CS 205: Final Exam Question 3 - Factor Decomposition

16:198:205

Decomposing a number N into composite factors (e.g., $2059 = 29 * 71$) is generally understood to be a hard problem for large values of N. For some N of specific forms, factoring can actually become quite easy; observing that $40467 = 226^2 - 103^2$, we have via the difference of squares formula

$$40467 = 226^2 - 103^2 = (226 - 103) * (226 + 103) = 123 * 329. \quad (1)$$

One special case of an easily factorable number is when N is a perfect power of some integer base, i.e., $N = a^k$ for some integer a, $k > 1$.

Consider the question, "is $N = 10200$ a perfect square?" One way to check might be to compute the square root. Since $\sqrt{10200} \approx 100.995$, as the fractional part is not 0, N cannot be a perfect square. This, however, requires computation over the real numbers, a topic largely untouched in this course. The goal of this problem is to approach this computation, purely in terms of integer arithmetic.

In each of the following, when we ask for a big-O bound, we are interested in as tight an upper bound as you can justify.

1) Give a big-O bound for the number of bits needed to represent a number N.

2) Give a big-O bound for the bit-complexity of multiplying two integers between 1 and N. What is the worst case? Note, the bound should be simply in terms of N.

3) Given that $1 \leq \sqrt{N} \leq N$, consider sequentially taking each number $a \in \{1, 2, 3, \ldots, N-1, N\}$ and computing a^2. Stop when either i) $a^2 = N$ and you have found the square root, or ii) $a^2 > N$ and N does not have an integer square root. Give a big-O bound for the overall complexity of this search in terms of N.

4) We are effectively searching the set $\{1, 2, 3, \ldots, N-1, N\}$ for the value $\sqrt{N}$. For a given a, we can't compare a to $\sqrt{N}$ since we don't know $\sqrt{N}$. However, we can compare a^2 to N. Use this idea as the basis of a *binary search*-type algorithm. Give a big-O bound on the overall complexity of this search in terms of N. How does it compare to the previous?

5) Given an integer $k > 1$, give a big-O bound for the bit-complexity of computing the k-th power of an integer between 1 and N. What is the worst case? Note that the bound should be in terms of N and k.

6) Similar to Questions 3, 4 above, given that $1 \leq N^{1/k} \leq N$, we may consider searching the sequence $\{1, 2, 3, \ldots, N-1, N\}$ for $N^{1/k}$. For a given a, we may compare a^k to N to determine if we are too high or too low. Use this idea as the basis of a *binary search*-type algorithm. Give a big-O bound on the overall complexity of this search, in terms of N and k.

7) Questions 5, 6 above assumed that k was known. What if k is unknown? If N is a perfect (non-trivial) power, what is the smallest possible value of k that power might be? What is the largest? Give a big-O bound on the largest possible k.

8) Consider repeating the algorithm in Question 6, for every k over the indicated range above, to search all possible powers to see if $N = a^k$ for some a, k. For clarity, write out the pseudocode of this algorithm given an input N. Give a big-O bound on the overall complexity of this search, in terms of N. Is this efficient?

Bonus

9) The approach outlined above suggested a binary search for the base a for any given k, but utilizes a sequential search for the correct value of k. Could a binary search be utilized to find k as well?

10) Can you come up with a better algorithm for checking if N is a perfect (non-trivial) power?

11) Consider modifying the problem so that instead of finding a, k such that $N = a^k$, you are given a large prime number P and asked to find a, k such that $N \equiv a^k \mod P$. What must change in the above approach? Try to outline an algorithm for this problem, given N, P, and given a big-O bound on its complexity. Note, operating mod P, you may assume all multiplications take constant time.