CSSE 220

Sorting Algorithms

Algorithm Analysis and Big-O

Searching

Import *SortingAndSearching* project from repo

Questions?

Let’s see…

WHAT IS SORTING?

WHY STUDY SORTING?

- At least 5 well-known algorithms that have the same functionality:
  1. Selection sort
  2. Insertion sort
  3. Merge sort
  4. Quick sort
  5. Heap sort
- Can do an analysis of each algorithm and compare the results
- Sorting is done every day all the time – think of the results of a google search

Course Goals for Sorting: You should…

- Be able to describe basic sorting algorithms:
  - Selection sort – $O(N^2)$
  - Insertion sort – $O(N^2)$
  - Merge sort – $O(N * \log_2(N))$
- Know the run-time efficiency of each
- Know the best and worst case inputs for each

Course Goals for Sorting: You should...

- Sorting Terminology:
  - Non-decreasing: use ≤
  - Non-increasing: use ≥

Input Size	Logrithmic	Linear		Quadratic
n	log2(n)	n	n*log2(n)	n^2
1	0.00	1	0	1
10	3.32	10	33	100
100	6.64	100	664	10,000
1,000	9.97	1,000	9,966	1,000,000
10,000	13.29	10,000	132,877	100,000,000
100,000	16.61	100,000	1,660,964	10,000,000,000
1,000,000	19.93	1,000,000	19,931,569	1,000,000,000,000
10,000,000	23.25	10,000,000	232,534,967	100,000,000,000,000
100,000,000	26.58	100,000,000	2,657,542,476	10,000,000,000,000,000

Selection Sort

- Basic idea:
  - Think of the ~~list~~ array as having a
  - sorted part (at the beginning) and an unsorted part (the rest)

0	1	2	3	4	5	6	7	8	9
38	44	87	2033	99	1500	100	90	239	748

  - 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

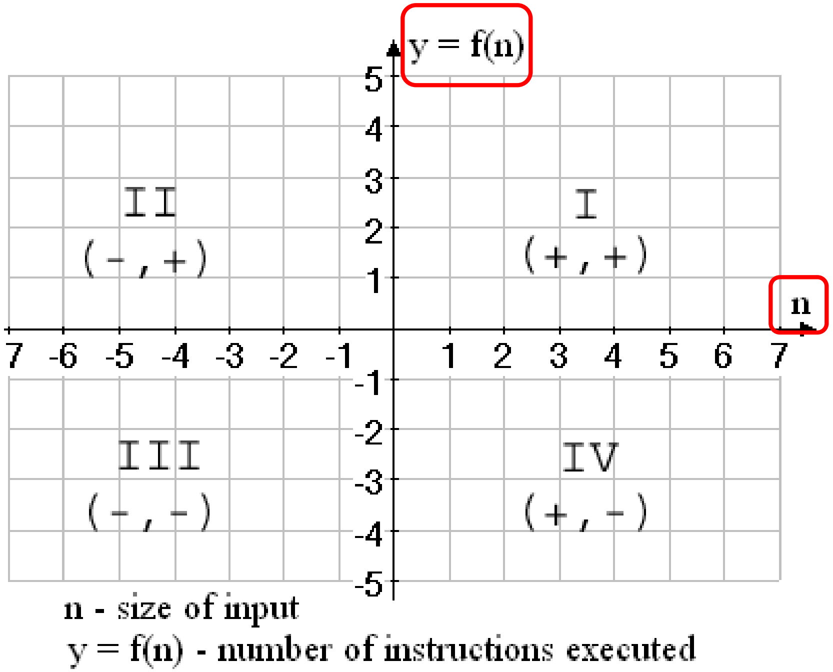
- In Eclipse, determine how long does selection sort take on:
  - 10,000 elements?
  - 20,000 elements?
  - …
  - 80,000 elements?

Q1

Performance Analysis Basics

Come up with a math function f(n) such that it does the following:

- input: n = size of the problem to be solved by the algorithm
- output: y = f(n) - the number of instructions executed
- Only care about Quadrant I



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()`**?
- Look at number of times compareTo() is called as a shortcut way to determine the Big-O

Q2-Q7

Summation Notation & Facts

$$\sum_{k=1}^{n} k = ?$$

Open form

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n$$

$$\sum_{k=1}^{n} k = \frac{n * (n+1)}{2}$$

Closed form

Induction is used to prove this

Summation Notation & Facts

$$\sum_{k=0}^{n-1} k = ?$$

Open form

$$\sum_{k=0}^{n-1} k = 0 + 1 + 2 + \cdots + n - 1$$

$$\sum_{k=0}^{n-1} k = \frac{n * (n-1)}{2}$$

Closed form

Induction is used to prove this

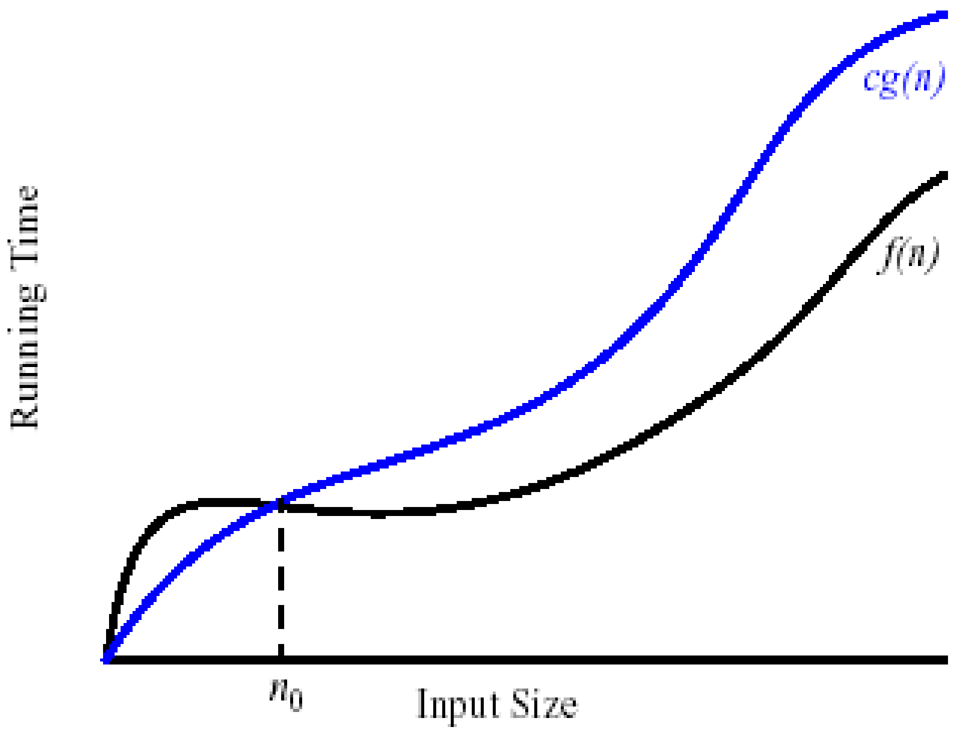
Big-Oh Notation

- In analysis of algorithms we care about differences between algorithms on very large inputs, i.e., as $n \to \infty$
- We say, “selection sort takes on the order of n^2 steps”
- Big-Oh gives a formal definition for “on the order of”

Q8

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~~ array as having

    a sorted part (at the beginning) and an unsorted part (the rest)

0	1	2	3	4	5	6	7	8	9
38	44	87	2033	99	1500	100	90	239	748

  - 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!

Q9-Q18

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?
  - Brute force approach is required

Binary Search of Sorted Data

- A divide and conquer strategy

- Basic idea:
  - Divide the ~~list~~ array in half
  - Decide whether result should be in upper or lower half
  - Recursively search that half

Analyzing Binary Search

- Analyze Binary search assuming the value searched for is at the start or end of the ~~list~~ array
- Question: How many times can you divide a number by 2, and then repeatedly divide the result by 2 until the result ≤ 1?

- What's the best case of Binary Search?

- What's the worst case Binary Search?

Q19

Study MergeSort for next class

WORK TIME

Q20-Q21