Algorithm Analysis 01

Computer Science Department
Eastern Washington University
Yun Tian (Tony) Ph.D.

Today

- Introduction
- How to tell an algorithm is better than another?
  - Approach 1 – experimental study
  - Approach 2 – Theoretical Analysis

Introduction

```
public boolean contains(E target) {
  for (int i = 0; i < size; i++) {
    if (data[i].equals(target)) {
      return true;
    }
  }
  return false;
}
```

- Hard to analyze because of the if statement.

Best-case Analysis

- Tells us how fast the program runs if we get really lucky about the data.
  - contains() could find the data at the first index.
  - It takes only one basic operation.
  - Complexity of O(1)

Worst-case Analysis

- Contains()
- We have to go through all elements in the ArrayList.
- This means assuming that target is not in the ArrayList, giving a running time of O(n).

Average-case Analysis

- Contains()
- Requires that we make some assumption about what the “average” data set looks like.
- We have to consider all cases for the input ArrayList.
- It is quite challenging
  - Require us to define a probability distribution on the set of inputs.

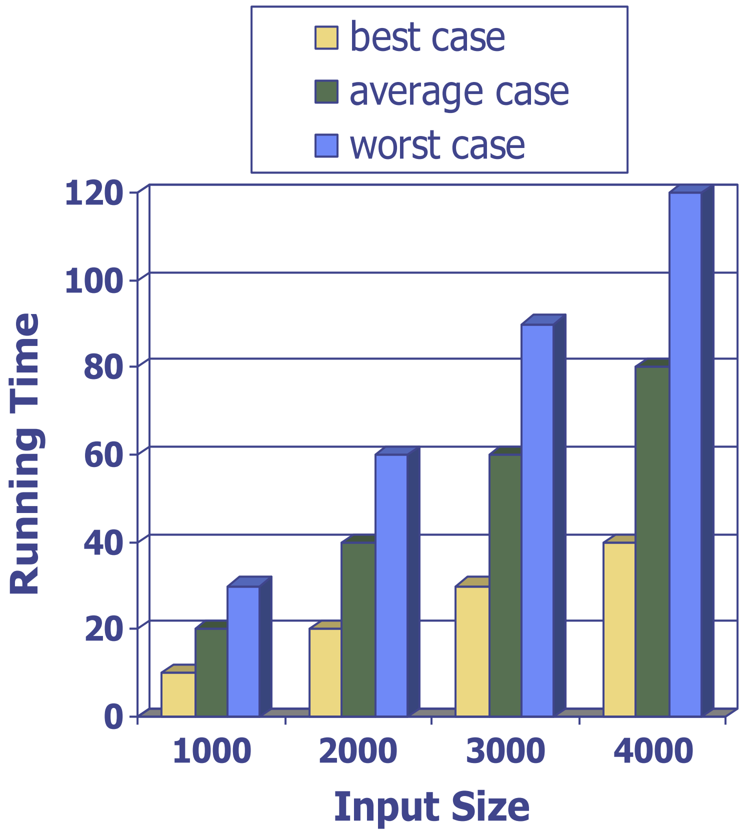
Average-case Analysis

- Contains()
- The average running time is:

$$\sum_{\text{events}} \langle \text{probability of event occuring} \rangle \cdot \langle \text{running time if event occurs} \rangle$$

- Depends on the input distribution, the running time of an algorithm can be anywhere between the worst-case and the best-case time.

Best,Average and Worst Case



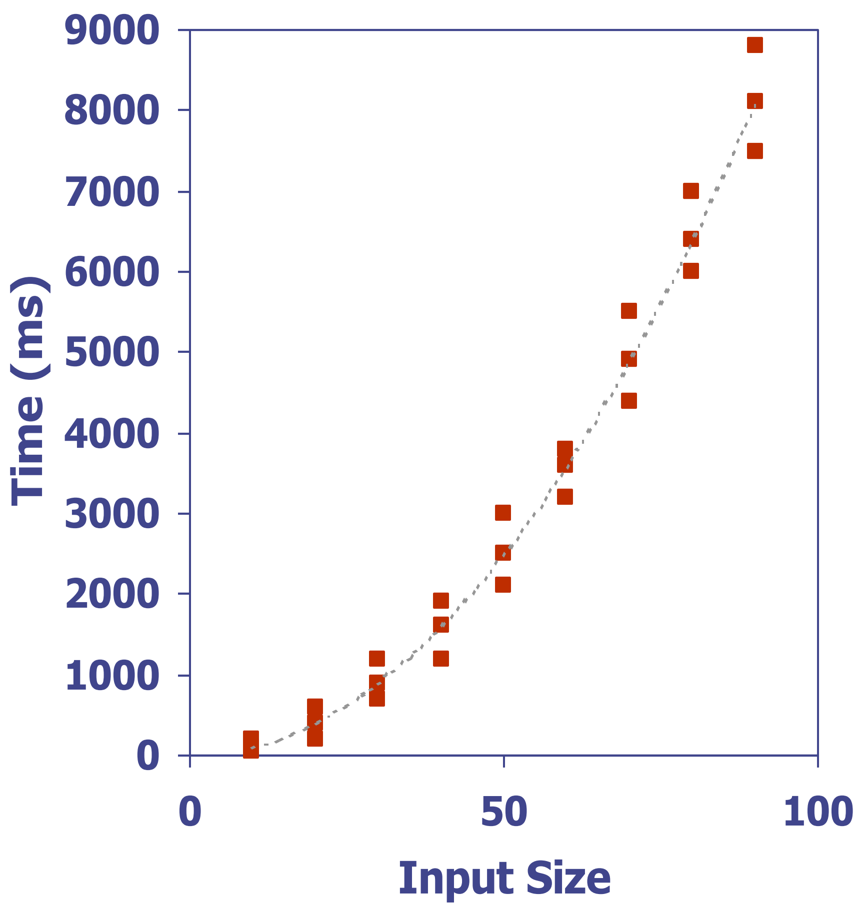
Worst Case

- The running time of an algorithm typically grows with the input size.
  - Except for algorithms that have constant running time.
- Average case time is often difficult to determine.
- We focus on the worst case running time.
  - Easier to analyze
  - Crucial to applications such as games, finance and robotics.

Approach 1: Experimental Studies

- Write a program implementing the algorithm
- Run the program with inputs of varying size and composition
- Use a method like System.currentTimeMillis() to get an accurate measure of the actual running time
- Plot the results

Experimental Studies



Approach 1: Experimental Studies

- Limitations of Experiments
  - It is necessary to implement the algorithm, which may be difficult
  - Results may not be indicative of the running time on other inputs not included in the experiment.
  - In order to compare two algorithms, the same hardware and software environments must be used.

Approach 2: Theoretical Analysis

- Uses a high-level description of the algorithm instead of an implementation.
- Characterizes running time as a function of the input size n.
- Takes into account all possible inputs.
- Allows us to evaluate the speed of an algorithm independent of the hardware/software environment.

Approach 2: Theoretical Analysis

- Primitive Operations in Algorithms
  - Basic computations performed by an algorithm
  - Identifiable in pseudo code.
  - Largely independent from the programming language
  - Correspond to a low-level instruction with an constant execution time.

Approach 2: Theoretical Analysis

- Example of Primitive Operations in Algorithms
  - Evaluating an expression
  - Assigning a value to a variable
  - Indexing into an array
  - Calling a method (if the method is independent of program size)
  - Returning from a method.
  - Compare two numbers.
  - Following an object reference.

Approach 2: Theoretical Analysis

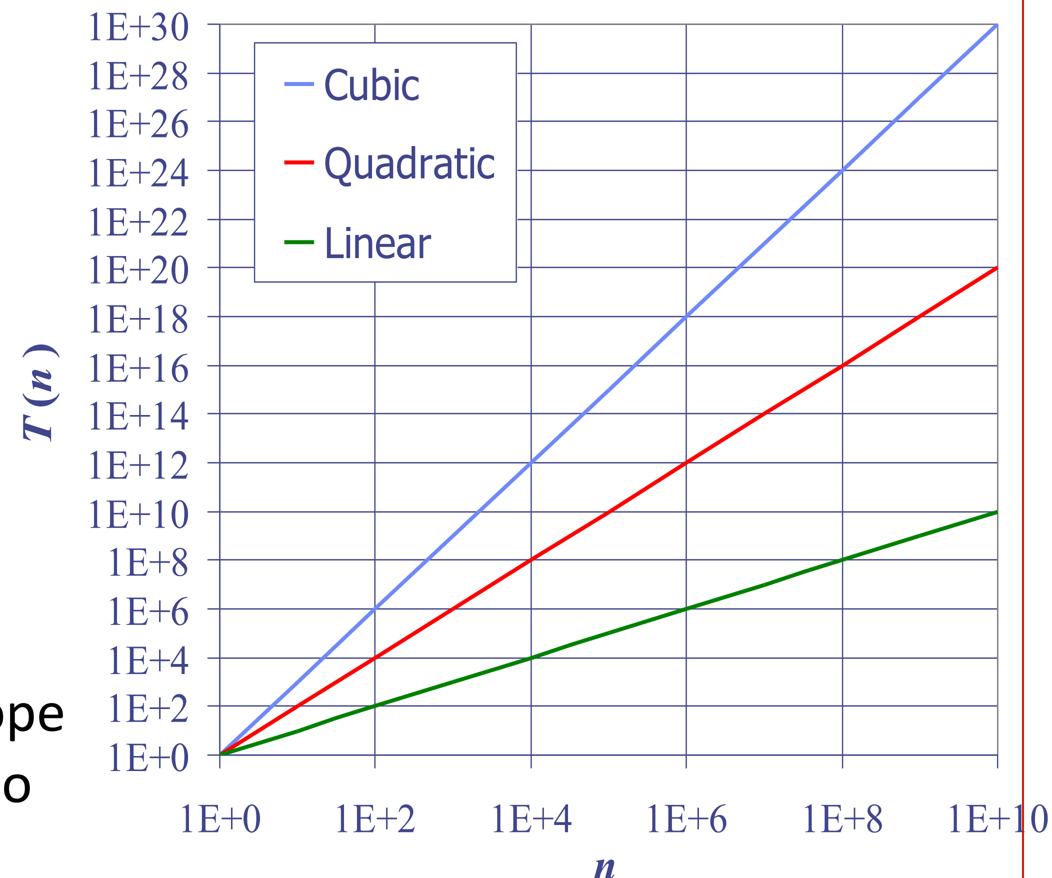
- Basic Idea
  - Instead of trying to determine the specific execution time of each primitive operation,
  - But, we will simply ***count*** how many primitive operations are executed and use this number ***t*** as a measure of the running time of the algorithm.

Approach 2: Theoretical Analysis

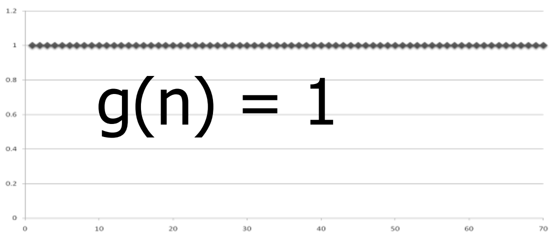
- Basic Idea
  - In this approach, we assume the running times of different primitive operations will be fairly similar.
    - Thus the number **t** is proportional to the actual running time of that algorithm.
    - Where **t** is the number of primitive operations the algorithm performs.

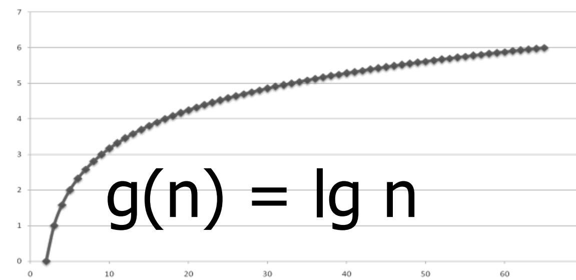
Functions in Theoretical Analysis

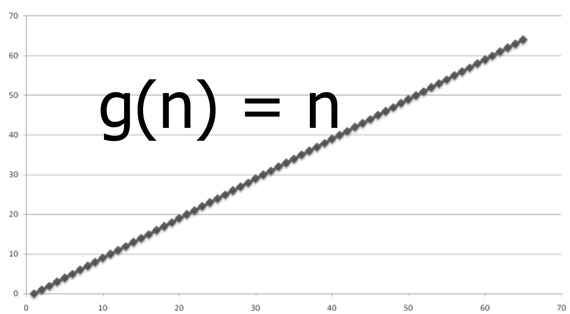
- Seven functions that often appear in algorithm analysis:
  - Constant ≈ 1
  - Logarithmic ≈ $\log n$
  - Linear ≈ n
  - N-Log-N ≈ $n \log n$
  - Quadratic ≈ n^2
  - Cubic ≈ n^3
  - Exponential ≈ 2^n
- In a log-log chart, the slope of the line corresponds to the growth rate

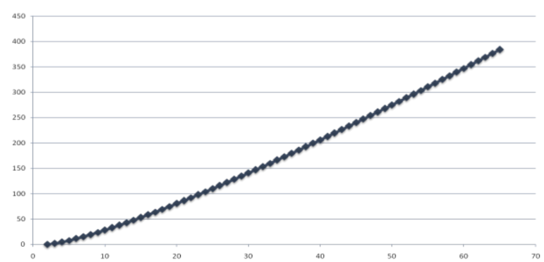


Functions in Theoretical Analysis



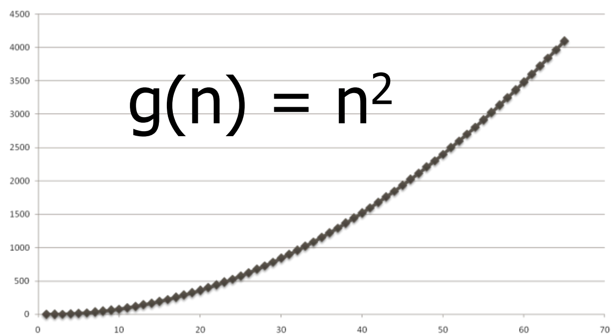


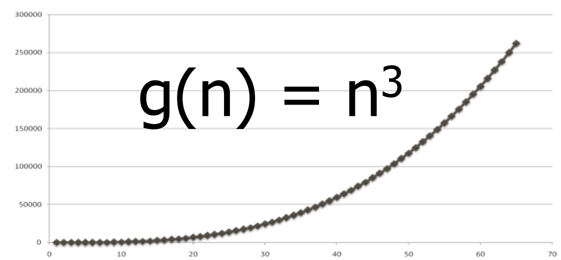


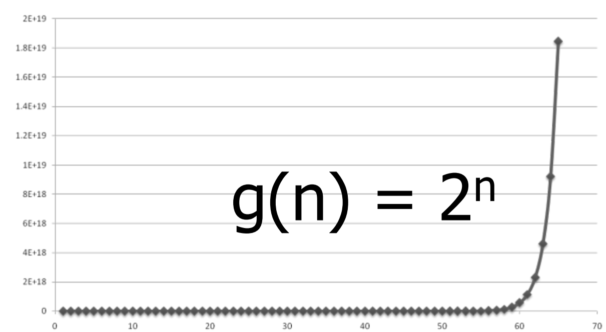


g(n) = n lg n

Functions in Theoretical Analysis







Counting Primitive Operations

- By inspecting the pseudocode or the implementation, we can determine the maximum number of primitive operations executed by an algorithm, as a function of the input size.

Counting Primitive Operations

```
public String toString() {
        String result = "";  (1) + (1)        1        N+1            N
        Node cur;(1)
        for( cur = this.head.next; cur!= this.head; cur = cur.next) {
                results += cur.data + "\n";  //(2N)
        }
        return result; //(1)
}//if the size of this list is N
```

Total   4N + 6. Here, f(n) = 4N + 6 is the growth rate function.

Take Home Summary

- Two approaches to analyze the algorithm
- Better to use theoretical analysis
- Seven functions
- Growth Rate Function(GRF)

Next Class

- Big-Oh Notation