CSC236 tutorial exercises, Week #9
sample solution

These exercises are intended to give you some practice applying the Master Theorem[1] to algorithm design.

1. Consider the following sketch of a divide-and-conquer algorithm $r(s)$ for reversing a string:

   (a) s is a string.

   (b) If $\text{len}(s) < 2$, return s

   (c) Else, partition s into three roughly equal parts: prefix s_1, suffix s_3, and mid-section s_2, and return $r(s_3) + r(s_2) + r(s_1)$.

   (d) You may assume that the time complexity of string concatenation of $s_3 + s_2 + s_1$ is proportional to $\text{len}(s_3) + \text{len}(s_2) + \text{len}(s_1)$

   Use the Master Theorem to find the asymptotic time complexity of function r in terms of $\text{len}(s)$. Be sure to show all the components of your analysis, including the values of a, b, and d. How does this compare to the complexity of simply copying the string elements in reverse order, using a loop?

   sample solution: The algorithm divides the “problem” into 3 (roughly) equal parts, calls the function recursively 3 times on those parts, divides the problem in constant time, and combines the result in time proportional to $\text{len}(s) = n$. This means $b = 3, a = 3, d = 1$. Since $a = 3 = 3^1 = b^d$, I consult the middle branch of the Master Theorem and conclude that the worst-case time complexity is in $\Theta(n \log_3 n)$. Copying the original string backwards has complexity only $\Theta(n)$, so this divide-and-conquer algorithm has worse algorithmic time complexity.

2. Describe a ternary version of MergeSort where the list segment to be sorted is divided into three (roughly) equal sub-lists, rather than two. Use the Master Theorem to find the asymptotic time complexity of your ternary MergeSort in terms of the length of the list segment being sorted, and compare/contrast it with the version we analyzed in class. Be sure to show all the components of your analysis, including the values of a, b, and d.

   sample solution: A sketch of a ternary sort algorithm for list of comparables A

   (a) If $\text{len}(A) < 2$ return A

   (b) Otherwise partition A into three roughly equal, consecutive, sublists: A_1, A_2, A_3.

   (c) ternary Mergesort A_1, A_2, and A_3

   (d) three-way merge the now-sorted A_1, A_2, A_3.

[1] Very abbreviated version on next page...

This approach divides the problem into 3 roughly equal parts, calls the algorithm recursively 3 times on those parts, takes constant time for splitting, but time proportional to len(A) for the merging. Thus $a = 3, b = 3, d = 1$, and $a = 3 = 3^1 = b^d$, so our worst-case asymptotic time complexity is in $\Theta(n \log_3 n)$, which is equivalent to $\Theta(n \lg n)$ for standard MergeSort, since $lg(n) = lg(3) \times \log_3(n)$ — a constant multiple.

3. Consider the following sketch of bisection algorithm $bis(f, a, b, \gamma, \delta)$ to approximate a root of a function:

   (a) $f : \mathbb{R} \mapsto \mathbb{R}$ is a function, $a, b \in \mathbb{R}$ with $f(b) \times f(a) \leq 0$, $\gamma, \delta \in \mathbb{R}^+$

   (b) If $|b - a| < \gamma$ return $(a + b)/2$.

   (c) If $|f(a)| < \delta$ return a.

   (d) If $|f(b)| < \delta$ return b.

   (e) If $f(a) \times f([a + b]/2) \leq 0$ return $bis(f, a, (a + b)/2, \gamma, \delta)$.

   (f) Otherwise return $bis(f, (a + b)/2, b, \gamma, \delta)$

   Use the Master Theorem to find the asymptotic time complexity of function bis in terms of $|b - a|/\gamma$. Be sure to show all the components of your analysis.

   sample solution: This algorithm divides the problem in half, chooses between various options in constant time, and in the worst case calls the algorithm recursively once. Thus $a = 1, b = 2, d = 0$, so $a = 1 = 1 = 2^0 = b^d$, so the asymptotic complexity is in $\Theta(\lg(n))$

$$T(n) = \begin{cases} k & \text{if } n \leq b \\ a_1 T(\lceil n/b \rceil) + a_2 T(\lfloor n/b \rfloor) + f(n) & \text{if } n > b \end{cases}$$

$$T(n) \in \begin{cases} \theta(n^d) & \text{if } a < b^d \\ \theta(n^d \log_b n) & \text{if } a = b^d \\ \theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$