

CSSE 220 Day 13

Sorting Algorithms
Algorithm Analysis and Big-O
Searching

Checkout *SortingAndSearching* project from SVN

Questions?

What is sorting?

»» Let's see...

Why study sorting?


» Shlemiel the Painter

Shlemiel gets a job as a street painter, painting the dotted lines down the middle of the road. On the first day he takes a can of paint out to the road and finishes 300 yards of the road. "That's pretty good!" says his boss, "you're a fast worker!" and pays him a kopeck.

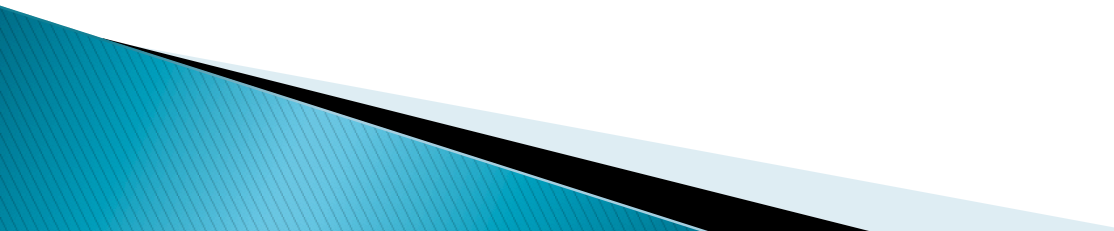
The next day Shlemiel only gets 150 yards done. "Well, that's not nearly as good as yesterday, but you're still a fast worker. 150 yards is respectable," and pays him a kopeck.

The next day Shlemiel paints 30 yards of the road. "Only 30!" shouts his boss. "That's unacceptable! On the first day you did ten times that much work! What's going on?"

"I can't help it," says Shlemiel. "Every day I get farther and farther away from the paint can!"



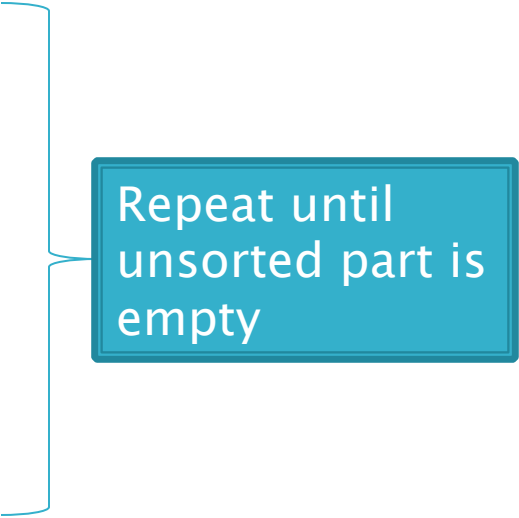
Course Goals for Sorting: You should...

- ▶ Be able to **describe** basic sorting algorithms:
 - Selection sort
 - Insertion sort
 - Merge sort
 - Quicksort
 - ▶ Know the **run-time efficiency** of each
 - ▶ Know the **best and worst case** inputs for each
- 

Selection Sort

▶ Basic idea:

- Think of the list as having a **sorted part** (at the beginning) and an **unsorted part** (the rest)
- Find the **smallest** value in the unsorted part
- Move it to the **end** of the sorted part (making the sorted part bigger and the unsorted part smaller)



Repeat until
unsorted part is
empty

Profiling Selection Sort

- ▶ **Profiling**: collecting data on the run-time behavior of an algorithm
- ▶ How long does selection sort take on:
 - 10,000 elements?
 - 20,000 elements?
 - ...
 - 80,000 elements?

Analyzing Selection Sort

- ▶ **Analyzing**: calculating the performance of an algorithm by studying how it works, typically mathematically
- ▶ Typically we want the **relative** performance as a function of input size
- ▶ Example: For an array of length n , how many times does **selectionSort()** call **compareTo()**?

Handy Fact

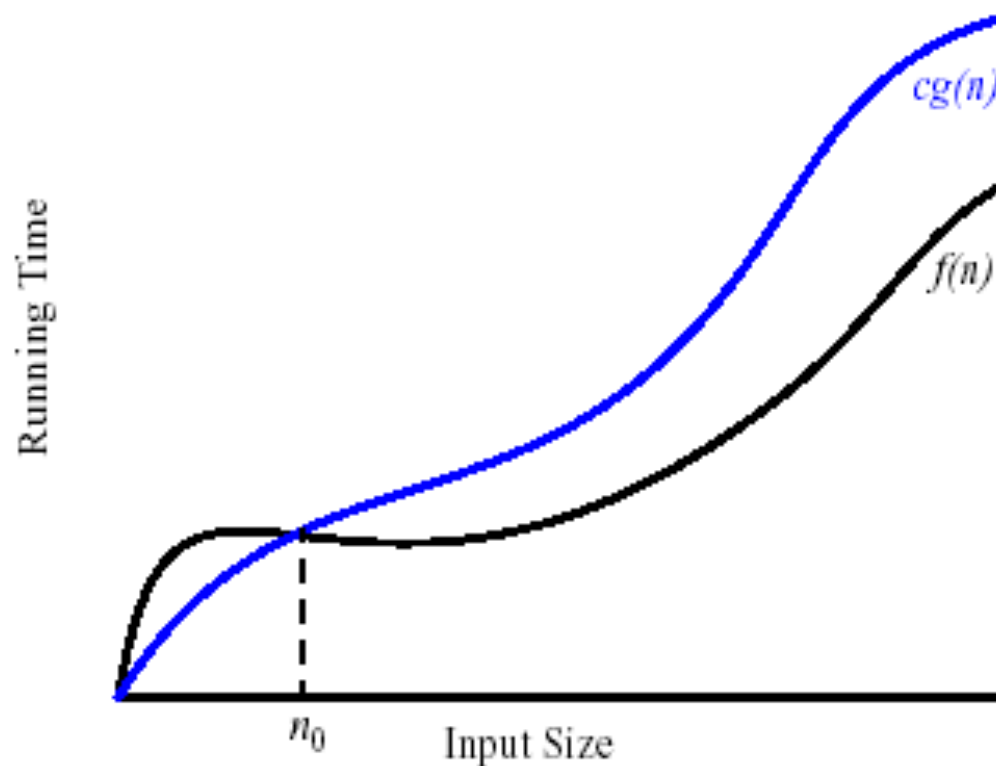
$$1 + 2 + \dots + (n - 1) + n = \frac{n(n + 1)}{2}$$

Big-Oh Notation

- ▶ In analysis of algorithms we care about differences between algorithms on very large inputs
- ▶ We say, “selection sort takes on the order of n^2 steps”
- ▶ Big-Oh gives a formal definition for “on the order of”

Formally

- ▶ We write $f(n) = O(g(n))$, and say “ f is big-Oh of g ”
- ▶ if there exists positive constants c and n_0 such that
- ▶ $0 \leq f(n) \leq c g(n)$
for all $n > n_0$
- ▶ g is a **ceiling** on f






Another Interesting Comic on Sorting ... follow link
<http://www.smbc-comics.com/?db=comics&id=1989>

Perhaps it's time for a break.

Insertion Sort

▶ Basic idea:

- Think of the list as having a **sorted part** (at the beginning) and an **unsorted part** (the rest)
- Get the **first** value in the unsorted part
- Insert it into the **correct** location in the sorted part, moving larger values up to make room



Repeat until
unsorted
part is
empty

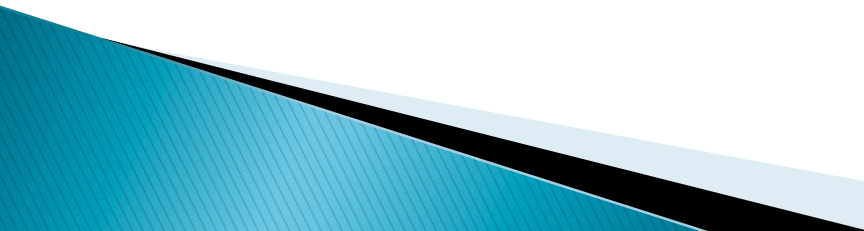
Insertion Sort Exercise, Q10–19

- ▶ **Profile** insertion sort
- ▶ **Analyze** insertion sort assuming the inner while loop runs the maximum number of times
- ▶ What input causes the worst case behavior? The best case?
- ▶ Does the input affect selection sort?

Ask for help if you're stuck!

Q9–Q18

Searching

- ▶ Consider:
 - Find Royal Mandarin Express's number in the phone book
 - Find who has the number 208-0521
 - ▶ Is one task harder than the other? Why?
 - ▶ For searching unsorted data, what's the worst case number of comparisons we would have to make?
- 

Binary Search of Sorted Data

- ▶ A **divide and conquer** strategy
- ▶ Basic idea:
 - Divide the list in half
 - Decide whether result should be in upper or lower half
 - Recursively search that half

Analyzing Binary Search

- ▶ What's the best case?
- ▶ What's the worst case?
- ▶ We use **recurrence relations** to analyze recursive algorithms:
 - Let $T(n)$ count the number of comparisons to search an array of size n
 - Examine code to find recursive formula of $T(n)$
 - Solve for n

Work Time

»» Review Homework.