Weston Jones 2/1/18

Intro to Algorithms – Section 2 HW2

**Q1. (10 points; 5 points per part) Give an algorithm (pseudo code, with explanation) to compute 2^2^n in linear time, assuming multiplication of arbitrary size integers takes unit time. What is the bit-complexity if multiplications do not take unit time, but are a function of the bit-length.**

I created a table of values for 2^2^n using python to look for patterns:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| N | 0 | 1 | 2 | 3 | 4 |
| 2^2^n | 2 | 4 | 16 | 256 | 65536 |

The sequence starts at 2 and then each successive term is just the previous squared, so I wrote this code:

Foo(n):

Result = 2

For i = 0 to n

Result = result \* result

Return result

If multiplication is a O(1) operation, there are only n multiplications to do, so runtime should be linear.

If multiplication is a function of number size (I’m going to say O(n^2), since that’s the fastest multiplication algorithm we know so far), then O(n) multiplications times O(n^2) per multiplication = O(n^3) overall runtime.

**Q2. (10 points total; 5 points per part) Consider the problem of computing N! = 1 · 2 · 3···N.  
(a) If N is an n-bit number, how many bits long is N! in O() notation (give the tightest bound)?**

I wrote n! as a function of just n like so:

N! = 1 \* 2 \* 3 \*4 \* … \* n

Ln(n!) = ln(1 \* 2 \* 3 \* 4 \* … \* n)

Ln(n!) = ln(1) + ln(2) + ln(3) + ln(4) + … + ln(n)

This simplifies down to n! = nln(n) – n

Log base 2 of the above expression should give the approximate number of bits needed to store n! as a function of n.

**(b) Give an algorithm to compute N! and analyze its running time.**

Foo(n)

Result = 1

For i = 1 to n

Result \*= result \* i

The above function needs to do n multiplications. We know that multiplication is O(n^2) operation in terms of bit complexity, so total runtime of the above code is O(n^3).

**Q3. (10 points; 5 points per part) Find the GCD of 1492 and 1776, using**

**a) the prime factorization method and using Euclid's method, and**

Prime Factorization:

1492 = 2 \* 746 = 2 \* 2 \* 373

1776 = 2 \* 888 = 2 \* 2 \* 444 = 2 \* 2 \* 2 \* 222 = 2 \* 2 \* 2 \* 2 \* 111 = 2 \* 2 \* 2 \* 2 \* 3 \* 37

The prime factorization of 1492 is 22 \* 373

The prime factorization of 1776 is 24 \* 3 \* 37

The largest number we can make using the prime factors that is common to both 1492 and 1776 is 22 or 4, so 4 is the GCD of 1492 and 1776.

Euclid’s Method:

|  |  |  |  |
| --- | --- | --- | --- |
| A | B | R (a mod b) | q |
| 1776 | 1492 | 284 | = 1776 - 1492 |
| 1492 | 284 | 72 | = 1492 – 5\*284 |
| 284 | 72 | 68 | = 284 – 3\*72 |
| 72 | 68 | 4 | = 72 – 68 |
| 68 | 4 | 0 |  |

4 is the number than results in a remainder of 0, so 4 is the GCD of 1492 and 1776.

**b) express the GCD as an integer linear combination of the two inputs.**

Using, the extended Euclid’s algorithm and the work above, we have:

4 = 72 – 68

4 = 72 – (284 – 3\*72)

4 = -284 + 4\*72

4 = -284 + 4(1492-5\*284)

4 = 4\*1492 – 21\*284

4 = 4\*1492 – 21(1776 – 1492)

4 = -21\*1776 + 25\*1492