CSC207.01 2014S, Class 19: Linear and Binary Search

Overview

- Preliminaries.
  - Admin.
  - Upcoming work.
  - Questions on the exam.
- Analysis of binary search.
- Functions as parameters.
- Generics (e.g., Comparator)
- Lab.

Preliminaries

Admin

- Start Eclipse!
- Today’s self-gov problems
  - How do we get students to wash their hands?
  - What should first-year students do during spring break?
    - Couch surf at friends and family
- I looked at some C code with Walker today. I apologize for the extreme differences in our approaches.
  - Do you need more explanation of my perspective?
- Tomorrow’s review session is up in the air; the storm may change my plans. I’ll send email tonight.
- Reminder: Summer research applications are (mostly) due on Friday.
- Extra credit:
  - CS Extras: Technical Interviews
  - CS Table: Skip Lists
  - More?

Upcoming Work

- Finish the exam.
  - Email me questions!
- Today’s writeup: Exercise 3
  - Subject: CSC 207 Writeup 10: Searching (OPTIONAL NAME)
- Reading for friday: Loop Invariants (forthcoming)
Questions on the exam

Analyzing binary search

- Analyzing iterative and recursive algorithms

- Iterative:
  - Count the number of times each loop runs.
  - Count the number of steps in the loop.
  - Multiply.

(defun insertion-sort (lambda (lst) (let (loop ([remaining lst] [sorted null]) (if (null? remaining) sorted (loop (cdr remaining) (insert (car remaining) (sorted)))))))

- Analysis
  - n repetitions
  - each involves 1 test, 1 cdr, 1 call to insert
  - A call to insert is in O(n)
  - n*(3 + n) = n^2 + 3n is in O(n^2)

- Recursive functions:
  - Write a recurrence relation for running time
  - Make that recursive definition non-recursive (closed form)
  - Typical informal mechanisms:
    - Work out values, starting at the bottom
    - Continually expand, look for a pattern

\[ t(n) = c + t(n/2) \]
\[ t(1) = d \]
\[ t(2) = c + t(2/2) = c + t(1) = c + d \]
\[ t(4) = c + t(4/2) = c + t(2) = c + c + d = 2c + d \]
\[ t(8) = c + t(8/2) = c + t(4) = c + 2c + d = 3c + d \]
\[ t(16) = c + t(8) = c + 3c + d = 4c + d \]

pattern: \[ t(2^k) = k \cdot c + d \]

\[ t(n) = c + t(n/2) = c + c + t(n/4) = 2c + t(n/4) = 2c + c + t(n/8) = 3c + t(n/8) = 3c + c + t(n/16) = 4c + t(n/16) \]

pattern: \[ t(n) = kc + t(n/2^k) \] When \( n = 2^k \), this is \( t(n) = kc + t(1) = kc + d \) When \( n = 2^k \), \( k = log_2(n) \)

So \( t(n) = log_2(n)c + d \) is in \( O(log_2(n)) \)

Functions as parameters

When you write a searching or sorting algorithm, you often want a function as a parameter
(define search
  (lambda (lst ok?)
    (if (null? lst)
        #f
        (if (ok? (car lst))
            (car lst)
            (search (cdr lst) ok?)))))

Java does not (currently) allow functions as first class values. But it does allow objects/interfaces as first-class values. Instead of passing in a function, we pass in an object that contains that function/method.

```java
public interface Predicate {
    public boolean ok(Object o);
} // interface Predicate

public class LessThanTwo implements Predicate {
    public boolean ok(Object o) {
        return (o instanceof Number) && (((Number) o).doubleValue < 2.0);
    } // ok
} // class LessThanTwo
```

Great idea in Scheme: Anonymous functions

```scheme
(search students (lambda (student) (and (here? student) (awake? student))))
```

Java has anonymous classes! We’ll look at them later.

What’s the difference between Java’s Comparator and Comparable?

- Comparable: Something that has a natural ordering: Thing 1, compare yourself to Thing 2
- Comparator: Something that knows how to compare, using a desired ordering. Each comparator takes two things and says which "comes first" using some criterion.

**Generics (e.g., Comparator<T>)**

See forthcoming reading.

**Lab**

What’s wrong with the following?
if (vals[mid] > val)
    vals = Arrays.copyOfRange(vals, 0, mid);
else if (vals[mid] < val)
    vals = Arrays.copyOfRange(vals, mid+1, vals.length);
int mid = lower + upper/2;

int mid = (lower + upper)/2;

I’m proud of you.

Copyright (c) 2013-14 Samuel A. Rebelsky.

This work is licensed under a Creative Commons Attribution 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by/3.0/ or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.