1) (10 pts) ANL (Algorithm Analysis)

What is the run-time of the function `hash_func` shown below, in terms of *n*, the length of its input string? Please provide sufficient proof of your answer. (9 out of the 10 points are awarded for the proof. 1 point is awarded for the answer.)

```
#include <stdio.h>
#include <string.h>
#define MOD 1072373
#define BASE 256

int hash_func(char* str);
int hash_func_rec(char* str, int k);

int hash_func(char* str) {
    return hash_func_rec(str, strlen(str));
}

int hash_func_rec(char* str, int k) {
    if (k == -1) return 0;
    int sum = 0;
    for (int i=k-1; i>=0; i--)
        sum = (BASE*sum + str[i])%MOD;
    return (sum + hash_func_rec(str, k-1))%MOD;
}
```

The method to analyze the run time of a recursive function is to set up a recurrence relation equal to the run time of the function, and then solve that recurrence relation.

For this example, let T(k) be the run-time of hash_func_rec, where k is the second input parameter.

This function is a wrapper function for the function call hash_func_rec, which is initially called with an input parameter of k = n. We first see that if the input k == -1, the function immediately terminates. If k = 0, there will be a constant amount of work (for loop that runs once then the quick recursive call), thus, T(0) = 1. (Alternatively, one can write T(0) = c, for some constant c. For Big-Oh analysis, either will suffice.)

Otherwise, the for loop runs exactly k times. Then, the function makes a recursive call with the value k-1. It follows that the recurrence relation that governs this function is

T(k) = O(k) + T(k-1), T(0) = 1.

If we iterate the recurrence several times, it can be shown that the pattern is that

T(k) = $\sum_{i=1}^{k} O(i) \le \sum_{i=1}^{k} ci = \frac{ck(k+1)}{2} \in \boldsymbol{O}(k^2)$. Since we are asked to answer the question in terms of the variable n, where n is the length of the input string, the run-time of the algorithm is $\boldsymbol{O}(\boldsymbol{n^2})$.

In general, a recurrence of the form T(n) = T(n-1) + f(n) with a constant value for a small input value of T will have the solution T(n) = $\sum_{i=1}^{n} f(i) + c$, for some constant c. (This can be shown by iterating down to a base case.)

Alternatively, one can note that if we "unroll" the recursion, the code effectively runs a nested set of loops where the first loop runs n times, the second loop runs n-1 times, etc., last loop runs once. From that observation, we obtain the same summation as the one shown above.

Grading: 2 pts for recognizing that the initial recursive call does O(n) work.
2 pts for recognizing that the effective input size to the recursive call is n-1, if the input size of the previous input was n.
2 pts for either setting up the recurrence relation or summation
4 pts for solving the recurrence relation or summation

If an answer of O(n^2) is given without any justification, award 1 pt as stated.)