HW 1B: The Master Theorem

1. Use the master theorem to find the Θ class of algorithms with the following recurrence relations. (You may assume that $T(n)$ is constant for $n = 1$.)

A. $T(n) = 2\,T(n/4) + 1$

B. $T(n) = 2\,T(n/4) + \sqrt{n}$

C. $T(n) = 2\,T(n/4) + n$

D. $T(n) = 2\,T(n/4) + n^2$

2. Use the master theorem to find the Θ class of algorithms with the following recurrence relations. (You may assume that $T(n)$ is constant for $n = 1$.)

A. $T(n) = 2\,T(n/4) + n^4$

B. $T(n) = T(7n/10) + n$

C. $T(n) = 16\,T(n/4) + n^2$

D. $T(n) = 7\,T(n/3) + n^2$

E. $T(n) = 7\,T(n/2) + n^2$

3. Use the master theorem to find a Θ class for each of the following algorithms. Explain your reasoning.

A. Mergesort. Assume the average or worst case

B. Binary search, to find some predefined value within a pre-sorted array. (Assume the average or worst case.)

C. Clustering. An algorithm that takes a collection of n objects, and groups similar objects together. It starts by taking all n objects and splitting it into two clusters of size $\frac{n}{2}$, so that each object is in a cluster of its most similar peers. Assume that this splitting-in-half process can be done in $\Theta(n^3)$ time. It then recurses on each side, so each half is again split in half—all the way down until each object is by itself in a "cluster" of size 1.

D. Ternary search. This is a technique to efficiently find the maximum of a unimodal array of doubles. (Every value in the array is along a smooth continuum with its neighbors, and there is only one overall maximum point—there are no local maxima made of "baby hills" in the arrays.) Here's how it works:

- Find an element `i` that is one-third of the way through the array, and `j` that is two-thirds of the way through.
- If `array[i] < array[j]`, the maximum cannot be in the first third. Recurse on the last two-thirds.
- If `array[i] > array[j]`, the maximum cannot be in the last third. Recurse on the first two-thirds.
- If `array[i] = array[j]`, the maximum must be in the middle third. Discard the outer thirds and recurse on the middle.

Assume each comparison of two elements can be done in constant time. What is the Θ class of this algorithm?

Est. 1888 UNIVERSITY of PUGET SOUND

E. Karatsuba's algorithm for fast integer multiplication. Usually integer multiplication of two n-bit numbers is $\Theta(n^2)$, since you must multiply every bit in one number by every other bit in the other number. If bit shifts and addition are cheap enough, Karatsuba's algorithm can speed this up.

- Let the two n-bit numbers be a and b. Split the bits of the two numbers in half, separating the most-signficant bits from the least-significant bits. Call the 4 resulting $\frac{n}{2}$-bit numbers a_H, a_L, b_H, and b_L. (H is for "high" bits, L for "low".) Then, where $\ll$ is left bit-shifting:

$$a = a_H \ll \frac{n}{2} + a_L \qquad b = b_H \ll \frac{n}{2} + b_L$$

- The product ab is thus:

$$\begin{aligned} ab &= (a_H \ll \frac{n}{2} + a_L) \times (b_H \ll \frac{n}{2} + b_L) \\ &= (a_H b_H) \ll n + (a_H b_L + b_H a_L) \ll \frac{n}{2} + a_L b_L \end{aligned}$$

  This formulation has 4 multiplications.
- However, the middle term can be reformulated as:

$$(a_H b_L + b_H a_L) = (a_L + a_H)(b_L + b_H) - a_H b_H - a_L b_L$$

  Since $a_H b_H$ and $a_L b_L$ need to be calculated anyway, this lets us replace 4 multiplications with 3.
- The three multiplications to be done are all on $\frac{n}{2}$-bit numbers, and can be done recursively. Multiplying single bits can be done in constant time.

Assume that the various addition and shifting can be done in linear time, and that n is a power of 2. What is the Θ class?

Est. 1888
UNIVERSITY of PUGET SOUND