Name: **EID:**

Quiz #11

Problem 1: Polynomial Time Algorithms

Assume you are given an algorithm A that does some polynomial time work *and* makes some calls to subroutines B_i, each of which does some polynomial time work.

1. Show that if A makes a constant number of calls to subroutines (i.e., it calls each of $B_1, B_2, \ldots, B_c$ once, where c is a constant), then A runs in polynomial time overall.

Solution

Assume that algorithm A runs in polynomial time, not counting calls to subroutines. That is to say A has worst case asymptotic complexity $O(n^k)$ for some k, again, not counting calls to subroutines. A makes a constant numbrer c of calls to polynomial time $B_1, B_2, \ldots, B_c$, each of which has worst-case asymptotic complexity $O(n^s)$, for some constant s. The size of the outputs of the subroutines must be polynomial in the size of the inputs (otherwise the subroutines wouldn't run in polynomial time); in fact, the size of the outputs of the subroutines is $O(n^s)$. Assume A is run with an input of size n. It then runs subroutine B_1, with an input size n_1. How large can n_1 be? Since A runs in $O(n^k)$ time, it follows that n_1 must also be $O(n^k)$. Assume that n_1 is exactly n^k. How long does B_1 take? It takes $O(n_1^s)$ time. Since B_1 runs in $O(n_1^s)$, the size of its output is at most n_1^s. By similar reasoning, B_2 takes time $O(n_2^s)$. We continue this for c steps. How long does this take in total? It takes the time for A pus the time for each of the subroutines. This is $O(n^k)+O(n_1^s)+\cdots+O(n_c^s)$. Since each $n_i = O(n_{i-1}^k)$, the total time is $O((((n^k)^s)^s...)^s) = O(n^{ks^c})$ because there are at most c of the s exponents. This is polynomial in n because c is a constant.

2. Show that if A makes a polynomial number of calls to subroutines (i.e., c in the above is not a constant but is some polynomial function of n), then A **does not** run in polynomial time overall.

Solution

Taking the above argument and replacing c with n^m shows that A might take exponential time if it makes a polynomial number of calls to polynomial time subroutines.
To better explain, here is a simple example:

```
def B(a):
    b = 0
    for i in range(a * 2):
        b += 1
        return b
def A(x):
    a = 1
    for i in range(x):
        a = B(a)
```