CSSE 220 Day 14

Key Concepts
Big-Oh
Function Objects Intro
Work on Minesweeper

CSSE 220 Day 14

- BallWorlds grades are on ANGEL, and scoring sheets should be in your repository. Update your project, and you should see it.
- Minesweeper Progress Report due today at the end of class.
  - Name it **Day 14 progress Report.xlsx** . Commit it to your repository.
- Extra-credit HW problem: the exam programming problems. If you do it for credit, you must do it by 11:59 PM tonight.

Answers to your questions

- Minesweeper
- Material you have read
- Anything else

Today's agenda

- More on Algorithm analysis – Big Oh
- Function objects (a.k.a Functors)
- Work on Minesweeper

Interlude (prelude)

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

--Martin Golding

Program efficiency, part 2

- **Some simple efficiency tips**
  - If a statement in a loop calculates the same value each time through, move it outside the loop
  - Store and retain data on a “need to know” basis.
  - Don’t store what you won’t reuse!
    - Do store what 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 make a variable be a field when it can be a local variable of a method.

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;
  else if ( a[i].compareTo(soughtItem) == 0 )
    return i;
return NOT_FOUND;
```

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

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?

Figure 5.1

Running times for small inputs

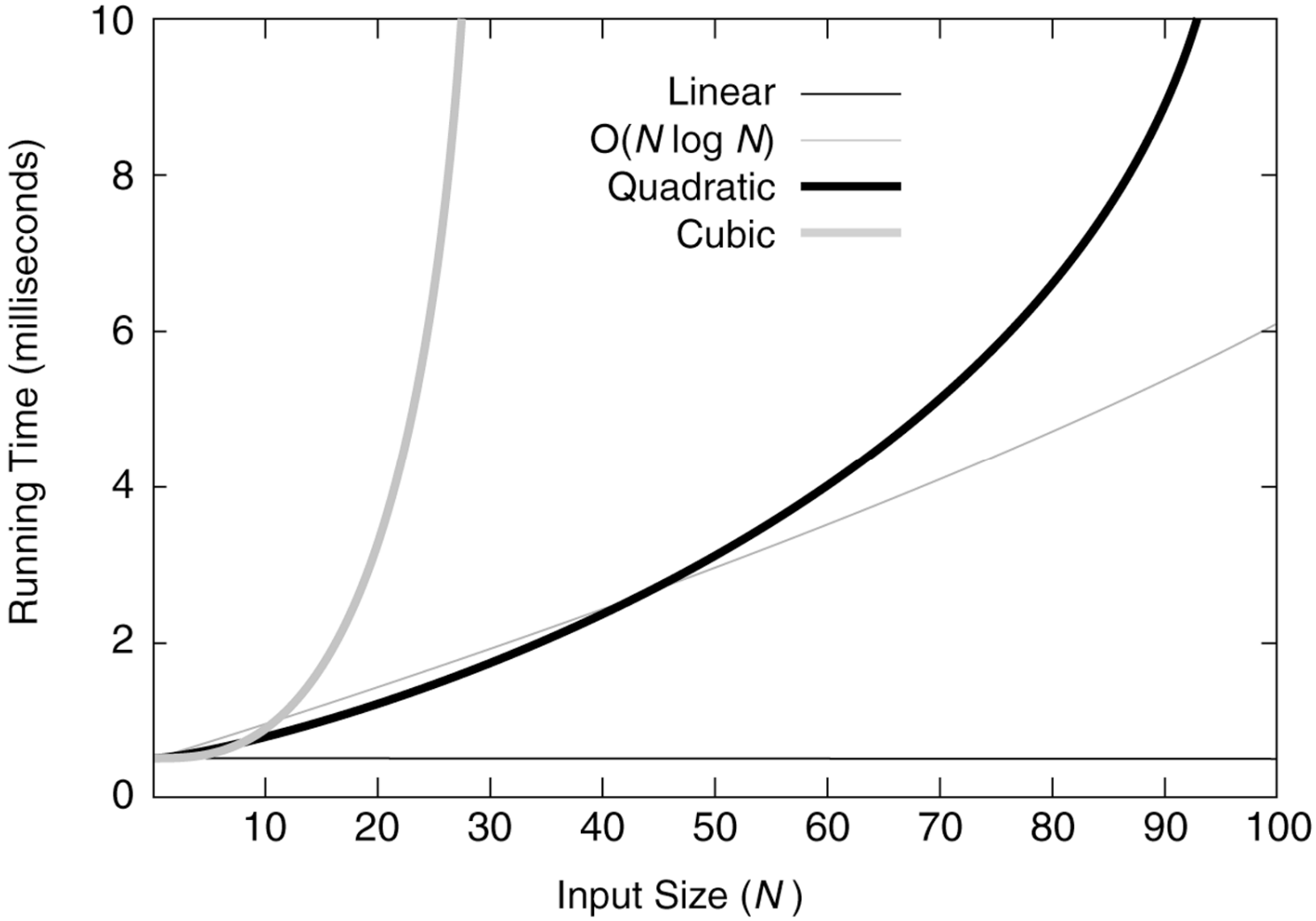


Figure 5.2

Running times for moderate inputs

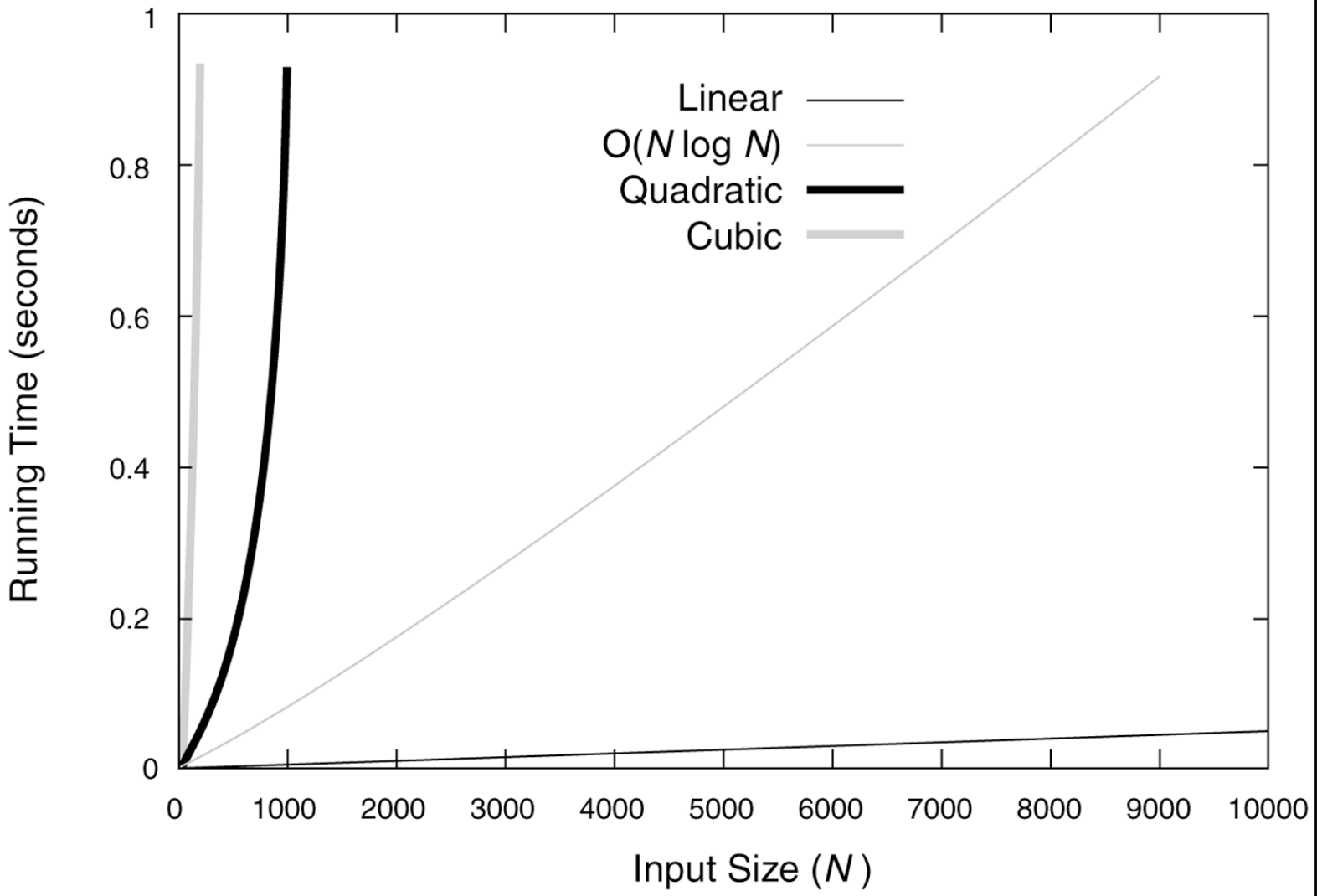


Figure 5.3

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

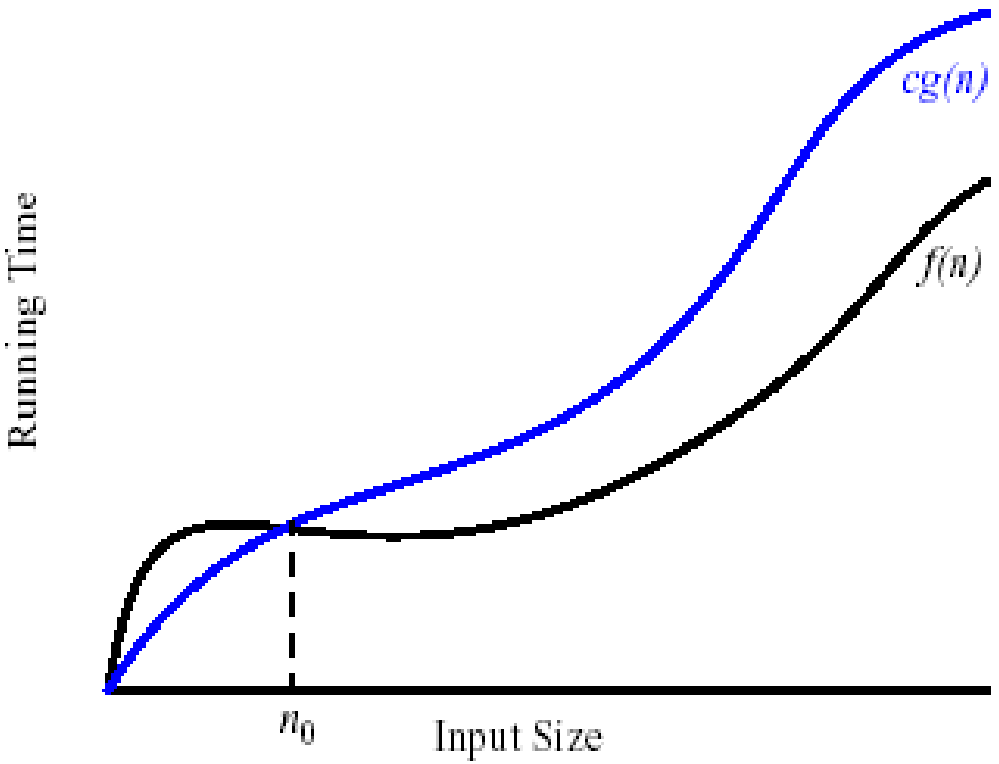
a.k.a "log linear"

Asymptotic analysis

- We only really care what happens when N (the size of a problem) gets large.
- Is the function linear? quadratic? etc.

- 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) $\leq$ c g(n) for $n \geq n_0$
  - 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).



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

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

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

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 sufficient for all examples in this course.

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

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!

Comparable review:

- interface java.lang.Comparable<T>
- **Type Parameters**: T – the type of objects that this object may be compared to
- `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.

compareTo: the fine print

```
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.

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.

Limitations of Comparable!

- How would we write `compareTo()` for a Rectangle class? What would be the basis for comparison?
- There is more than one natural way to compare Rectangles!
- What if I don't want to commit to any particular method?
- It would be nice to be able to create and pass comparison methods to other methods …

Function Objects (a.k.a. Functors)

- We'd like to be able to pass a method as an argument to another method. (what is the role of arguments to methods in general?)
  - This is not a new or unusual idea.
  - You pass other functions as arguments to Maple's *plot* and *solve* functions all of the time (on a later slide).
  - C and C++ provide *qsort*, whose first argument is a comparison function.
  - Scheme has a *sort* function, which can take a function as its first argument.

```
Chez Scheme Version 7.4
Copyright (c) 1985-2007 Cadence Research Systems
> (sort > '(7 3 9 -2 5 -6 0 4 1 -8))
(9 7 5 4 3 1 0 -2 -6 -8)
> (sort (lambda (x y) (< (abs x) (abs y)))
        '(7 3 9 -2 5 -6 0 4 1 -8))
(0 1 -2 3 4 5 -6 7 -8 9)
```

Similar example in Python

```
>>> list = [4, -2, 6, -1, 3, 5, -7]
>>> list.sort()
>>> list
[-7, -2, -1, 3, 4, 5, 6]
>>> def comp (a, b):
        return abs(a) - abs (b)

>>> list.sort(comp)
>>> list
[-1, -2, 3, 4, 5, 6, -7]
```

The comp function is passed as an argument to the sort method.

Similar example in Maple

```
> sort([3, 7, -3, 4, -6, 1, 8], `<`);
                    [-6, -3, 1, 3, 4, 7, 8]
=
> sort([3, 7, -3, 4, -6, 1, 8], `>`);
                    [8, 7, 4, 3, 1, -3, -6]
=
> absless := (x, y) -> abs(x) < abs(y);
                    absless := (x, y) -> |x| < |y|
=
> sort([3, 7, -3, 4, -6, 1, 8], `absless`)
                    [1, -3, 3, 4, -6, 7, 8]
=
>
```

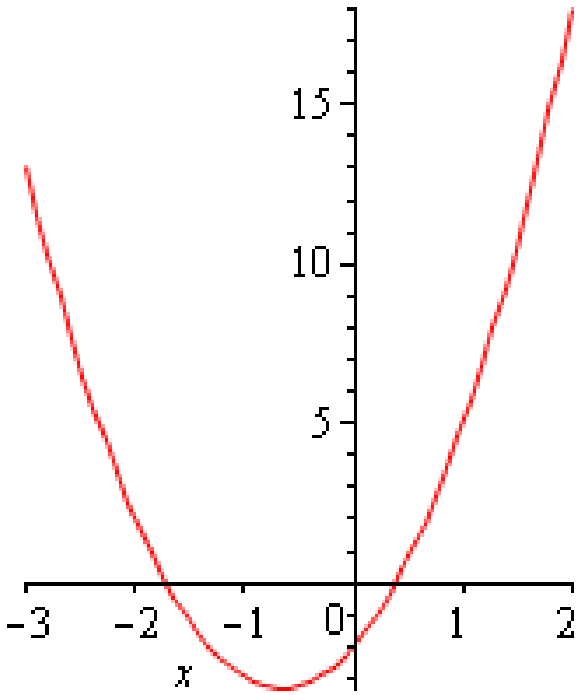
More Maple functions as parameters

```
> f := x->3*x^2 + 4*x - 2;
```

$$f := x \to 3x^2 + 4x - 2$$

=

```
> plot(f(x), x=-3..2);
```



=

```
> solve(f(x), x);
```

$$-\frac{2}{3} + \frac{\sqrt{10}}{3}, -\frac{2}{3} - \frac{\sqrt{10}}{3}$$

Java Function Objects

- What's it all about?
  - Unfortunately, Java (unlike C++) doesn't allow functions to be passed as arguments.
  - But we can create objects whose whole purpose is to pass a function into a method. They are called *function objects*, a.k.a. *functors*.
- Weiss DS book's example: **Comparator**

Work on Minesweeper

- Don't forget to commit your progress report to the repository before the end of class.