2) (5 pts) ALG (Hash Tables)

There are two hash functions that take in strings as input shown below. Each returns an integer in between 0 and 1,000,002. (Note: 1,000,003 is a prime number.) Which of these two is a better hash function? Explain the weakness in the other function.

```
int f1(char* str) {

    int i = 0, res = 0;
    while (str[i] != '\0') {
        res = (256*res + (int)(str[i]))%1000003;
        i++;
    }
    return res;
}

int f2(char* str) {

    int i = 0, res = 0;
    while (str[i] != '\0') {
        res = (res + (int)(str[i]))%1000003;
        i++;
    }
    return res;
}
```

The better function is f1. This function is expressing the string in base 256, mod 1000003. The other function, f2, is simply adding the Ascii values of the individual characters of the string mod 1000003. One reason this is problematic is that the sum of the Ascii values of most strings tends to be pretty small. Each individual value is near 100 and most everyday strings have no more than 12 or 13 letters. So the probability of getting a hash value less than 1200 is extremely high and the probability of getting a hash value in between 1200 and 1,000,002 is extremely low. A good hash function has a roughly equal probability that any of the hash values will be generated on an arbitrary input. Secondly, any strings that are anagrams of one another are definitively going to create a collision, since order of the characters in a string doesn't at all affect the hash function value returned by f2. Good hash functions are such that there's no formulaic way to create two different inputs that map to the same output in a reliable way.

Grading: There is no need to have the depth of this response for full credit. Give 2 pts for answering that f1 is better, and upto 3 pts for the flaws of f2. Any cogent response which explains why the hash values are unlikely to be equally distributed should get full credit. Decide partial as necessary. Max score for those who pick f2 is 1 out of 5.