CSC 212: Data Structures and Abstractions
Introduction to Analysis of Algorithms

Marco Alvarez
Department of Computer Science and Statistics
University of Rhode Island

Spring 2018

Analysis of Algorithms

Algorithm

“Any well-defined computational procedure that takes some value, or set of values, as input and produces some value, or set of values, as output.”

[ Cormen et al., Introduction to Algorithms, 3rd Ed.]

Amount of resources necessary to execute an algorithm?

- **Time Complexity** (running time)
- **Space Complexity** (memory)
- Resources typically depend on **input size**

Why Analysis of Algorithms?

- Classify algorithms/problems
- Predict performance/resources
- Provide guarantees
- Understand underlying principles

Administrativia

- **Readings**
  - links posted on the course website, especially OpenDSA

- **Discussion Sections**
  - average attendance: 30 students

- **Piazza**
  - ask, answer, review, …
Time Complexity

Empirical Analysis vs Mathematical Model

Analyzing running time

<table>
<thead>
<tr>
<th>Empirical Analysis</th>
<th>Mathematical Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run algorithm</td>
<td>Analyze algorithm</td>
</tr>
<tr>
<td>Measure actual time</td>
<td>Develop Model</td>
</tr>
</tbody>
</table>

Empirical Analysis

- Implement algorithm
- Run on different input sizes
- Record actual running times
- Calculate hypothesis
- Predict and validate
**Timing Algorithms**

**Recursive**

```c
1 uint64_t fib_rec(uint16_t n) {
2     if (n < 2)
3         return n;
4     return fib_rec(n-1) + fib_rec(n-2);
5 }

or just ...

1 uint64_t fib_rec(uint16_t n) {
2     return (n < 2) ? n : fib_rec(n-1) + fib_rec(n-2);
3 }
```

**Iterative**

```c
1 uint64_t fib_iter(uint16_t n) {
2     uint64_t sum;
3     uint64_t prev[] = {0, 1};
4     if (n < 2) {
5         return n;
6     }
7     for (uint16_t i = 2; i <= n; i++) {
8         sum = prev[0] + prev[1];
9         prev[0] = prev[1];
10        prev[1] = sum;
11     }
12     return sum;
13 }
```

\[ F_0 = 0 \]

\[ F_1 = 1 \]

\[ F_n = F_{n-1} + F_{n-2} \]

0 1 1 2 3 5 8 13 21 34 …
Timing …

1 void time_func(uint16_t n, const char *name) {
2     uint64_t val;
3     Clock::time_point tic, toc;
4     if (!strcmp(name, "Rec")) {
5         tic = Clock::now();
6         val = fib_rec(n);
7         toc = Clock::now();
8     }
9     if (!strcmp(name, "Iter")) {
10        tic = Clock::now();
11        val = fib_iter(n);
12        toc = Clock::now();
13     }
14     std::cout << name << " fib(" << n << ")": " << std::fixed <<
15        std::setprecision(4) << Seconds(toc-tic).count() << " sec."
16     std::cout << std::endl;
17 }
18
19 int main(int argc, char **argv) {
20     uint16_t n = (uint16_t) atoi(argv[1]);
21     time_func(n, argv[2]);
22 }

Comparing

Limitations of Empirical Analysis

- Requires implementing the algorithm
  - may be difficult
  - implementation details also play a role
- Variations in HW, SW, and OS affect analysis
Analyzing running time

Empirical Analysis
- Run algorithm
- Measure actual time

Mathematical Model
- Analyze algorithm
- Develop Model

Mathematical Model
- High-level analysis — no need to implement
- Independent of HW/SW
- Based on counts of elementary operations
  - additions, multiplications, comparisons, etc
  - exact definition not important but must be relevant to the problem

Recursion Tree
- Visualize call structure, for example: fib_rec(5)

Problem?
- Same computation multiple times
- Let \( c_n \) be the number of calls to fib_rec()

\[
\begin{align*}
  c_0 &= 1 \\
  c_1 &= 1 \\
  c_2 &= c_1 + c_0 + 1 = 3 \\
  c_3 &= c_2 + c_1 + 1 = 5 \\
  c_4 &= c_3 + c_2 + 1 = 9 \\
  c_5 &= c_4 + c_3 + 1 = 15 \\
  c_6 &= c_5 + c_4 + 1 = 25 \\
  c_7 &= c_6 + c_5 + 1 = 41 \\
\end{align*}
\]

Exponential Time!

\( c_n \) at least doubles every other time, i.e., \( c_n > 2^{n/2} \)
Math Review

- Summations
- Powers
- Logarithms
- Proof Techniques
- Basic Probability and Combinatorics