CSCE 3110
Data Structures and Algorithms

Algorithm Analysis I

Reading: Weiss, chap. 2

Credits of some slides go to S. Skiena at SUNY Stony Brook.

Content

- Algorithm Analysis
  - Empirical tests
  - Mathematical analysis
- Asymptotic notation
  - Big Oh
  - Big Omega
  - Big Theta
  - Little Oh
  - Little Omega
- Asymptotic Dominance

Problem Solving: Main Steps

1. Problem definition
2. Algorithm design / Algorithm specification
3. Algorithm analysis
4. Implementation
5. Testing
6. [Maintenance]

Problem Solving: Main Steps

1. Problem definition
2. Algorithm design / Algorithm specification
3. Algorithm analysis
4. Implementation
5. Testing
6. [Maintenance]

Our focus

1. Problem Definition

- What is the task to be accomplished?
  - Exa. 1: Calculate the average of the grades for a given student
  - Exa. 2: Detect all human faces in an image (object detection problem)

- What are the time/space/speed/accuracy requirements ?
  - For Exa. 1: Calculate the average grade in 0.001s (there may be more than 10,000 students checking grades at the same time) – time
  - For Exa. 2: Detection accuracy should be above 0.9 – accuracy

2. Algorithm Design / Specifications

- Algorithm: Finite set of instructions that, if followed, accomplishes a particular task.
- Describe: in natural language/pseudo-code/flow chart/etc.
- Criteria to follow:
  - **Input:** Zero or more quantities (externally produced)
  - **Output:** One or more quantities
  - **Definiteness:** Clarity, precision of each instruction
  - **Finiteness:** The algorithm has to stop after a finite (may be very large) number of steps
  - **Effectiveness:** Each instruction has to be basic enough and feasible

3. Algorithm Analysis

We will get back to it later

4, 5, 6: Implementation, Testing, Maintenance

- Implementation
  - Decide on the programming language to use
    - C/C++/Java/Python/Matlab/Assembly, etc.
    - Write clean, well documented code

- Test, test, test

- Integrate feedback from users, fix bugs, ensure compatibility across different versions → Maintenance

3. Algorithm Analysis

- Goal
  - Predict the resources that an algorithm requires

3. Algorithm Analysis

- Goal
  - Predict the resources that an algorithm requires

- What kind of resources?
  - Memory
  - Communication bandwidth
  - Hardware requirements
  - Running time
  - …

3. Algorithm Analysis

- Goal
  - Predict the resources that an algorithm requires

- What kind of resources?
  - Memory
  - Communication bandwidth
  - Hardware requirements
  - Running time
  - …
- Approaches
  - Empirical tests
  - Mathematical analysis

Algorithm Analysis - Empirical Tests

- Steps
  - Implement algorithm in a given programming language
  - Measure running time with several inputs
  - Infer running time for any (possible) inputs

Algorithm Analysis - Empirical Tests

- Steps
  - Implement algorithm in a given programming language
  - Measure running time with several inputs
  - Infer running time for any (possible) inputs
- Pros:
  - No math, straightforward method
- Cons:
  - Not reliable, heavily dependent on
    - The sample inputs
    - Programming language and environment

Algorithm Analysis - Mathematical Analysis

- Use math to estimate the running time of an algorithm
  - almost always dependent on the size of the input
  - Alg. 1 running time is n^2 for an input of size n
  - Alg. 2 running time is $n \times \log(n)$ for an input of size n

Algorithm Analysis - Mathematical Analysis

- Use math to estimate the running time of an algorithm
  - almost always dependent on the size of the input
  - Alg. 1 running time is n^2 for an input of size n
  - Alg. 2 running time is $n \times \log(n)$ for an input of size n
- Pros:
  - formal, rigorous
  - no need to implement algorithms
  - machine-independent
- Cons:
  - math knowledge (may be extremely complex)

Algorithm Analysis - How?

- The time taken by an algorithm depends on the input
  - Searching in a sequence of 1000 numbers takes longer than in a sequence of 3 numbers
  - Some algorithms take different amounts of time on two inputs of the same size
    - Searching in a sorted sequence
- Input size (problem size)
  - depends on the problem being studied
  - usually, number of items in the input
  - Exa. 1: if searching in a sequence, number of elements
  - Exa. 2: graph algorithms: input size in terms of vertices and edges

Algorithm Analysis - An Example

- Find the position of the smallest number in a sequence

Algorithm Analysis - An Example

- Find the position of the smallest number in a sequence

Input: A sequence S of n numbers $\{a_1, a_2, \ldots, a_n\}$, where $a_1 != a_2 != \ldots != a_n$

Output: *pos*, the position of the smallest element in the input sequence

```
pos ← 0;
min ← S[0];
i ← 1;
while i < n do
    if S[i] < min then
        pos ← i;
        min ← S[i];
    end
    i ← i + 1;
end
return pos
```

Algorithm Analysis - An Example

- Find the position of the smallest number in a sequence

Input: A sequence S of n numbers $\{a_1, a_2, \ldots, a_n\}$, where $a_1! = a_2! = \ldots! = a_n$

Output: *pos*, the position of the smallest element in the input sequence

```
pos ← 0;
min ← S[0];
i ← 1;
while i < n do
    if S[i] < min then
        pos ← i;
        min ← S[i];
    end
    i ← i + 1;
end
return pos
```

Assume:

Ta: time for assigning operation

Tc: time for comparing two numbers

Ti: increase *i* by 1

Algorithm Analysis - An Example

- Find the position of the smallest number in a sequence

Input: A sequence S of n numbers $\{a_1, a_2, \ldots, a_n\}$, where $a_1 != a_2 != \ldots != a_n$

Output: pos, the position of the smallest element in the input sequence

```
pos ← 0;
min ← S[0];
i ← 1;
while i < n do
    if S[i] < min then
        pos ← i;
        min ← S[i];
    end
    i ← i + 1;
end
return pos
```

Assume:

Ta: time for assigning operation

Tc: time for comparing two numbers

Ti: increase *i* by 1

Shortest running time?

Longest running time?

Algorithm Analysis - An Example

- Worse case (longest running time)

  $$(2Ta + 2Tc + Ti)\mathrm{n}+(Ta - Tc - Ti)$$

- Best case (shortest running time)

  $$(2Tc + Ti)\mathrm{n}+(3Ta - Tc - Ti)$$

Algorithm Analysis - An Example

- Best case analysis

- Worst case analysis

- Average case analysis

Algorithm Analysis - An Example

- Best case analysis
  - shortest running time for any input of size n
  - often meaningless, one can easily cheat
- Worst case analysis

- Average case analysis

Algorithm Analysis - An Example

- Best case analysis
  - shortest running time for any input of size n
  - often meaningless, one can easily cheat
- Worst case analysis
  - longest running time for any input of size n
  - it guarantees that the algorithm will not take any longer
  - provides an upper bound on the running time
  - worst case occurs often when searching for an item that does not exist
- Average case analysis

Algorithm Analysis - An Example

- Best case analysis
  - shortest running time for any input of size n
  - often meaningless, one can easily cheat
- Worst case analysis
  - longest running time for any input of size n
  - it guarantees that the algorithm will not take any longer
  - provides an upper bound on the running time
  - worst case occurs often when searching for an item that does not exist
- Average case analysis
  - running time averaged for all possible inputs
  - it is hard to estimate the probability of all possible inputs

Our Position on Complexity Analysis

- Generally speaking, we will use the worst-case complexity as our preferred measure of algorithm efficiency
- Worst-case analysis is generally easy to do, and "usually" reflects the average case. Assume I am asking for worst case analysis unless otherwise specified!
- Randomized algorithms are of growing importance and require an average-case type analysis to show off their merits.

Algorithm Analysis - An Example

- Worse case

$$(2T_a+2T_c+T_i)n+(T_a-T_c-T_i)$$

- Best case

$$(2T_c+T_i)n+(3T_a-T_c-T_i)$$

Algorithm Analysis - An Example

- Worse case

$$(2T_a+2T_c+T_i)n+(T_a-T_c-T_i)$$

- Best case

$$(2T_c+T_i)n+(3T_a-T_c-T_i)$$

This kind of analysis is tedious, as it depends on *Ta*, *Tc* and *Ti*.

Algorithm Analysis - An Example

- Worse case

$$(2Ta+2Tc+Ti)n+(Ta-Tc-Ti)$$

- Best case

$$(2Tc+Ti)n+(3Ta-Tc-Ti)$$

This kind of analysis is tedious, as it depends on *Ta*, *Tc* and *Ti*.

As number of operation increases, it will become more complex to analyze the running time!

Random-Access Machine (RAM)

In order to predict running time, we need a (simple) computational model: the Random-Access Machine (RAM)

- Instructions are executed sequentially
  - No concurrent operations
- Each basic instruction takes a constant amount of time
  - Arithmetic operation: add, subtract, multiply, divide, remainder, shift left/shift right, etc.
  - Data movement: load, store, copy
  - Loops and subroutine calls are not simple operations. They depend upon the size of the data and the contents of a subroutine. "Sort" is not a single step operation.
- Each memory access takes exactly 1 step

We measure the run time of an algorithm by counting the number of steps

Running time

- on a particular input, number of primitive operations (steps, instructions) executed
- assume that all primitive operations have the same constant cost
- if calling a subroutine, the actual cost must be used

Algorithm Analysis – An example

- Simplifications:
  - All basic instructions have constant time, $Ta = Tc = Ti = T$
  - Sometimes, look only at the leading term
    - drop lower-order terms
    - Ignore the constant coefficient in the leading term

Algorithm Analysis – An example

- Simplifications:
  - All basic instructions have constant time, $Ta = Tc = Ti = T$
  - Sometimes, look only at the leading term
    - drop lower-order terms
    - Ignore the constant coefficient in the leading term

- Worst case

$$(2Ta+2Tc+Ti)\text{n}+(Ta-Tc-Ti) \longrightarrow ?$$

- Best case

$$(2Tc+Ti)\text{n}+(3Ta-Tc-Ti) \longrightarrow ?$$

Algorithm Analysis – An example

- Simplifications:
  - All basic instructions have constant time, $Ta = Tc = Ti = T$
  - Sometimes, look only at the leading term
    - drop lower-order terms
    - Ignore the constant coefficient in the leading term

- Worst case

$$(2Ta+2Tc+Ti)n+(Ta-Tc-Ti) \rightarrow T(5n-2) \rightarrow n$$

- Best case

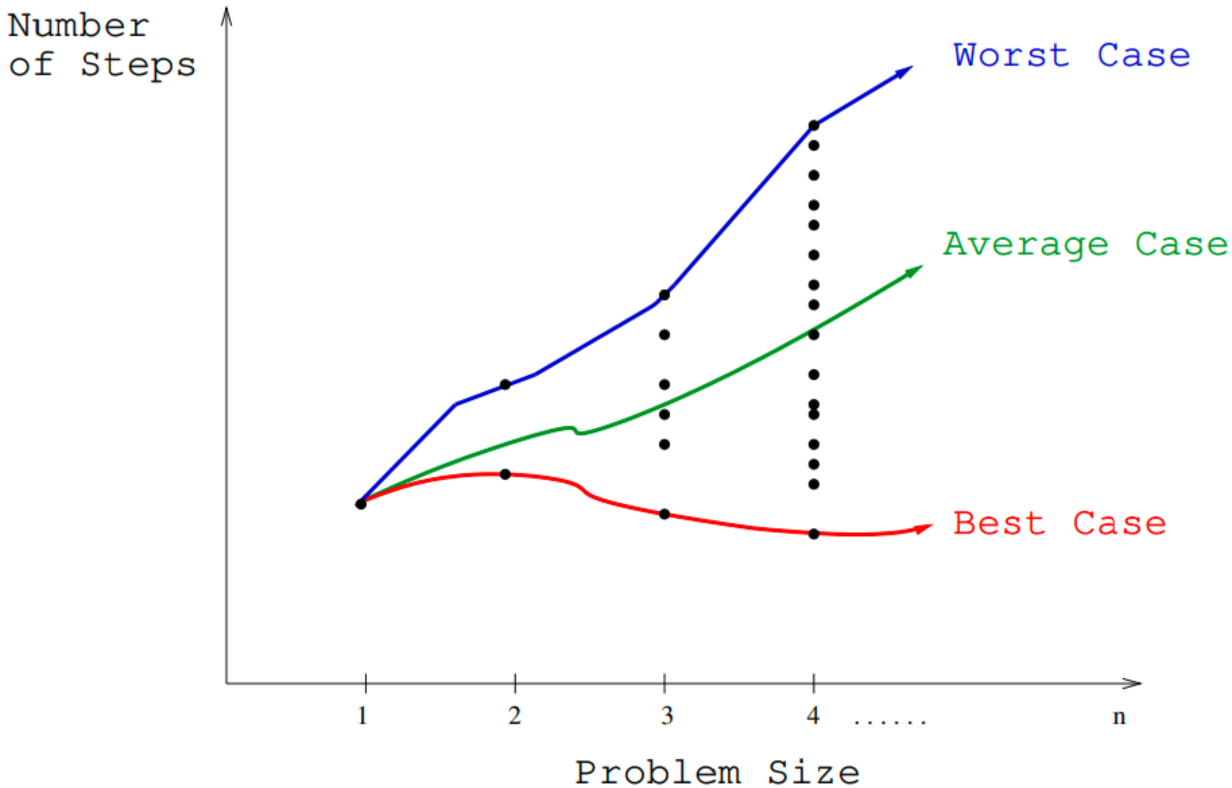
$$(2Tc+Ti)n+(3Ta-Tc-Ti) \rightarrow T(3n+1) \rightarrow n$$

The RAM Model of Computation

- The worst-case (time) complexity of an algorithm is the function defined by the maximum number of steps taken on any instance of size n
- The best-case complexity of an algorithm is the function defined by the minimum number of steps taken on any instance of size n
- The average-case complexity of the algorithm is the function defined by an average number of steps taken on any instance of size n

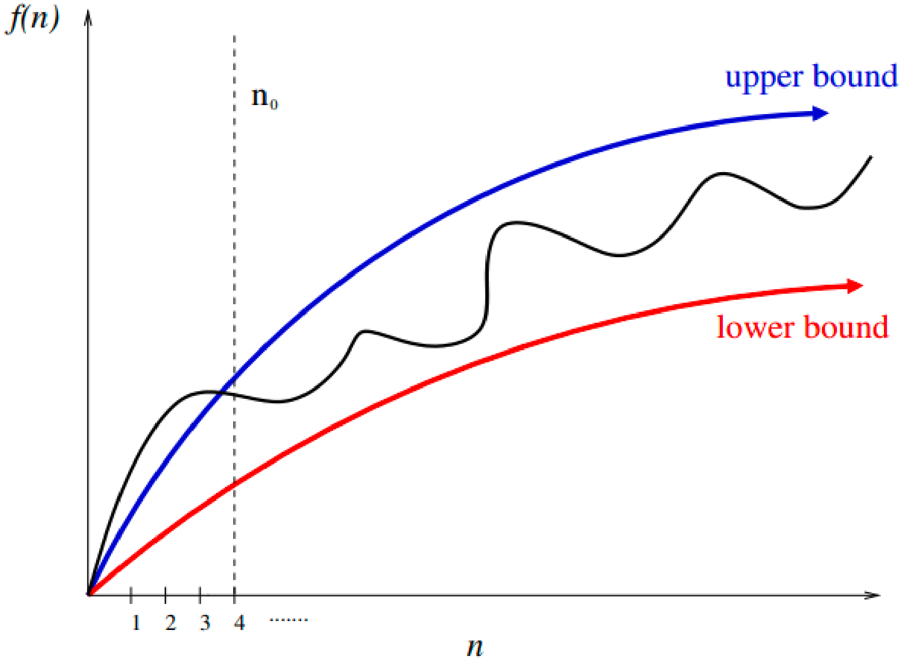
The RAM Model of Computation

- Each of the three complexities defines a numerical function: **time vs. size!**



Exact Analysis is Hard!

- Best, worst, and average case are difficult to deal with because the precise function details are very complicated



- It easier to talk about upper and lower bounds of the function.

Next Class

Algorithm Analysis I (cont.)

Reading: Weiss, chap. 2