

# CSSE 220

Sorting Algorithms  
Algorithm Analysis and Big-O  
Searching

Checkout *SortingAndSearching* project from SVN

Questions?

Let's see...

# **WHAT IS SORTING?**

Shlemiel the Painter

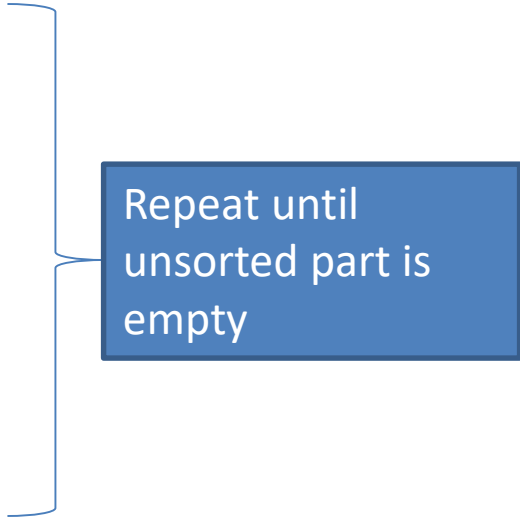
# **WHY STUDY SORTING?**

# Course Goals for Sorting: You should...

- Be able to **describe** basic sorting algorithms:
  - Selection sort
  - Insertion sort
  - Merge sort
- 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

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

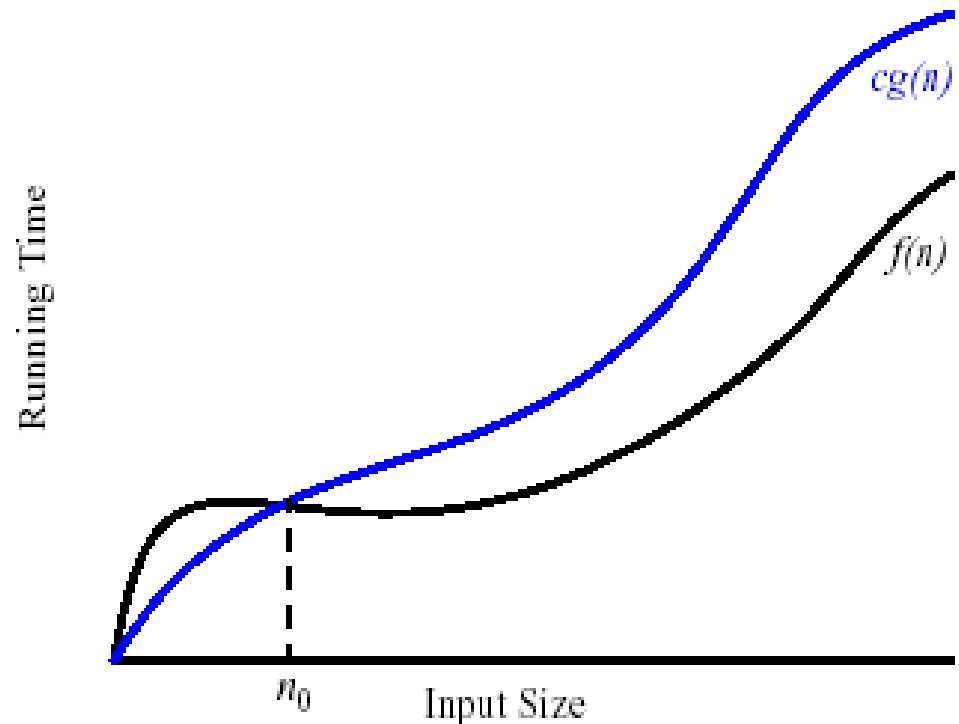


# Big-Oh Notation

- 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”

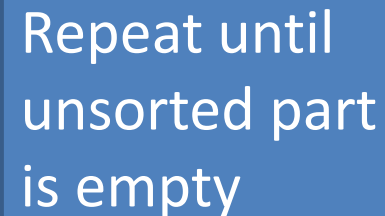
# Formally

- 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  $f$



# 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

- **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!

# Searching

- Consider:
  - Find China Express's number in the phone book
  - Find who has the number 208-2063
- 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?
- **Analyze** Binary search assuming the value searched for is at the start or end of the list

Study MergeSort for next class

**WORK TIME**