CSSE 220 Day 22

Generics and Comparable
Analysis of Algorithms intro
Function Objects intro

Nothing new to check out from SVN today

Exam contents

- Exam will NOT include Chapter 14.
  - Except for the intro to analysis and big-oh which we will cover today.
  - I want to give you more time for the ideas to sink in.
  - Also I want to do a couple of other things before we get to the heart of chapter 14.
- The Computer part of the exam will not ask you to do any GUI programming.
  - There most likely will be GUI programming on the Final exam.
  - Likely things for you to do for Exam 2 Computer part:
    - Algorithms, recursion, classes, interfaces, inheritance, abstract classes, ArrayLists and Arrays.

VectorGraphics and Exam 2

- On the Written part, I may ask something about how your team did some particular aspect of the project
  - As a way of checking to make sure that everyone understands everything you did for the project
- Do you have questions about the exam ?

Questions

- Vector Graphics
- Exam
- Recursion
- Anything Else

Generic types and Collections

- Also Comparable interface

Generic Types and Collections

- Before Java 1.5 (still supported, but gives warnings):

```
ArrayList a = new ArrayList();
Integer b = new Integer(7);
a.add(b);
Integer c = (Integer)(a.get(0));
```

Explicit class cast required.

- New version (using Java generic type):

Implicit creation of Integer wrapper for 7 (auto-boxing)

<Integer> is a "type argument" to the declaration of ag.

```
ArrayList<Integer> ag = new ArrayList<Integer>();
ag.add(7);              // automatic wrapping of int.
int cg = ag.get(0); // auto unwrapping of Integer.
```

No class cast required.

automatic unboxing: Integer→int.

Efficiency: Compile time *vs* run-time checking

Comparable review:

- interface java.lang.Comparable<T>
- **Type Parameters**: T - the type of objects that this object may be compared to
- **`int compareTo(T other)`**
  - Compares this object with the specified object for ordering purposes.
  - Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

compareTo: the fine print

`int compareTo(T o)` **from the JDK API documentation**

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

The implementor must ensure `sgn(x.compareTo(y)) == -sgn(y.compareTo(x))` for all `x` and `y`. (This implies that `x.compareTo(y)` must throw an exception iff `y.compareTo(x)` throws an exception.)

The implementor must also ensure that the relation is transitive: `(x.compareTo(y)>0 && y.compareTo(z)>0)` implies `x.compareTo(z)>0`.

Finally, the implementor must ensure that `x.compareTo(y)==0` implies that `sgn(x.compareTo(z)) == sgn(y.compareTo(z))`, for all `z`.

It is strongly recommended, but *not* strictly required that `(x.compareTo(y)==0) == (x.equals(y))`. Generally speaking, any class that implements the `Comparable` interface and violates this condition should clearly indicate this fact. The recommended language is "Note: this class has a natural ordering that is inconsistent with equals."

In the foregoing description, the notation `sgn(`*expression*`)` designates the mathematical *signum* function, which is defined to return one of `-1`, `0`, or `1` according to whether the value of *expression* is negative, zero or positive.

java.lang

Interface Comparable<T>

Type Parameters:

`T` - the type of objects that this object may be compared to

- Any class that implements **`Comparable`** contracts to provide a **`compareTo()`** method

Method Detail

String is a Comparable class. If it did not already have a compareTo() method, how would you write it?

compareTo

`int compareTo(T o)`

Compares this object with the specified object for order. Returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.

- Therefore, we can write generic methods on **`Comparable`** objects. For example, in the **`java.util.Arrays`** class:

`static void`	`sort(Object[] a, int fromIndex, int toIndex)` Sorts the specified range of the specified array of objects into ascending order, according to the natural ordering of its elements.

Q3

Example of using Arrays.sort

```
import java.util.Arrays;

public class StringSort {

   public static void main(String[] args) {
      String [] toons = {"Mickey", "Minnie", "Donald",
                          "Pluto", "Goofy"};
      Arrays.sort(toons);
      for (String s:toons)
         System.out.println(s);
   }
}
```

Collections.sort can similarly be used to sort ArrayLists and other Collection objects.

Output:

```
Donald
Goofy
Mickey
Minnie
Pluto
```

Measuring program efficiency

- General hints on efficiency
  Examples
  Big-oh and its cousins

Measuring program effciency

- What kinds of things should we measure?
  - CPU time
  - memory used
  - disk transfers
  - network bandwidth
- Mostly in this course, we focus on the first two, and especially on CPU time
- To measure running time, we can call `System.currentTimeMillis()`

Program Efficiency, part 2

- **Some simple efficiency tips**
  - If a statement in a loop calculates the same value each time through, move it outside (usually before) the loop
  - Store and retain data on a "need to know" basis
    - Don't store values that you won't reuse
    - Do store values that you need to reuse
  - Don't put everything into an array when you only need one or two consecutive items at a time
  - Don't declare a variable as a field if it can be a local variable of a method

Q4

Familiar example:
Linear search of a sorted array of Comparable items

```
for (int i=0; i < a.length; i++)
  if ( a[i].compareTo(soughtItem) > 0 )
    return NOT_FOUND;  // perhaps NOT_FOUND == -1
  else if ( a[i].compareTo(soughtItem) == 0 )
    return i;
return NOT_FOUND;
```

- What should we count?
- Best case, worst case, average case?

Q5

Another algorithm analysis example

Does the following method actually create and return a copy of the string `s`?

What can we say about the running time of the method? (where **N** is the length of the string **s**)

What should we count?

```
public static String stringCopy(String s) {
  String result = "";
  for (int i=0; i<s.length(); i++)
     result += s.charAt(i);
  return result;
}
```

Don’t be too quick to make assumptions when analyzing an algorithm!

How can we do the copy more efficiently?

Q6

Break

Interlude

- Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.

  --Martin Golding

Figure 5.1

Running times for small inputs

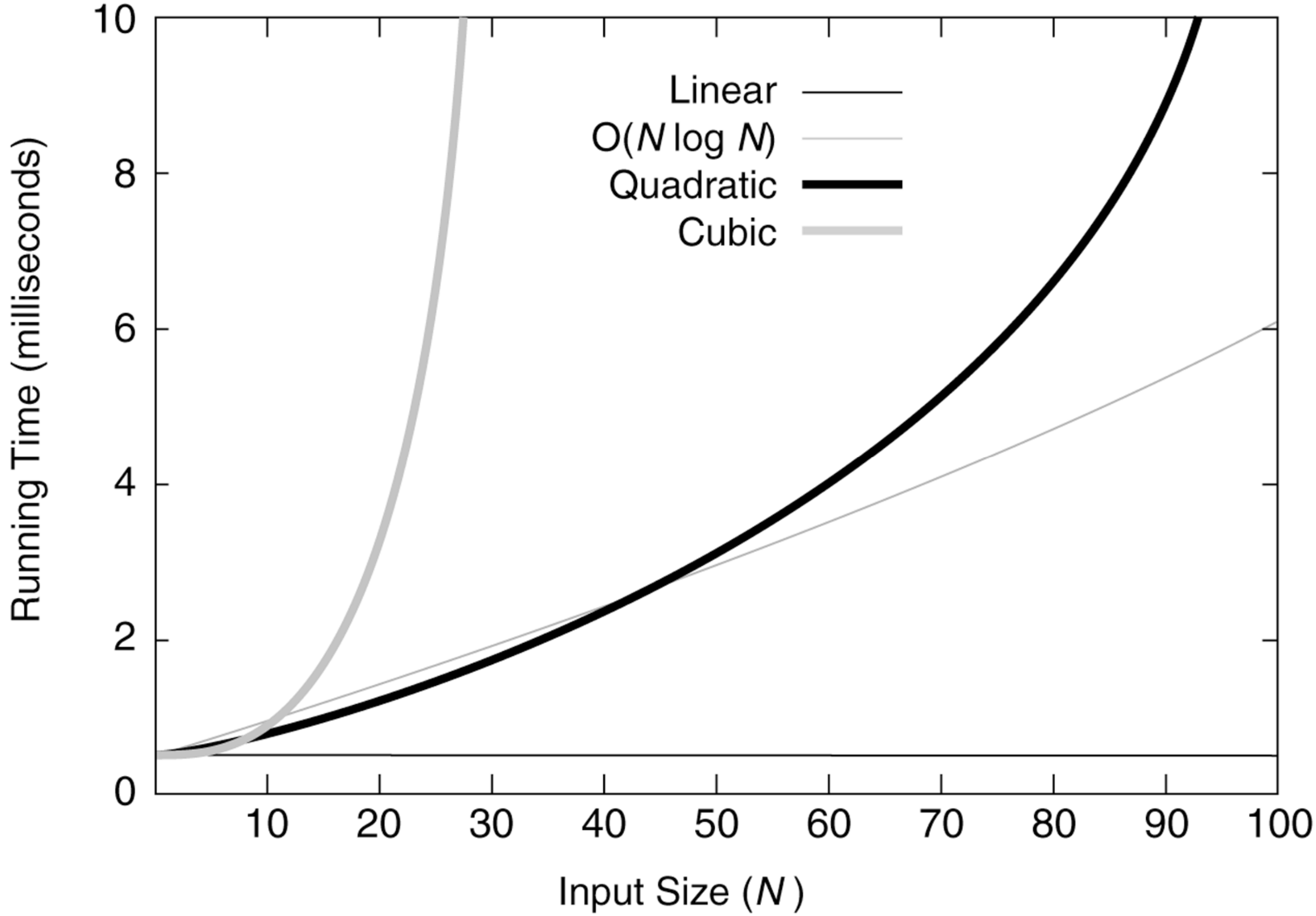


Figure 5.2

Running times for moderate inputs

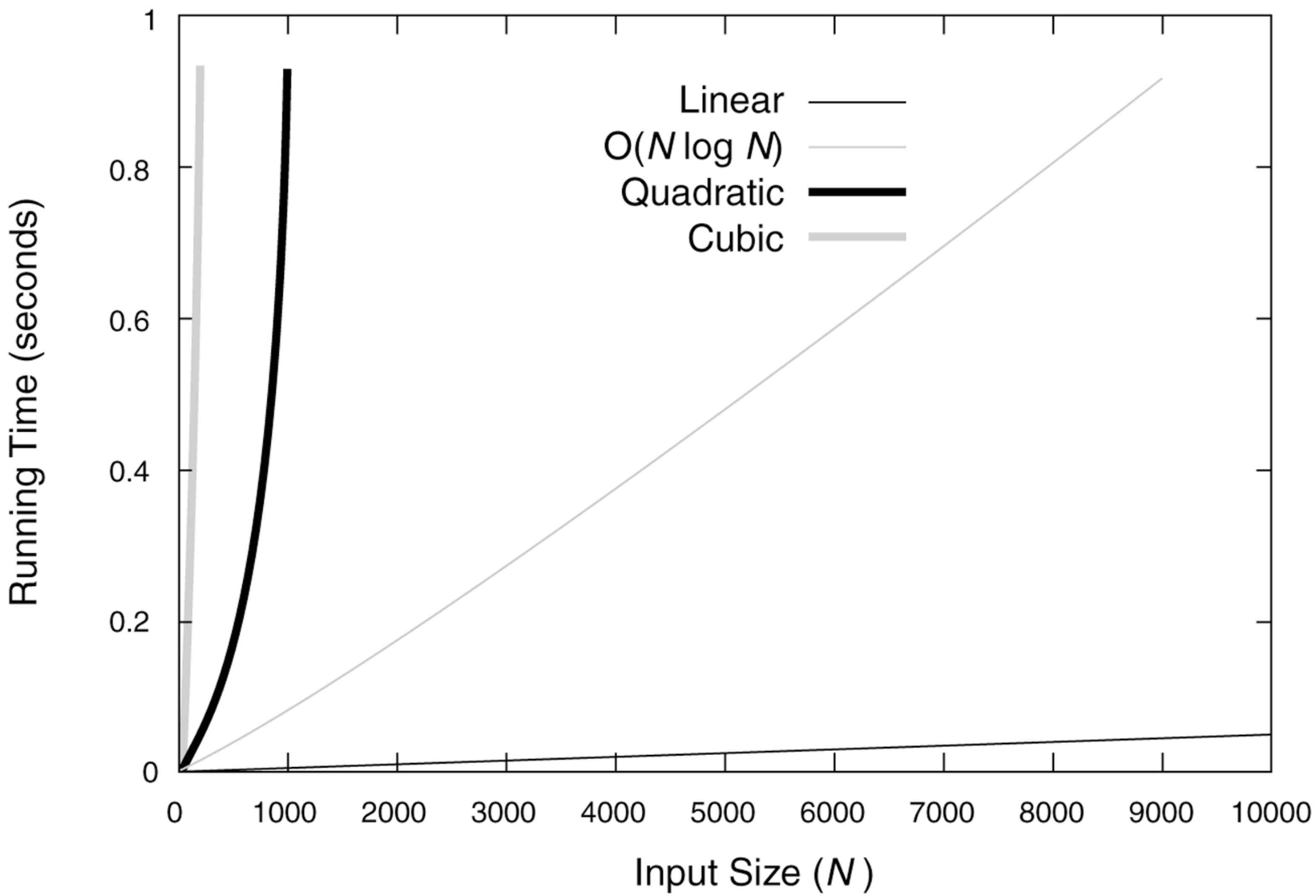


Figure 5.3

Functions in order of increasing growth rate

Function	Name
c	Constant
$\log N$	Logarithmic
$\log^2 N$	Log-squared
N	Linear
$N \log N$	N log N
N^2	Quadratic
N^3	Cubic
2^N	Exponential

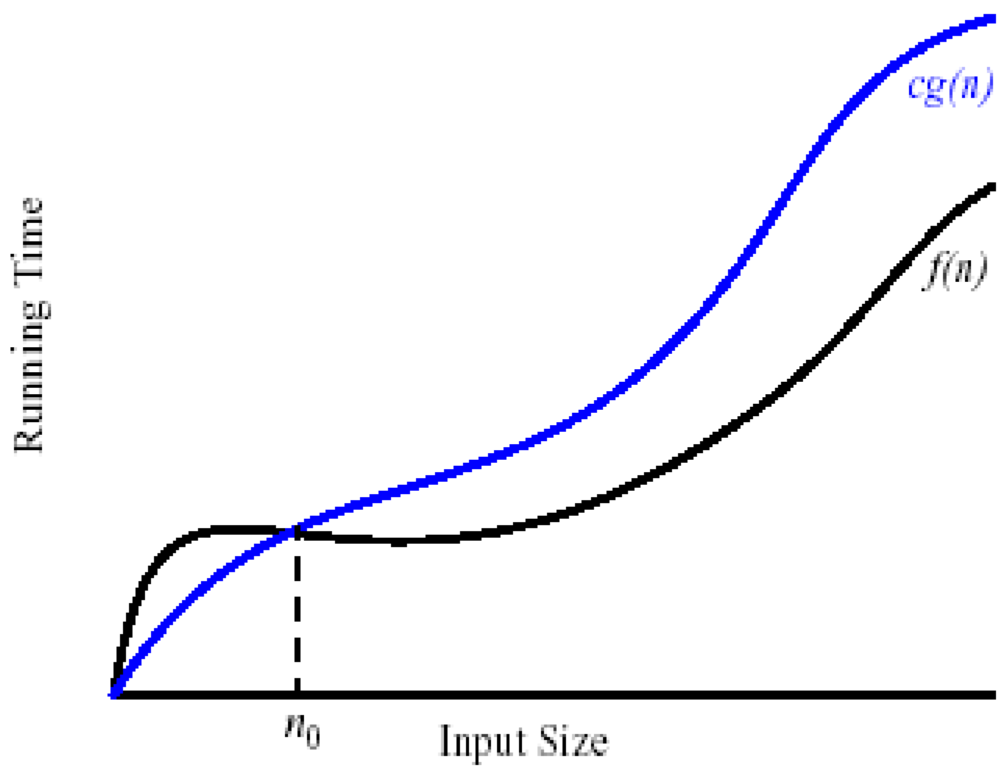
a.k.a "log linear"

Asymptotic analysis

- We only really care what happens when N (the size of a problem) gets large
- Is the function basically linear, quadratic, etc. ?
- For example, when **n** is large, the difference between $\mathbf{n}^2$ and $\mathbf{n}^2 - 3$ is negligible

- The "Big-Oh" Notation
  - given functions f(n) and g(n), we say that f(n) is O(g(n)) if and only if $f(n) \le c\, g(n)$ for $n \ge n_0$
  - c and n_0 are constants, f(n) and g(n) are functions over non-negative integers

In this course, we won't be so formal . We'll just say that f(N) is O(g(N) means that f(n) is eventually smaller than a constant times g(n).



Q7

- Simple Rule: Drop lower order terms and constant factors.
  - $7n - 3$ is $O(n)$
  - $8n^2 \log n + 5n^2 + n$ is $O(n^2 \log n)$

- Special classes of algorithms:
  - logarithmic: $O(\log n)$
  - linear $O(n)$
  - quadratic $O(n^2)$
  - polynomial $O(n^k), k \geq 1$
  - exponential $O(a^n), n > 1$

- "Relatives" of the Big-Oh
  - $\Omega(f(n))$: Big Omega
  - $\Theta(f(n))$: Big Theta

Recap: O, Ω, Θ

- f(N) is O(g(N)) if there is a constant c such that for sufficiently large N, $f(N) \leq cg(N)$
  - Informally, as N gets large the growth rate of **f** is bounded above by the growth rate of **g**
- f(N) is Ω(g(N)) if there is a constant c such that for sufficiently large N, $f(N) \geq cg(N)$
  - Informally, as N gets large the growth rate of **f** is bounded below by the growth rate of **g**
- f(N) is Θ(g(N)) if f(N) is O(g(n)) and f(N) is Ω(g(N))
  - Informally, as N gets large the growth rate of **f** is the same as the growth rate of **g**

Limits and asymptotics

- consider the limit $$\lim_{n \to \infty} \frac{f(n)}{g(n)}$$
- What does it say about asymptotics if this limit is zero, nonzero, infinite?
- We could say that knowing the limit is a sufficient but not necessary condition for recognizing big-oh relationships.
- It will be all we need for all examples in this course.

Q8

Apply this limit property to the following pairs of functions

1. N and N^2
2. $N^2 + 3N + 2$ and N^2
3. $N + \sin(N)$ and N
4. $\log N$ and N
5. $N \log N$ and N^2
6. N^a and N^n
7. a^N and b^N $(a < b)$
8. $\log_a N$ and $\log_b N$ $(a < b)$
9. $N!$ and N^N

Q9

Big-Oh Style

- **Give tightest bound you can**
  - Saying that $3N+2$ is $O(N^3)$ is true, but not as useful as saying it's $O(N)$ [What about $\Theta(N^3)$?]
- **Simplify:**
  - You *could* say:
  - $3n+2$ is $O(5n-3\log(n) + 17)$
  - and it would be technically correct…
  - It would also be poor taste … and put me in a bad mood.
- **But… if I ask "true or false: $3n+2$ is $O(n^3)$", what's the answer?**
  - True!
  - There may be "trick" questions like this on assignments and exams.
  - But they aren't really tricks, just following the big-Oh definition!

Work Time

- Begin work on your last VectorGraphics cycle. Finish before next class meeting.
  Get help as needed.