runtime?”

Let  $T(k)$  = number of calls to fib

$$T(0) = T(1) = 1$$

$$\begin{aligned}
 T(k) &= T(k-1) + T(k-2) \\
 &\geq 2T(k-2) \\
 &\geq 2 \times 2T(k-4) \\
 &\geq \underbrace{2 \times 2 \times \dots \times 2}_{k/2 \text{ times}} \times 1 \\
 &= 2^{k/2}
 \end{aligned}$$

**Conclusion:** at least “exponential time”!

## Remembering Redundant Computation (memoization)

**Idea:** remember redundant computation (memoize)

**Algorithm:** Memoized Recursive Fibonacci

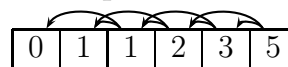
fib-helper(k)

1. if  $\text{memo}[k] \geq 0$  return  $\text{memo}[k]$
2. (else) return  $\text{fib-helper}(k-1) + \text{fib-helper}(k-2)$

fib(k)

1.  $\text{memo} = \text{new int}[k];$
2.  $\text{memo}[0] = 0, \text{memo}[1] = 1, \text{memo}[2, \dots, k] = -1;$
3. return  $\text{fib-helper}(k)$

**Example:**



## Analysis

- cost to fill in each entry: 1 additions.
- number of entries:  $k$
- total cost:  $T(k) = k$  additions.

**Conclusion:** “linear time”.

**Note:** memoizing redundant computation is essential part of “dynamic programming”.

## Iterative Algorithm

**Algorithm:** Iterative Memoized Fibonacci

fib(k):

1. `memo = new int[k];`
2. `memo[0] = 0, memo[1] = 1`
3. for `i = 2..k`

$$\text{memo}[i] = \text{memo}[i-1] + \text{memo}[i-2]$$
4. return `memo[k]`

**Question:** Can we compute fib with less memory (space)?

**Algorithm:** Iterative Fibonacci

fib(k):

1. `last[0] = 0, last[1] = 1;`
2. for `i = 2..k`
  - (a) `tmp = last[1]`
  - (b) `last[1] = last[0] + last[1]`
  - (c) `last[0] = tmp`
3. return `last[1]`

**Question:** faster alg?

## Fast Fibonacci

**Note:** algorithm operates on `last` like a matrix multiply

fib(k):

1.  $z = [0 \ 1]; A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$
2. multiply  $z \times \underbrace{A \times A \cdots \times A}_{k-2 \text{ times}}$
3. return  $z[1]$

**Note:** just need to compute  $z \times A^{k-2}$

## Exponentiation

“compute  $A^k$ ”

**Note:** If  $k = k_1 + k_2$  then  $A^k = A^{k_1} A^{k_2}$

- compute  $A^{k_1}$  and  $A^{k_2}$  and multiply.
- if  $k_1 = k_2$  then redundant computation

**Idea:** factor  $A^k = (A^{k/2})^2 \times A^{k \% 2}$

**Algorithm:** Repeated Squaring

1. if  $k = 1$  return  $A$
2.  $k' = \lfloor k/2 \rfloor$ .
3.  $B = \text{repeated-square}(A, k')$ .
4. if  $k$  odd  
return  $B \times B \times A$
5. else  
return  $B \times B$

## Analysis

Let  $T(k)$  = number of multiplies.

$$T(1) = 0$$

$$\begin{aligned} T(k) &= T(k/2) + 2 \\ &= T(k/4) + 2 + 2 \\ &= \underbrace{2 + 2 + 2 \cdots 2}_{\log k \text{ times}} \\ &= 2 \log k \end{aligned}$$

**Note:** finding subproblems is important part of “divide and conquer”

**Algorithm:** Fibonacci numbers via repeated squaring

fib(k):

1.  $A = \begin{bmatrix} 0 & 1 \\ 1 & 1 \end{bmatrix}$ .
2.  $z = [0 \ 1] \times \text{repeated-square}(A, k-2)$ .
3. return  $z[1]$ .

## Analysis

$2 \log k$  2x2 matrix multiplies.

## Conclusions

- runtime analysis
- memoization
- divide and conquer