CS106B
Summer 2013

Handout #13
July 15, 2013

Section Handout 3

Problem One: CHeMoWIZrDy

Some words in the English language can be spelled out using just element symbols from the Periodic Table. For example, the word "began" can be spelled out as **`BeGaN`** (beryllium, gallium, nitrogen), and the word "feline" can be spelled out as **`FeLiNe`** (iron, lithium, neon). Not all words have this property, though; the word "interesting" cannot be made out of element letters, nor can the word "chemistry" (though, interestingly, the word "physics" can be made as **`PHYSICS`** (phosphorous, hydrogen, yttrium, sulfur, iodine carbon, sulfur)

Suppose that you are given a **`Lexicon`** containing all the element symbols in the periodic table. Write a function

```
bool isElementSpellable(string text, Lexicon& symbols);
```

that accepts as input a string, then returns whether that string can be written using only element symbols. If you'd like, you can use the fact that all element symbols are at most three letters.

Problem Two: Big-O Notation

Below is a simple function that computes the value of m^n when n is a nonnegative integer:

```
int raiseToPower(int m, int n) {
    int result = 1;
    for (int i = 0; i < n; i++) {
        result *= m;
    }
    return result;
}
```

i. What is the big-O complexity of the above function, written in terms of m and n? You can assume that it takes the same amount of time to multiply together any two numbers.

ii. If it takes 1μs to compute **`raiseToPower(100, 200)`**, about how long will it take to compute **`raiseToPower(50, 400)`**?

Below is a recursive function that computes the value of m^n when n is a nonnegative integer:

```
int raiseToPower(int m, int n) {
    if (n == 0) return 1;

    return m * raiseToPower(m, n - 1);
}
```

iii. What is the big-O complexity of the above function, written in terms of m and n? You can assume that it takes the same amount of time to multiply together any two numbers.

iv. If it takes 1μs to compute **`raiseToPower(100, 200)`**, about how long will it take to compute **`raiseToPower(50, 400)`**? Why can't you give an exact value for the runtime?

It turns out that there is a much faster way to compute m^n when n is a nonnegative integer. The idea is to modify the recursive step as follows.

- If n is an even number, then we can write as $n = 2k$. Then $m^n = m^{2k} = (m^k)^2$
- If n is an odd number, then we can write $n = 2k + 1$. Then $m^n = m^{2k+1} = m \cdot (m^{2k}) = m \cdot (m^k)^2$

Based on this observation, we can write this recursive function:

```
int raiseToPower(int m, int n) {
    if (n == 0) return 1;

    if (n % 2 == 0) {
        int z = raiseToPower(m, n / 2);
        return z * z;
    } else {
        int z = raiseToPower(m, n / 2);
        return m * z * z;
    }
}
```

v. What is the big-O complexity of the above function, written in terms of m and n? You can assume that it takes the same amount of time to multiply together any two numbers.

vi. If it takes 1μs to compute `raiseToPower(100, 100)`, about how long will it take to compute `raiseToPower(50, 10000)`?

vii. *(Challenge problem, if you have the time)* What happens to the big-O time complexity if you rewrite the function in the following way?

```
int raiseToPower(int m, int n) {
    if (n == 0) return 1;

    if (n % 2 == 0) {
        return raiseToPower(m, n / 2) * raiseToPower(m, n / 2);
    } else {
        return m * raiseToPower(m, n / 2) * raiseToPower(m, n / 2);
    }
}
```