Mathematical Insights in Computing, Day 13

Asymptotic Notation and Analyzing Algorithms

Problem 1. Binary Search

Consider the following pseudocode for a recursive algorithm.

BinarySearch(*list*, ***x***):

Our input is a sorted list, from smallest to greatest. For example: 1 2 4 6 8 9 10 12 18 19 24.
*We want to figure out where a particular number **x** falls in the list. For example: **x** = 18.*

i. *Let **n** be the length of the list.*
ii. *Look at the middle number (the $(n/2)^{th}$ number) in the list. For example: 9.*
 iii. *Is M equal to x? If so, return (n/2).*
 iv. *Is M greater than x? If so, return **BinarySearch**(the bottom half of the list, x).*
 v. *Is M less than x? If so, return **BinarySearch**(the top half of the list, x).*

a). Consider step *iv*. How big is "the bottom half of the list"? (Express your answer in terms of n.)

b). Steps *i*, *ii*, and *iii* take a constant number of operations, each. Call this O(1).

Let *T(n)* be the number of operations (the time) that this algorithm takes in order to find something in a list that is *n* elements long.

Write a recurrence describing this algorithm: Express *T(n)* in terms of *n*, O(1), and *T*.

c). Solve the recurrence using guess-and-check. How many operations does the algorithm need to do, to find some item within a list of n elements?

(Hint: For easier guessing-and-checking, try out some examples. How long does it take for a list of 2 things? A list of 4 things? 8? 16?)

Express your answer using big-O notation. It should only depend on n.

Problem 2. Sorting Functions

Rank the following functions by order of growth. There might be ties – two functions f and g may be tied if and only if $f(n) = \Theta(g(n))$. Put any functions which tie in the same group, then organize all the groups in order from slowest-growing to fastest-growing.

2^{n^2}	2^n	3^n
$n \log_2 n$	$n^{\log n}$	n^n
$(n+1)!$	$n!$	$n(n+1)/2$
$\log_{10} n$	$\log_2 n$	$(\log_2 n)(\log_2 \log_2 n)$
n^2	$n^{2.5}$	n^3
n	$\sqrt[3]{n}$	$\sqrt{n}$
2^{100}	$\mathrm{BB}(n)$	$\mathrm{Ackermann}(n)$

Problem 3. Elementary-School Algorithms

a) **Addition**. Consider the problem of adding two base-10 numbers.
Describe, in English or pseudocode, the algorithm you were taught for doing this.

Analyze this algorithm. For two n-bit numbers, how long does it take? Use big-O notation.

b) **Subtraction.** Consider the problem of subtracting two base-10 numbers.
Describe, in English or pseudocode, the algorithm you were taught for doing this.

Analyze this algorithm. For two n-bit numbers, how long does it take? Use big-O notation.

c) **Multiplication**. Consider the problem of multiplying two base-10 numbers.
Describe, in English or pseudocode, the algorithm you were taught for doing this.

Analyze this algorithm. For two n-bit numbers, how long does it take? Use big-O notation.

d) **Division.** Consider the problem of dividing two base-10 numbers.
Describe, in English or pseudocode, the algorithm you were taught for doing this.

Analyze this algorithm. For two n-bit numbers, how long does it take? Use big-O notation.

Problem 4. An Easy Hard Problem

Archaeologists recently unearthed a mysterious box off the coast of the Isle of Boolean Circuits. Although the box cannot be opened, it is covered in abstruse computer science symbols. The archaeologists have brought it to you for your expert advice.

The back side of the box is covered in exactly one hundred tiny switches, which can either be set to ON or OFF. On the front side of the box is a smiley face. Deciphering the ancient symbols, you discover that legend has it that only when the correct combination of switches is flipped, the smiley face will glow a happy green color, and all the worlds' problems (including, of course, world hunger, world peace, the P vs. NP question, and the Halting Problem) will finally be solved.

a) You want to find the right combination of switches, and there's no time to lose! What algorithm will you use in order to find it? Describe in pseudocode or English.

b) Analyze your algorithm. Maybe you'll get lucky and find the right combination early, but suppose that you don't – that the very last combination you check is the right one. How many combinations will you try before finding that correct one?

c) Is your algorithm (circle one): exponential or polynomial

d) You've got other stuff to be doing with your life than flipping switches day in and day out. Using your mad programming skills, you set up a robot to flip the switches for you using that algorithm you designed.

The robot is capable of flicking switches very fast, one every ten milliseconds, but since there are one hundred switches it still takes your robot one whole second to test any particular combination.

How long will it be until your robot solves world hunger? Note that the universe is approximately 13,798,000,000 years old, and that there are 31,557,600 seconds per year.