Questions

1. In lecture 1, I asked you to consider the slow multiplication problem in which one computes $a * b$ by adding a to itself b times, i.e. $a + a + a + ... + a$. Assuming that a and b both have N digits, how does the time taken for this algorithm depend on the number of digits N.

   Hint: the time for the fast multiplication algorithm grows as N^2, where we ignore constants and terms that depend only on N. The time for the slow multiplication algorithm is *not* N^2.

2. How is grade school multiplication so much faster than slow multiplication by repeated sums? What trick does grade school multiplication use so that the number of steps grows as N^2 rather than $N * 10^N$?

3. Convert the following decimal numbers to binary.

   (a) 34

   (b) 82

4. Show that 26 + 27 = 53, by converting 26 and 27 to binary, computing the sum in binary, and then converting back to decimal.

5. A *byte* is $n = 8$ bits. When the byte is interpreted as a positive 8 bit integer, the values range from 0 to 255. Write out the bytes (binary numbers) that correspond to integers 127 to 130.

6. Integers are typically represented in a computer using a fixed number of bits, namely 8, 16, 32, or 64. In Java, these correspond to primitive types *byte*, *short*, *int*, and *long.*

   To represent positive integers only, some languages (C, but not Java) allow you to declare the type to be *unsigned*, e.g. *unsigned int.*

   What is the largest positive number that can be represented (using *unsigned* integer) for each of the above lengths, and roughly how much is each when it is written in decimal?

   Hint: Use the fact that $2^{10} \approx 10^3$.

7. Convert $(736)_8$ and $(1205)_8$ from base 8 to decimal.

8. Perform the sum $(1205)_8 + (736)_8$.

9. Perform the sum: $(2430)_5 + (2243)_5$.

10. Convert 238 from decimal to base 5.

11. Convert the following hexadecimal numbers to decimal:

    (a) `11`

    (b) `25`

    (c) `FF`

    (d) `10A`

Note sometimes one writes hexadecimal numbers starting with `0X` but this is not necessary if it is clear from the context that the numbers in hexadecimal.

12. Carry out the following sums, assuming the numbers are written in base 16 (hexadecimal):

    (a) `9` + `3`

    (b) `3D` + `10`

    (c) `FA` + `D0`

    (d) `DD` + `DD`

13. Rewrite the expression $\log_4(2n^{16})$ to simplify the argument of the log function.

14. Evaluate $\log_{16}(2)$.

15. Consider the number `0xabc`. What is this number mod 8 ? Write your answer in base 10.

16. Suppose an integer has n_1 digits when represented in base β_1. Roughly how many digits would it have when written in base β_2 ?

17. The "slow division" algorithm method repeatedly subtracts the divisor `b` from the dividend `a`. Suppose the dividend and divisor have n_a and n_b digits (base 10) respectively, where $n_a > n_b$. What is the 'big O' time complexity the algorithm ?

    Hint: you need to express this in terms of both n_a and n_b.

Answers

1. *[ASIDE: This is the most challenging question in the bunch. If the solution below is difficult to wrap your head around right now, then don't worry!]*

   The slow multiplication algorithm uses a `for` loop, where we loop b times. Since b has N digits, b itself is a number between 10^{N-1} and 10^N. (For example, if $N = 3$, then b is between 100 and 999.) So, the slow multiplication algorithm loops between 10^{N-1} and 10^N times.

   In each pass through the loop, we add the number a to the accumulated total sum. The number a has N digits, and so adding a involves about N steps. Therefore, the total number of steps for slow multiplication takes roughly between $N * 10^{N-1}$ and $N * 10^N$ steps. Here we are ignoring constants and only paying attention to terms that depend on N. Note that slow multiplication is indeed *extremely* slow since $N * 10^{N-1}$ is a much bigger number than N^2 when N is large!

2. Grade school multiplication uses the trick that if we are given the number *a in its decimal representation*, then we can compute $10 * a$ or $100 * a$ etc very easily. We don't need to add a to itself 10 times, or 100 times, etc. Rather we can just tack on 0's. That is essentially what you are doing with grade school multiplication when you have the jagged *tmp* matrix which you sum up. Your grade school teacher explained that to you back in grade 3. You might have appreciated it more if she had given you the slow multiplication algorithm and given you examples with $N = 3$ or more.

3. (a) $(100010)_2 = 2^5 + 2^1 = 32 + 2$

   (b) $(1010010)_2 = 2^6 + 2^4 + 2^1 = 64 + 16 + 2$

4. $$\begin{array}{ll} & 00011010 \leftarrow 26 \\ + & \underline{00011011} \leftarrow 27 \end{array}$$

   To perform the addition, use the same algorithm that you use with decimal, except that you are only allowed 0's and 1's. Whenever the sum in a column is 2 or more, you carry to the next column, since it contributes to the next power of 2:

   $$2^i \ + \ 2^i = 2^{i+1}$$

   So,

   $$\begin{array}{lll} & 00110100 & \leftarrow \text{carry bits} \\ & 00011010 & \leftarrow 26 \\ + & \underline{00011011} & \leftarrow 27 \\ & 00110101 & \leftarrow 53 \end{array}$$

   This is basically how computers do arithmetic as you will learn in COMP 273.

5. 127 is 01111111, 128 is 10000000, 129 is 10000001, 130 is 10000010

6. The largest positive integer you can represent with N bits is $2^N - 1$. Since $2^{10} \approx 10^3$, we have
   - $2^8 - 1 = 255$ as we said in the previous question
   - $2^{16} - 1 \approx 2^6 * 10^3 \ = \ 64,000$
   - $2^{32} - 1 \approx 2^2 * 2^{30} \ = \ 2^2 * (10^3)^3 \ = \ 4 * 10^9$
   - $2^{64} - 1 \approx 2^4 * 2^{60} \ = \ 16 * (10^3)^6 \ = \ 16 * 10^{18}$ which is a very big number.

7.
$$(736)_8 = 7 * 8^2 + 3 * 8 + 6 = (478)_{10}$$
$$(1205)_8 = 1 * 8^3 + 2 * 8^2 + 0 * 8 + 5 = (645)_{10}.$$

8.
```
   101
  (1205)_8
+ (736)_8
-------
  (2143)_8      which is 1123 in base 10.
```

9. The answer is $(10223)_5$.

```
   110    <--- carry into the next column (next power of 5)
   2430
  +2243
   ----
  10223
```

If you would like to practice converting between different bases, see the website:
`http://www.cleavebooks.co.uk/scol/calnumba.htm`

10. Apply the same algorithm we saw for binary, but now we divide by 5 at each step and write down the remainder.

```
             decimal      base 5 coefficients
i              238            a[i]
-             -----           ----
0               47              3
1                9              2
2                1              4
3                0              1
```

and so $(238)_{10} = (1423)_5 = 1 * 5^3 + 4 * 5^2 + 2 * 5^1 + 3 * 5^0 = 125 + 100 + 10 + 3.$

11. (a) 17
    (b) 37
    (c) 255
    (d) 266

12. (a) C
    (b) 4D
    (c) 1CA
    (d) 1BA

13. $\frac{1}{2} + 16 \log_4 n$

14. The answer is $\frac{1}{4}$. Why? Note $16 = 2^4$. So the question is asking, what is x where $(2^4)^x = 2^1$. But note $(2^4)^x = 2^{4x}$.

15. The answer is 4.

    For this question, it is important that you realize you will only need to consider the last digit in the number, namely `c`. To understand why, think of the binary representation of the number. The last four bits are determined by the last hex digit `c`, namely `1100` which is 12 in decimal. So, the last three of these bits `100` is the original number mod 8. And `100` in decimal is 4.

    [Updated: Jan. 17] I edited the solutions for Q16 and Q17 below.

16. For the intuition here, think about converting from base 16 to base 2. The number of bits (base 2) will be 4 times the number of hex digits, since every hex digit requires 4 bits to encode it ($16 = 2^4$). Similarly when converting from base 8 to base 2, you triple the number of coefficients i.e. octal digits to bits ($8 = 2^3$).

    The calculation is less obvious when the bases are not both powers of 2 and for that we need to do some math. Following arguments at the start of lecture 3, if integer m has n_1 digits when represented in base β_1, then $\beta_1^{n_1-1} \leq m < \beta_1^{n_1}$. So,

$$n_1 = floor(\log_{\beta_1} m) + 1$$

$$n_2 = floor(\log_{\beta_2} m) + 1$$

    Using the log property, $\log_b x = log_b a * log_a x$, we have

$$n_2 = floor(\log_{\beta_2} \beta_1 \; \log_{\beta_1} m) + 1$$

    so

$$n_2 \approx n_1 \log_{\beta_2} \beta_1.$$

17. Let's solve the problem approximately. `a` and `b` are approximating 10^{n_a} and 10^{n_b}, respectively (approximately means within a factor of 10 in this case, or "an order of magnitude"), and so `a/b` is approximately $10^{n_a - n_b}$. So the slow division algorithm will subtract `b` from `a` about $10^{n_b - n_a}$ times. The number of steps for each subtraction is proportional to the number of digits of b, namely n_b. So the answer is $O(n_b \; 10^{n_a - n_b})$.