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,
  - parlementary 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, analize, 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

\[
\text{fib}(k): \\
\begin{align*}
&1. \text{ if } k \leq 1 \text{ return } k \\
&2. \text{ (else) return } \text{fib}(k - 1) + \text{fib}(k - 2)
\end{align*}
\]

Example:

\[
\begin{array}{c}
\text{fib}(5) \\
\text{fib}(4) \quad \text{fib}(3) \\
\text{fib}(3) \quad \text{fib}(2) \\
\text{fib}(2) \quad \text{fib}(1) \quad \text{fib}(1) \quad \text{fib}(0) \\
\text{fib}(1) \quad \text{fib}(0)
\end{array}
\]

Analysis

“what is runtime?”

Let \( T(k) = \text{number of calls to fib} \)

\[
T(0) = T(1) = 1 \\
T(k) = T(k - 1) + T(k - 2) \\
\geq 2T(k - 2) \\
\geq 2 \times 2T(k - 4) \\
\geq 2 \times 2 \times \cdots \times 2 \times 1 \\
\geq 2^{k/2} \times 1 \\
= 2^{k/2}
\]

Conclusion: at least “exponential time”!

Remembering Redundant Computation (memoization)

Idea: remember redundant computation (memoize)

Algorithm: Memoized Recursive Fibonacci

\[
\text{fib-helper}(k): \\
\begin{align*}
&1. \text{ if } \text{memo}[k] \geq 0 \text{ return } \text{memo}[k] \\
&2. \text{ (else) return } \text{fib-helper}(k - 1) + \text{fib-helper}(k - 2)
\end{align*}
\]

Example:

\[
\begin{array}{cccccc}
0 & 1 & 1 & 2 & 3 & 5
\end{array}
\]

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

\[
\text{fib}(k): \\
\begin{align*}
&1. \text{ memo } = \text{ new int}[k]; \\
&2. \text{ memo}[0] = 0, \text{ memo}[1] = 1, \text{ memo}[2, . . . , k] = -1; \\
&3. \text{ return } \text{fib-helper}(k)
\end{align*}
\]

Example:

\[
\begin{array}{cccccc}
0 & 1 & 1 & 2 & 3 & 5
\end{array}
\]
1. memo = new int[k];
2. memo[0] = 0, memo[1] = 1
3. for i = 2..k
   memo[i] = memo[i-1] + 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 \( z \times A \times A \cdots \times A \) \( k - 2 \) 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 \( A^k = (A^{k/2})^2 \times A^{k \mod 2} \)

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.
\[
\begin{align*}
T(1) &= 0 \\
T(k) &= T(k/2) + 2 \\
&= T(k/4) + 2 + 2 \\
&= 2 + 2 + 2 \cdots 2 \\
&= 2 \log k
\end{align*}
\]

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