

Discrete Mathematical Structures

CS 3233 Lecture 20

Prof. William Winsborough

October 19, 2005

Business

- Grading policy
 - Final grades will be based on exam and homework **scores**
 - Letter grades presented for first midterm are intended only as a general idea of how students are doing
- Recall second midterm is November 11
- I will be out of town 11/8-11

Average-case Complexity

- While worst-case complexity analyzes the case in which the input causes the algorithm's cost to be maximal,
 - Average-case Complexity considers all input cases.
 - Also called Expected-case Complexity or, less formally, Expected Cost
- The expected cost of executing an algorithm is the sum over the different cases of the cost of each case times its probability
 - Can be difficult to calculate
 - Assumes you know the probability of each case

Linear Search Expected Cost

```
proc linear search(x:int; a1,a2,...,an: distinct ints)
  i := 1
  while (i ≤ n and x ≠ ai)
    i := i + 1
  if i ≤ n then location := i else location := 0
```

- Each case has x appearing in a different location
- When x is in position i, $1 \leq i \leq n$, $2i + 1$ comparisons are performed
- If we assume x appears somewhere in the list and that each position is equally probable,
 - The probability of each case is $1/n$
 - The expected cost is $\sum_{1 \leq i \leq n} (2i+1)(1/n)$

Simplification of Sum

$$\begin{aligned}\sum_{1 \leq i \leq n} (2i+1)(1/n) &= (1/n) \sum_{1 \leq i \leq n} (2i+1) \\ &= (1/n) (n + \sum_{1 \leq i \leq n} 2i) \\ &= (1/n) (n + 2 \sum_{1 \leq i \leq n} i) \\ &= (1/n) (n + 2((n(n+1))/2)) \\ &= (1/n) (n + n^2 + n) \\ &= 2 + n\end{aligned}$$

Which is $\Theta(n)$

Worst-case Analysis of Bubble Sort

```
proc bubble sort( $a_1, a_2, \dots, a_n$ )
```

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

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

```
    if  $a_j > a_{j+1}$  then interchange  $a_j$  and  $a_{j+1}$ 
```

- In pass i , $n-i$ comparisons are performed
 - (not counting the test to see if the inner loop is finished)
 - So cost is $\sum_{1 \leq i \leq n} (n-i) = \sum_{0 \leq k \leq n-1} k = (n-1)n/2$

Complexity	Terminology	Example
$O(1)$	Constant complexity	Find max value in first 100 elements
$O(\log n)$	Logarithmic complexity	Binary search
$O(n)$	Linear	Linear search
$O(n \log n)$	$n \log n$	Merge sort (ch 3)
$O(n^b)$	Polynomial	Bubble sort
$O(b^n), b > 1$	Exponential	Evaluate logical formulas via truth tables
$O(n!)$	Factorial	Traveling salesman



* Considered *tractable*

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

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
- 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 $P = NP$