Principles of Algorithm Analysis (Ch2)

Philip Chan

Why analyze algorithms? What do we want to analyze?

1 Empirical Analysis

• What is empirical analysis?
• actual data, random data, or perverse data
• faster algorithms vs. added complexity
• in what situations when performance analysis is not necessary?

2 Mathematical Analysis

• What is mathematical analysis?
• identify abstract operations to count (unit for counting)
• average-case, worst-case, best-case scenarios
• characterize by mathematical functions
• What does empirical compare to mathematical analysis?

3 Growth of Functions

• primary parameter $N$ (usually input size) of the mathematical function
• constant: 1
• logarithmic: $\log N$ ($\log_2 = \lg$, $\log_e = \ln$)
• linear: $N$
• $N \log N$
• quadratic: $N^2$
• cubic: $N^3$
• polynomial: quadratic and up
• exponential: $2^N, N!$
• Table 2.1, p39
• $2^{100} = 1.3 \times 10^{30}; 100! = 1.0 \times 10^{658}$
• How fast is your computer?
• 1 year = $3.1 \times 10^7$ seconds
4 Big-Oh (O) Notation

- A function $g(N)$ is said to be $O(f(N))$ if there exist constants $c_o$ and $N_o$ such that:
  
  $g(N) < c_o f(N)$ for all $N > N_o$. 

- Saying in words: “$g(N)$ is of order $f(N)$”

- $g(N) = N + 10$; $g(N)$ is $O(N)$?
- $N + 10$ is of order $N$
- $g(N) = 2N + 10$; $g(N)$ is $O(N)$?
- $g(N) = N^2 + 10$; $g(N)$ is $O(N^2)$?
- Big-Oh is an approximation
- Manipulation: Q2.21, p48.
- most interested in leading (largest) terms
- $O(f(x) + g(x)) \rightarrow O(f(x))$ if $O(f(x)) > O(g(x))$
- example: p45

- $(N + O(1))(N + O(\log N) + O(1))$
  $N^2 + O(N) + O(N \log N) + O(\log N) + O(N) + O(1)$
  Dropping all but the largest $O$-term:
    $N^2 + O(N \log N)$

- Asymptotic expression: a formula with one $O$-term

5 Examples of Algorithm Analysis

- Sequential (Linear) Search, Program 2.1, p54
- best, worst, average cases
- Binary Search, Program 2.2, p56 (what is the prerequisite?)
- best, worst cases
- worst case: easier to use a recurrence relation
- Empirical analysis: Table 2.4, p59 ($M$ is number of searches)

6 Basic Recurrences

Set up a recurrence relation; solve it (substituting or telescoping)

1. Binary search
   $C_N = C_{N/2} + 1$, for $N \geq 2$ with $C_1 = 1$
   $C_N = O(\log N)$

2. $C_N = C_{N-1} + N$, for $N \geq 2$ with $C_1 = 1$
   $C_N = (N(N + 1))/2 = O(N^2)$
   (Bubble sort, Program 6.4, p278)

3. $C_N = 2C_{N/2} + N$, for $N \geq 2$ with $C_1 = 0$
   $C_N = O(N \log N)$
   (Merge sort, Program 8.3, p354)
4. $C_N = C_{N/2} + N$, for $N \geq 2$ with $C_1 = 0$

$C_N = O(N)$

7 More Examples of Algorithm Analysis

- Divide-and-conquer to find the max, Prog 5.6, p210
- $T_N = 2T_{N/2} + 1$, for $N \geq 2$ with $T_1 = 1$
  $T_N = O(N)$
- Tower of Hanoi, p213
- $T_N = 2T_{N-1} + 1$, for $N \geq 2$ with $T_1 = 1$
  $T_N = 2^N - 1 = O(2^N)$ [derive it in HW4]
- Property 5.1 (p. 211): recursively divide into two non-overlapping parts; $N - 1$ calls
  Recurrence relation: $T_N = T_k + T_{N-k} + 1$, for $N \geq 1$ with $T_1 = 0$
  Solution: $T_N = N - 1 = O(N)$
  Proof by Induction:
  1. check $N = 1$: $T_1 = 1 - 1 = 0$
  2. assume $T_i = i - 1$ for any $i \geq 1$, show $T_{i+1} = i$
    $T_{i+1} = T_k + T_{i+1-k} + 1 = (k - 1) + ((i + 1 - k) + 1 = i$

8 Worst and Average Case Analyses

- Worst-case
  - make “guarantees” about when the algorithm will always finish
  - the worst case might be very different from a normal case in practice
  - good worst case algorithms are more complicated
- Average-case
  - make “predictions” about when the algorithm will usually finish
  - input model: every possible input is equally likely might not be realistic (random input model)
  - might require deep mathematical reasoning
  - might need to know standard deviation to characterize the distribution of running times

9 Computational Complexity

- study the “limitations” of algorithms (how difficult a given problem is)
- The complexity of a problem is the worst-case running time (in big-oh) of the “best” algorithm, which might or might not be known
- Upper Bound: The “best known” algorithm of a problem sets an upper bound of the problem’s complexity (difficulty can’t be higher)
- Lower Bound: A theoretical lower bound of the complexity of a problem can be found using intricate mathematical reasoning; it represents the intrinsic difficulty of the problem (difficulty can’t be lower)
- What if the upper bound and the lower bound are the same?
- binary search is “optimal”—no algorithms can perform fewer comparisons to find an item in a sorted array.