1. Problem on Hashing

(a) Consider the following hash function.

```c
unsigned int
Hash( const string & key, const int h_size )
{
    unsigned int value = 0;
    for ( int i=0; i<key.length(); i++ )
        value = value*37 + key[i];
    return value % h_size;
}
```

Write the mathematical formula for the hash value returned by this hash function.

**Solution:**

\[
\text{hashvalue} = \left( \sum_{i=0}^{\text{length}-1} 37^i \cdot \text{key[length} - i - 1] \right) \mod h\_size
\]

Equivalently, this can be written as:

\[
\text{hashvalue} = \left( \sum_{i=0}^{\text{length}-1} 37^{\text{length}-i-1} \cdot \text{key}[i] \right) \mod h\_size
\]

(b) Suppose you would like your web browser to automatically maintain a list of URLs of web sites that you visit in order of frequency of visits. Would you use a hash table to store the URLs? Why or why not? Justify your answer in terms of the running time.

**Solution:**

No, a hash table to store the URLs would not be very efficient if we wish to automatically maintain the visited web sites in order of frequency of visits.

We can use a hash table, using the URL of a web site as the key, to keep track of the number of times we visit each web site. However the items are not maintained in sorted order in a hash table. Therefore to display the web sites in sorted order, we would have to sort all \( n \) entries in the hash table, which would take \( O(n \log n) \) time. So each query to find the sorted list of all visited web sites would take \( O(n \log n) \) time, and a query to find the single most frequently visited web site would take \( O(n) \) time.

An AVL tree or an STL map (which uses an underlying balanced binary search tree), using the frequency of visits as the comparison key, would be a more efficient data structure for this task. After the initial \( O(n \log n) \) cost of inserting items into the tree, dynamically updating the frequency data every time a web site is visited takes \( O(\log n) \) time. It requires only \( O(n) \) time to list all the web sites in sorted order, and \( O(\log n) \) time to find the single most frequently visited site.