Introduction to Computer Science and Programming in C

Session 17: October 30, 2008

Columbia University

Announcements

- Homework 3 is out. Due date extended to November 11th.
- Check hw2 submission files.

Review

- Pointers and arrays behave similarly (in C)
- Memory Management
    - (<type>) malloc(N) ;
      Asks OS to give you N bytes of space, cast as <type>, returns pointer
    - free(<pointer>);
- Memory leaks

Today

- big-O notation
- Sorting algorithms

Measuring Algorithms

- In Computer Science, we want to be able to describe the running time and memory requirements of our algorithms
- A couple challenges:
    - Running time and space typically depend on input size
    - Algorithms are run on different machines

Measuring Algorithms

- For varying input sizes, we can write our time and space requirements as functions of **N**.
- For varying implementation, we need our description to not care about constant factors.

Example

- What is the running time of a function that sums an array of size 5 on a machine that takes 2 seconds to add numbers? $4 * 2 = 8$
- What if array is size **N**? $2(N\text{-}1)$
- What if it takes **c** seconds to add?

$c(N\text{-}1)$

Big-O

- $\boldsymbol{g(n) = \mathrm{O}(f(n))}$
  means that for some $\boldsymbol{c}$
  $\boldsymbol{g(n) \leq c(f(n))}$
- In other words, big-O means less than some constant scaling.
- In big-O notation, what is the running time to sum an array of size N? $c(N\text{-}1) = O(N)$

More Examples

- Space requirements for a 2-d NxN array?
- Space requirements for 10 2-d NxN arrays?
- Time required to set a char to ‘a’?

Sorting

- One of the most studied problems in CompSci
- We are given N numbers
- Put the numbers in order
  - least to greatest, greatest to least, alphabetical, etc.
  - compare two numbers at at time

Algorithm for Sorting

- In English: Given 50 index cards with numbers on them, how do you put them in order?
- Lots of different algorithms. We'll go over three

Bubble Sort

- Worst algorithm ever
- Start at beginning of deck
- Compare current and next cards. If next card should be before current, swap. Move to next card.
- Keep passing through deck until no more swaps necessary.

Bubble Sort Example

- 4 3 0 2 1
- **3 4** 0 2 1
- 3 **0 4** 2 1
- 3 0 **2 4** 1
- 3 0 2 **1 4**
- **0 3** 2 1 4
- 0 **2 3** 1 4
- 0 2 **1 3** 4

- 0 2 1 **3 4**
- **0 2** 1 3 4
- 0 **1 2** 3 4
- 0 1 **2 3** 4
- 0 1 2 **3 4**
- **0 1** 2 3 4
- 0 **1 2** 3 4
- 0 1 **2 3** 4

- 0 1 2 **3 4**
- Worst
- Algorithm
- Ever

Selection Sort

- Smarter cousin of Bubble Sort
- Find the smallest unsorted card
- Swap smallest with the first unsorted card
- Consider that card sorted, and repeat

Selection Sort Ex.

- **4** 3 0 2 1
- 4 **3** 0 2 1
- 4 3 **0** 2 1
- 4 3 0 **2** 1
- 4 3 0 2 **1**
- **0** 3 **4** 2 1
- 0 **3** 4 2 1
- 0 3 **4** 2 1
- 0 3 4 **2** 1
- 0 3 4 2 **1**
- 0 **1** 4 2 **3**
- 0 1 **4** 2 3
- 0 1 4 **2** 3
- 0 1 4 2 **3**
- 0 1 **2** **4** 3
- 0 1 2 **4** 3
- 0 1 2 4 **3**
- 0 1 2 3 4

Selection Sort Ex. 2

- 4 3 0 2 1 minimum is 0
- **0** 3 4 2 1 minimum is 1
- 0 **1** 4 2 3 minimum is 2
- 0 1 **2** 4 3 minimum is 3
- 0 1 2 3 4 minimum is 4

Merge Sort

- If deck is 2 or less cards, just sort and return
- Split deck into two halves
- Merge Sort each half-deck (recursion!)
- Then, merge the two half-decks:
  - Look at top of each deck. Take the smallest of the two. Repeat until decks are combined.

Merge Sort Example

- (4-3-0-2-1)
- (4-3) (0-2-1)
- **(3-4)** (0-2-1)
- (3-4) **((0) (2-1)**)
- (3-4) ((0) **(1-2)**)
- (3-4) (0-1-2)
- (3-4) (1-2) (0)
- (3-4) (2) (0-1)
- (3-4) (0-1-2)
- (4) (0-1-2-3)
- (0-1-2-3-4)

Running time

- Bubble Sort: O(N^2)
- Selection Sort: O(N^2)
  But the algorithm seems better organized.
- Merge Sort: O(N log(N))

Vote