

Discrete Mathematical Structures

CS 3233 Lecture 21

Prof. William Winsborough

October 21, 2005

Business

- Volunteers needed to administer IDEA survey
- Assignment 7
 - Due Thursday 10/27 2pm Dr. Winsborough's office
 - Section 2.3: 6, 8, 24
 - In 24, do the following:
 - First, do an analysis of the type suggested in Slide 3 of today's lecture, i.e., one in which the number of passes and the cost of each pass are both estimated by using Big-O to derive a Big-O bound on the total cost
 - Second, do a precise analysis of the worst-case in which the Big-O is not used, but rather the precise total number of comparisons is calculated. Do not count comparisons that are performed for loop bookkeeping
 - Third, determine whether the bound $g(n)$ you derived in the first part is accurate by checking whether the precise cost you calculated in the second part is $O(g(n))$
 - Fourth, calculate the expected-case cost under the assumption that each possible input sequence of n numbers is equally probable. (Hint: How is the number of comparisons required affected by the input sequence?)

Why is Big-O Handy?

- You don't have to worry about a lot of details

```
proc bubble sort(a1,a2,...,an)
```

```
  for i := 1 to n-1
```

```
    for j := 1 to n-i
```

```
      if aj > aj+1 then interchange aj and aj+1
```

- In pass i , $n-i$ comparisons are performed

- Without using Big-O, we have cost of

$$\sum_{1 \leq i \leq n} (n-i) = \sum_{0 \leq k \leq n-1} k = (n-1)n/2$$

- Labor-intensive because of manipulating summations

- Using Big-O, constant terms and factors are not significant

- In each of $O(n)$ passes, $O(n)$ comparisons are performed

- Total cost: $O(n^2)$

Problems versus Algorithms

- The complexity of a problem is characterized by the complexity of the best algorithm for solving it
- It is always possible to give a sub-optimal algorithm
 - In Chapter 3, we will see that the sorting problem is actually $n \log n$, even though the algorithms we have discussed so far are quadratic (n^2)

Theory versus Practice

- Some algorithms that are intractable in the worst case usually execute much faster than algorithms that are always tractable
- If the intractable cases are rare enough, such algorithms may be preferable
- Thus, while Big-O analysis is helpful, it cannot be considered 100% conclusive in all cases about the relative merits of different algorithms

Undecidable Problems

- There are problems that cannot be solved by any algorithm
 - Such problems are called *undecidable*
 - The book calls them *unsolvable*
- Alan Turing first proved that the *halting problem* is undecidable
 - Given an algorithm and an input, determine whether the algorithm halts when executed on the input

Problem Classes: P versus NP

- Problems in P (polynomial time) have polynomial-time algorithms
- Problems in NP (nondeterministic polynomial time) have no **known** polynomial-time algorithms, but for which a solution can be checked in polynomial time
 - While it remains open whether $P = NP$, most computer scientists consider this unlikely

NP-Complete Problems

- There is a sub-class of NP called NP-complete for which, if any such problem can be solved in polynomial time, they all can
 - Satisfiability of propositional formulas is an NP-complete problem
 - Given a truth assignment that makes the formula true, we can verify this in polynomial time
 - There is no known algorithm to find such assignments in polynomial time
 - If one were discovered, it would prove $P = NP$