CS311: Midterm Appeal

October 29, 2013

Professor Lathrop Section 3

Josh Davis

Problem 8

Part B
Use the master theorem to solve the recurrence to get an upper bound (Big O) running time of the algorithm.

Reasoning
Although the Skiena textbook that we are using doesn't mention it, *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein mentions the idea of *polynomially* bigger/smaller on page 94 of the third edition.

In the first case, $f(n)$ must be polynomially smaller than the function $O(n^{\log_b a - \epsilon})$. Which means $f(n)$ must be asymptotically smaller than $n^{\log_b a}$ by a factor of some n^ϵ and $\epsilon > 0$.

In our case, $f(n) = \Theta(1)$ and $O(n^{1-\epsilon})$. When I stated that: "$\Theta(1) \in O(n^{1-\epsilon})$ and polynomially bigger" I was just making a more general statement in that $\Theta(1)$ is smaller than the bigger $O(n^1 \cdot n^\epsilon)$ for any $0 < \epsilon < 1$. I didn't give a ϵ because there are an inifinite number of them and I felt that saying "polynomially bigger" implies it.

Problem 10

Part A
Give an example of a hash function $h(x)$ that is guaranteed to give non contant look up performance if used in a hash table.

Reasoning
Altering the buckets that the items are stored in when using a hash table isn't the only way to destroy constant look up time. By using a hashing function that has an absurd complexity will also destroy the look up time. I just used the Fibonacci function because it very clearly runs in $O(2^n)$. Thus when a lookup is called, it will hash the value once in exponential time, then see if it exists.