Unit 06: Runtime Analysis

Anthony Estey

CSC 115: Fundamentals of Programming II

University of Victoria

Unit 06 Overview

- Related Reading:
  - Textbook Chapter 4
- Learning Objectives: (You should be able to…)
  - understand the methodology we will use in this course to analyze algorithms and data structures, based on the size of the input data, *n*
  - compare and contrast different implementations based on an analysis of their asymptotic runtimes

Analysis

- What is the “right” way to analyze the speed of an algorithm?

- **Idea:** We could just execute it and see how long it takes to complete
- **Problem:** It is difficult to setup a controlled environment to accurately compare two different algorithms
  - The hardware environment will affect the results (processor, memory, etc.)
  - and the software environment will too (OS, programming language, compiler)
  - So this just isn’t feasible in a course like this this

- Instead, we can simply count the number of operations that need to be executed when running the algorithm

Analysis

- What is the “right” way to analyze the speed of an algorithm?

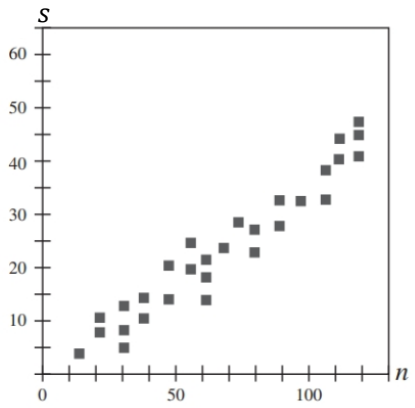
- **Idea:** Count the number of operations that occur!
- **Problem:** One algorithm might work better than another under different circumstances!

- It is best to perform several experiments on many different test inputs and with test inputs of various sizes, and in particular think about how one algorithm scales as our input size increases

Analysis

- What is the “right” way to analyze the speed of an algorithm?

- Proposed methodology:
  - Associate each algorithm a function $f(n)$ that characterizes the running time of the algorithm based on the number of Java statements that would need to be executed in terms of the input size n
- What is the best way to interpret / visualize the results?
  - Plot on a graph:
    - x-coordinate represents the input size, n
    - y-coordinate represents the number of statements, s



Assumptions

- Any single Java statement takes the same amount of time to run
- A method call's runtime is measured by the total statements executed inside the method's body
- A loop's runtime is N times the runtime of the statements inside the method's body (where N is the number of times the loop repeats)

Examples

```
statement1;
statement2;
statement3;
```

3

```
for (int i = 1; i <= N; i++) {
   statement4;
}
```

N

```
for (int i = 1; i <= N; i++) {
   statement5;
   statement6;
   statement7;
}
```

$3N$

Another example

```
public static void method1(int N) {

   for (int i = 1; i <= N; i++) {
      for (int j = 1; j <= N; j++) {
         statement1;
      }
   }

   for (int i = 1; i <= N; i++) {
      statement2;
      statement3;
      statement4;
      statement5;
   }
}
```

when $i = 1$:

$j = \cancel{1}\ \cancel{2}\ \cancel{3}\ \ldots\ N$

when $i = 2$:

$j = \cancel{1}\ \cancel{2}\ \cancel{3}\ \ldots\ N$

when $i = 3$:

$\vdots$

when $i = N$:

statement1 executes N times for each of the N inner loops

Another example

```
public static void method1(int N) {

   for (int i = 1; i <= N; i++) {
      for (int j = 1; j <= N; j++) {
         statement1;
      }
   }

   for (int i = 1; i <= N; i++) {
      statement2;
      statement3;
      statement4;
      statement5;
   }
}
```

N^2

$4N$

$N^2 + 4N$

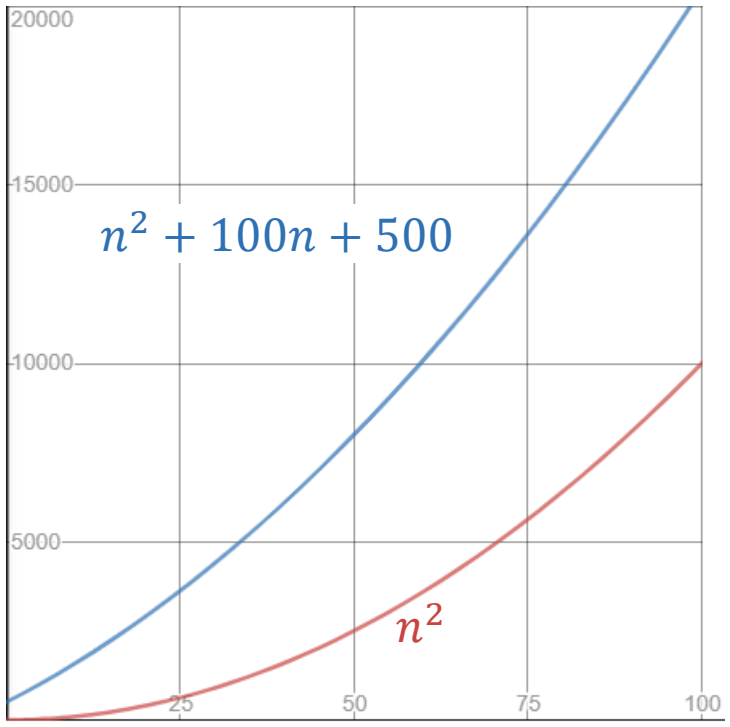
Algorithm Growth Rates

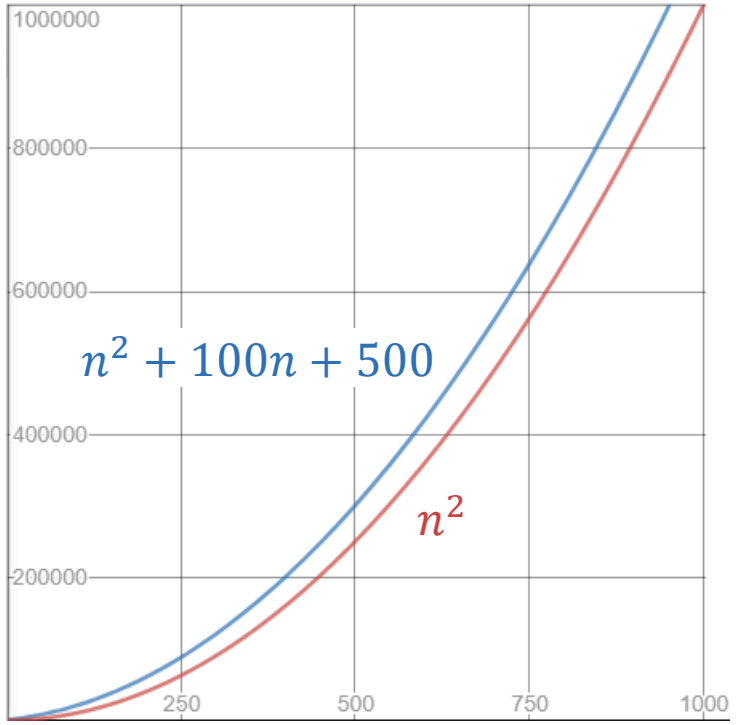
- At this point, we have seen that we can measure runtime based on the number of statements executed in proportion to the input size n
- We define the **growth rate** as the change in runtime as n changes.

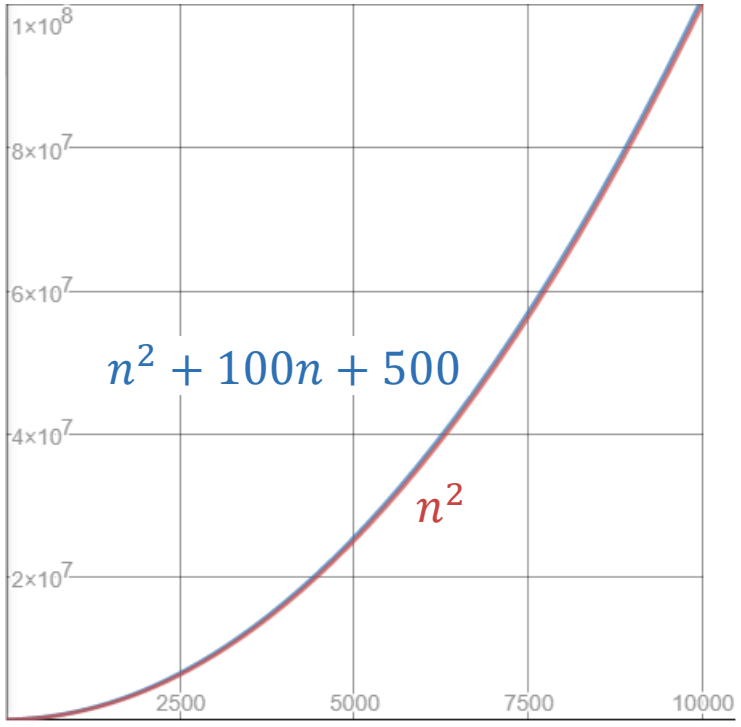
- For example, let's say we have computed the number of statements in method relative to n to be: $2n^2 + 12n + 5$
  - What can we say about the runtime for *very large* values of n
  - We will see that we can ignore some of the lower terms, as the growth rate will be dominated by the n^2 term.
  - We say this algorithm runs "on the order of" n^2, and write it as $O(n^2)$
  - For short, we call this "***Big-Oh of n-squared***"

Asymptotic Analysis

- So, $2n^2 + 12n + 5$ is $O(n^2)$...
  - We can just get rid of a bunch of values??
- Big-Oh notation characterizes the main factors affecting growth rate



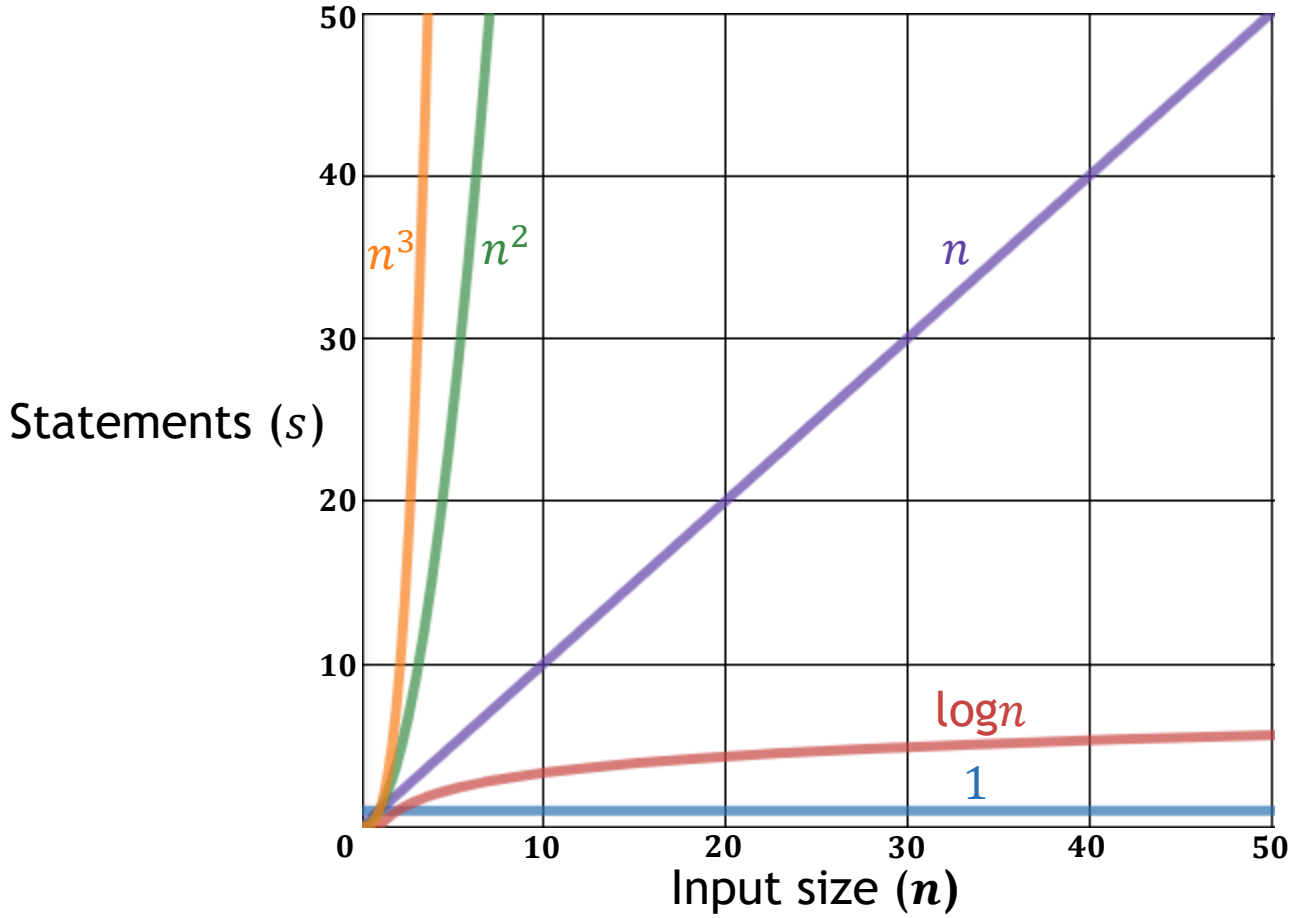




Graph images created using https://www.desmos.com/

Asymptotic Analysis

- In general, the algorithms we will see in this course will likely fall under one of the following categories:
  - Constant -- $O(1)$
  - Logarithmic -- $O(log\ n)$
  - Linear -- $O(n)$
  - Quadratic -- $O(n^2)$
  - Cubic -- $O(n^3)$



Graph images created using https://www.desmos.com/

Example: Array vs Linked List

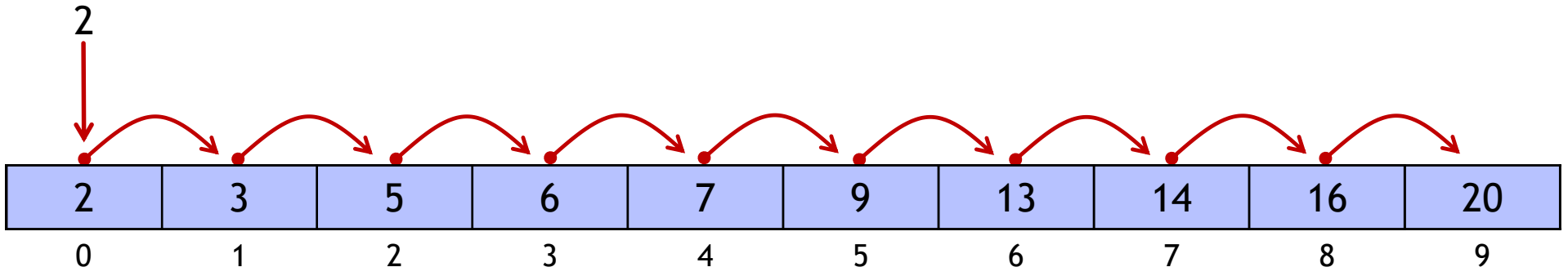
```
addFront(2);

public void addFront(int val) {
    for (int i = n; i > 0; i--) {
        data[i] = data[i-1];
    }
    data[0] = val;
    numElements++;
}
```

What if n is 10?
What if n is 10,000?
What if n is 10 billion?

The number of loop repetitions depends on n

This implementation is $O(n)$



Example: Array vs Linked List

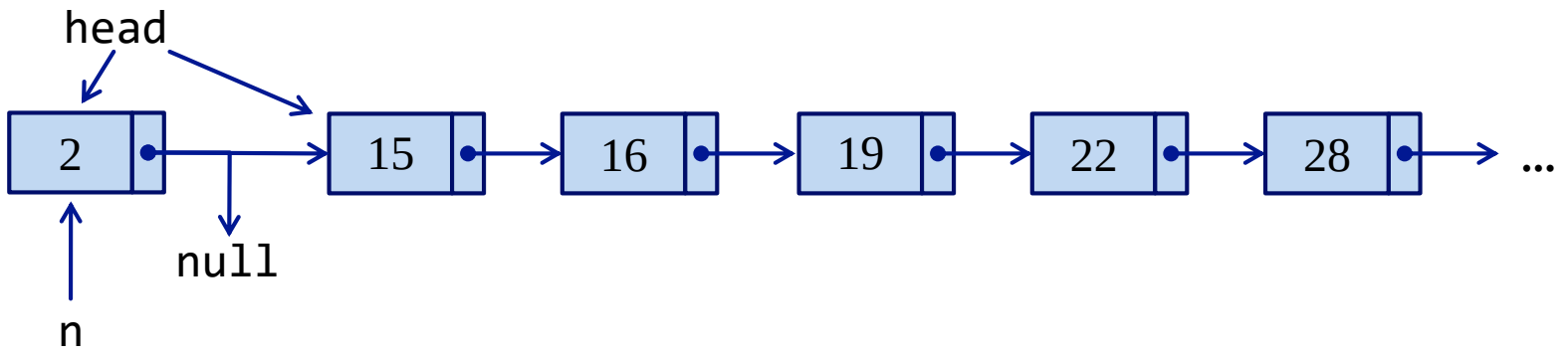
```
addFront(2);


public void addFront(int val) {
    Node n = new Node(val);
    n.next = head;
    head = n;
    numElements++;
}
```

What if n is 10?
What if n is 10,000?
What if n is 10 billion?

It doesn't matter how many elements are in the list!

This implementation is $O(1)$



Comparison

```
addFront(2);
```

- Linked list implementation: $O(1)$
- Array implementation: $O(n)$

- Linked list implementation does outperform the array implementation for all operations