Hyuntaek Oh

Samina Ehsan

CS 514\_400

Oct. 7, 2024

**Homework Assignment Week1: Python Programming and Complexity Analysis**

1. Write an efficient Python function named factors that returns all prime factors of an integer. For example, factors(12) returns [2,2,3]. If the input is a prime or 1 it returns an empty list. The factors should be listed in increasing order.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. Derivation of the running time of the algorithm
2. Assuming that multiplications (and additions) take constant time

In the first loop, the algorithm divides by 2 until cannot be divided by 2. The time complexity of the loop is since the number is exponentially reduced during every dividing by 2. In the second loop, the number , which is not dividable by 2, is divided by odd numbers from 3 to , meaning that the time complexity of the second loop is .

Thus, the total time complexity of the algorithm under the assumption that operations take constant time is:

1. Assuming that multiplication and division of -bit numbers take time and additions and subtractions take time.

The -bit number can be represented in binary. For example, 4-bit number 13 can be represented in 1101, meaning that dividing the number by 2 can be . So, multiplication and division of -bit numbers take time and additions and subtractions take time.

In the loop of dividing by 2, the time complexity of it is since the number is divided by 2 repeatedly, which is as same as (a). Therefore, the total time for the first loop is:

The loop dividing by odd numbers starts from 3 to . The number of iterations keep running while divisor is less than or equal to square root of . Its time complexity is . In the loop, the number is divided by odd numbers, taking time. Thus, the total time for the second loop is:

Finally, the total time complexity is under assumption:

1. Give a table vs. from the experimental results. Does your table closely match one of the running time functions derived in 2? How large can be so that is approximately 5 minutes? What if is 5 hours? 5 days? Factoring is a fundamental crypto-primitive that underlies modern cryptography. What size of makes it practically impossible for your algorithm to factorize, e.g., years?

A table vs. below is from the experimental results.

|  |  |  |
| --- | --- | --- |
|  |  |  |
| 1000000007 |  | 0.004 |
| 10000000019 |  | 0.012 |
| 100000000003 |  | 0.041 |
| 1000000000039 |  | 0.139 |
| 10000000000037 |  | 0.412 |
| 100000000000031 |  | 1.288 |
| 1000000000000037 |  | 3.996 |
| 10000000000000061 |  | 12.758 |

< **Table 1.** Measurement real-time vs. >

The time complexity of the algorithm is , and the empirical data from the table reflects it. As increases 10 times, the runtime increases roughly between 3 to 4 times. This growth is consistent with since the square root of 10 would be 3.xxx.

To estimate the size of for 5 minutes, 5 hours, and 5 days, we need to use the empirical data from the table. When is equal to 10,000,000,000,000,061 (), which is the last data in the table, the running time is 12.76 seconds. Let’s start from 5 minutes (300 seconds). The estimation is below:

For 5 hours (18,000 seconds),

For 5 days (432,000 seconds),

For 10 years (315,360,000 seconds),

1. State a useful invariant of the loop towards proving the correction of the algorithm.

Loop invariants in this algorithm are the current value of , which is the product of the prime factors that have not yet been added to the list at the start of each iteration of the loop, and the list contains all prime factors of the original input that has been discovered through the algorithm. So, we can understand it like:

, where the elements are in the list

Initialization: Before any loops, is the product of the prime factors, and the list does not have no prime factor at the start of the algorithm.

Maintenance: After the first iteration, if , the factor 2 is discovered and added to the list , meaning that is divided by 2. After the second iteration, if , the factor works in the same way as dividing by 2 did. The remaining still holds the remaining prime factors, so the invariant is maintained. On the other hand, if , the factor increases, and the invariant still holds because the factor is not part of , and nothing is added to the list .

Termination: When all the loops are finished, the remaining is a prime factor, which is greater than , and added to the list . Then, if there is only one prime factor, the algorithm returns an empty list.

1. Prove that the algorithm is correct using your previously defined invariant

Based on the previously defined invariant, we need to prove three stages: initialization, maintenance, and termination. In the initialization, the number is initial input that has not been divided, and a list is an empty list that has not discovered nothing. Since original is the product of all its prime factors, and no prime factors have been discovered yet, the invariant holds at this stage.

In the maintenance, there are two iteration loops: the first is division by 2, and the second is division by odd numbers. Before the first loop, if is dividable by 2 as a factor of , 2 will be added to the list . After the iteration, is updated and consists of remaining prime factors, and then the list contains 2 as a factor of . Before the second division by odd numbers, if is divisible by the factor , which is one of the odd numbers and can be a factor of . After dividing by , is still the product of the remaining prime factors, and contains all previously discovered prime factors including the new one, . If cannot divide , the next odd number will be , and loop invariant still holds since the previous factor is not a part of , and there is no change in the list .

Finally, in the termination, after all the loops are finished, the remaining has only the prime number that is greater than since the loops iterates under the condition . The last remaining prime number is inserted into the list if there is not only one prime factor, keeping the loop invariant holding.

Thus, the algorithm yields a list of all prime factors of initial value of .