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 multiplcation 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. The algorithm for addition of two positive integers is as easy with the binary representation as it is with the decimal representation.

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

5. A *byte* is $n = 8$ bits. When the bits are interpreted as positive 8 bit integers, 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. We represent numbers in base 8 as follows:

   $$(a[N-1] \cdots a[2]\ a[1]\ a[0]\ )_8 \ = \ a[N-1] * 8^{N-1} + \cdots + a[2] * 8^2 + a[1] * 8 + a[0]$$

   where the $a[i]$ are all in $\{ 0, 1, 2, \cdots, 7\}$.

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

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

9. Perform the following sum. The two numbers are represented in base 5:

   $$(2430)_5 + (2243)_5 \ .$$

   Do not convert the numbers from base 5 to decimal!

10. Convert 238 from decimal to base 5, that is, write 238 as a sum of powers of 5.

Answers

1. The slow multiplication algorithm uses a `for` loop, where we loop b times. Since b has N digits, it 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 at least 10^{N-1} 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. So this gives a total number of steps that grows with $N * 10^{N-1}$, where we are ignoring constants and only paying attention to terms that depend on N. The point is that slow multiplication is indeed *extremely* slow.

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,

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

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