Lecture 11

Sorting I

- Midterm – Preview
- Sorting
  - $O(n^2)$-time algorithms
  - $O(n \log n)$-time algorithms

Midterm – Preview

Sample Midterm

A sample midterm was posted today on the course website by Duane.

- It illustrates several question types.
- It illustrates the level of difficulty that you can expect.

Solutions will be posted and discussed on Tuesday's review session.

Sample Midterm

CS136
Williams College

This is a *closed book* exam. You have one hour and 15 minutes to complete the exam. All intended answers will fit in the space provided. You may use the back of the preceding page for additional space if necessary, but be sure to mark you answers clearly.

Be sure to give yourself enough time to answer each question— the points should help you manage your time.

In some cases, there may be a variety of implementation choices. The most credit will be given to the most elegant and efficient solutions.

Problem	Points	Description	Score
1	14	True/False	
2	10	Static	
3	26	Creating a Set class	
4	15	Recursion on Lists	
5	12	Big-O	
Total	77		

I have neither given nor received aid on this examination.

1. *(14 points)* .... True/False

1. True/false statements (2 points each). Justify each answer with a sentence or two.

a. Two instances of `class Association` in the `structure` package are equal if and only if their `keys` are equal, regardless of their `values`.

b. An instance variable declared as `protected` can be accessed by any method of the class in which it is declared.

c. A binary search can locate a value in a sorted `Vector` in $O(\log n)$ time.

Resources

Description | Lectures | Labs

Item
A sample midterm.

Update: It has now been posted.

Next Week's Schedule

An official email will be sent.
Below is a preview.

Midterm Exam

- Thursday night.
- 6pm time slot.
- 8pm time slot.

Review Session

- Tuesday at 4pm. Location TBA.

Classes and Labs

- No labs next week.
- Class only on Wednesday.

Office Hours

- Aaron on Monday 10am & 2pm in TBL 309A.
- Aaron on Tuesday 10am in TBL 309A.
- Duane on Monday night? TAs on Wednesday?

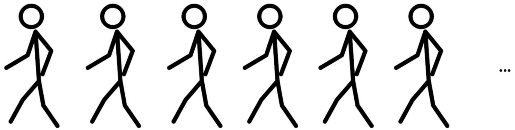
Check Google Calendar for updates

CSCI 136

Data Structures and Advanced Programming

Lectures | Labs | Resources

Mon. Sept. 27	Recursion II	Ch. 5	Duane: Lecture 8 Notes rec2.java (more recursion examples) Aaron: nanorc (.nanorc), 08-recursion.pdf (updated), Binary.java, Combo.java.



Consider the following situation.

- The first person in the line (i.e., the person on the left) is named Oscar.
- If the ith person in line is named Oscar, then the (i+1)st person is named Oscar.

What can we conclude? Why?
Consider variations of the above points, and what we can conclude.

Proof by Induction

Let S(n) be the sum of the first n odd numbers starting from 1. That is, $S(n) = 1 + 3 + \dots + 2n-1$.

Theorem: $S(n) = n^2$ for all $n \geq 1$.

Proof: We will prove that the statement is true by mathematical induction on n.

Base Case: $n = 1$. In this case, $S(1) = 1$ is the sum of the first 1 odd number starting from 1. The statement of the theorem claims S(1) equals $1^2 = 1$. This is true, so the base case is true.

Inductive Assumption: Assume that the statement is true for some k where $k \geq 1$.

Inductive Conclusion: Now we must prove that the statement is true for $n = k + 1$. The sum of the first k+1 odd numbers starting from 1 is

$$S(k+1) = 1 + 3 + \dots + 2(k+1)-1 = S(k) + 2(k+1)-1.$$

This is about the level of difficulty for an inductive proof in this course.

By the inductive assumption, $S(k) = k^2$. Therefore, we can substitute in this value and continue.

$$= k^2 + 2(k+1)-1 = k^2 + 2k + 2 - 1 = k^2 + 2k + 1 = (k+1)^2.$$

Therefore, $S(k+1) = (k+1)^2$. This proves that the statement is true for $n = k + 1$.

Therefore, by the principles of mathematical induction, the theorem is true for all $n \geq 1$.

Notes:

- The Recursion II slides were updated and posted. Includes a simpler induction proof.
- Added `.nanorc` file (but it is named `nanorc` to avoid problems) ← apologies for the delay!

Sorting

Sorting Problem

The goal of our *sorting problem* is to sort an array of n integers in non-decreasing order.

(Our solutions can easily be applied to other types of data that allow comparisons.)

1	-5	4	6	-7	4	-3

→

-7	-5	-3	1	4	4	6

Sorting a simple array.

<u>Input</u>

Array of integers $x_1, x_2, \ldots, x_n$.

<u>Output</u>

Array $y_1, y_2, \ldots, y_n$ with the same multiset of integers in *non-decreasing* order.

Activity: Advantages of Sorting?

Consider the following problems on a list of n integers:

1. *Closest pair*. Which pair of integers have the smallest absolute difference?
2. *Number of distinct elements*. What is the size of the underlying set?
3. *Mode*. Which integer occurs most frequently?
4. *Median*. Which integer (or integers) are in the middle?
5. *k^{th} largest*. For example, which integer is the 10^{th} largest?

Think about this for 2 minutes.
Then discuss it with your neighbor for 3 minutes.

In each of these cases determine the following:

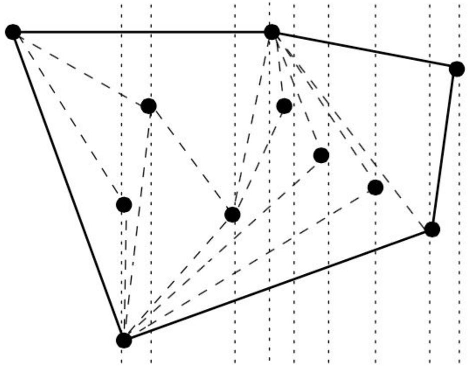
- Find an $O(n^2)$-time algorithm given an unsorted list.
- Find an $O(n)$-time algorithm given a sorted list.

When would sorting provide an overall advantage?

Applications of Sorting - Preprocessing for Algorithms

Many algorithms use sorting as an initial preprocessing step:

1. *Interval Scheduling*. The ending-first algorithm sorts intervals by ending time.
2. *Spanning Trees*. Kruskal's algorithm sorts by edges by cost.
3. *Shortest Paths*. Dijkstra's algorithm sorts by edges by cost.
4. *Huffman Encoding*. Symbols are sorted by frequency.
5. *Convex Hull*. Points are sorted by x-coordinate.



Properties of Sorting Algorithms

In our analysis we assume that the data is given in an unsorted array.

In some cases we may wish to have algorithms with additional properties:

- An algorithm is *in-place* if it does not require additional storage.

1	-5	4	6	-7	4	-3

In-place algorithms cannot create an additional array of any length either to return the sorted data or as temporary data.

- An algorithm is *stable* if equal items retain their initial relative order.

1	-5	4	2	-7	4	-3

→

-7	-5	-3	1	4	4	6

Stable algorithms maintain relative orders.

This property can be used to sort data with k keys in k steps (ie deck of cards)

Libraries and Practical Concerns

Almost every programming language has a well-known library for sorting.

In practice it is a good idea to use these libraries since they are highly optimized.

- In Python use `sorted(L)` or `L.sort()` to sort a list L.
- In Java use `Arrays.sort(A)` on any array `A` of `Comparable` objects.

Sorting Algorithms

We describe the following sorting algorithms conceptually, then consider implementations later.

1. Bubble Sort
2. Selection Sort
3. Insertion Sort
4. Merge Sort
5. Quicksort
6. Heap Sort
7. Bucket Sort

It is useful to learn about different sorting algorithms for various reasons:

- The algorithms illustrate different algorithmic principles and techniques.
- Different performance depending on the data.
  For example, Bubble Sort runs in $O(n^2)$-time, but it is very fast when the data is nearly sorted.
- Tradeoffs in terms of time/space and various properties.
  For example, Merge Sort guarantees $O(n \log(n))$-time but it is tricky to implement in-place.
- Heap Sort is a data-structure based sort.

Sorting in $O(n^2)$-time

Bubble Sort

Swap items in positions (1,2), (2,3), …, (n-1,n) if they are out of order.

- The i^{th} *pass* of this algorithm moves the i^{th} largest value into its correct position, and it can stop after examining position (n-i,n-i+1).

1	-5	4	6	-7	4	-3
-5	1	4	6	-7	4	-3
-5	1	4	6	-7	4	-3
-5	1	4	6	-7	4	-3
-5	1	4	-7	6	4	-3
-5	1	4	-7	4	6	-3
-5	1	4	-7	4	-3	6

The first pass moves the largest value into the last position.

<u>Analysis of Bubble Sort</u>

- It runs in $O(n^2)$-time in the worst-case.
  - There are n passes and each is O(n)-time.
    Actually the ith pass takes O(n-i+1)-time and the summation n + (n-1) + … + 1 is $O(n^2)$.
- If only the largest c values are out of place, then it runs in O(c·n)-time.
- It is in-place.
- It is stable if we swap based on < and not ≤.

Selection Sort

Find smallest value and move it to the first position. Repeat on the unsorted section.

- The *i^{th} pass* of this algorithm moves the i^{th} smallest into position i.
- The move can be done using a swap.

1	-5	4	6	-7	4	-3
-7	-5	4	6	1	4	-3
-7	-5	4	6	1	4	-3
-7	-5	4	6	1	4	-3
-7	-5	4	6	1	4	-3
-7	-5	-3	6	1	4	4
-7	-5	-3	6	1	4	4

The first three-and-a-half passes of the algorithm.

<u>Analysis of Selection Sort</u>

- It runs in $O(n^2)$-time in the worst-case.
  - There are n passes and each is O(n)-time.
    Actually the ith pass takes O(n-i+1)-time and the summation n + (n-1) + … + 1 is $O(n^2)$.
- It is in-place.
- It is stable if we choose the first instance of the smallest value.

Insertion Sort

Assume the first i values are sorted, and insert the i+1st value in the correct position.

Repeat for i = 1, 2, ... , n-1.

- The i^{th} *pass* of this algorithm moves the ith smallest into position i.

1	-5	4	6	-7	4	-3
-5	1	4	6	-7	4	-3
-5	1	4	6	-7	4	-3
-5	1	4	6	-7	4	-3
-7	-5	1	4	6	4	-3
-7	-5	1	4	4	6	-3
-7	-5	-3	1	4	4	6

The complete insertion sorting using 6 passes.

<u>Analysis of Insertion Sort</u>

- It runs in $O(n^2)$-time in the worst-case.
  - There are n-1 passes and each is O(n)-time.
    Actually the ith pass takes O(n-i+1)-time and the summation n + (n-1) + … + 1 is $O(n^2)$.
- It is in-place.
- It is stable if we insert to the right of any repeated value.

Where will the log(n) come from?

Sorting in O(n log n)-time