1. Explain the importance of Hash maps in Computer Science. Ensure that you cite credible academic resources.

Hash maps are another incredibly important data structure that has efficient ways to store, access, update and remove data. Through the use of hashing we can quickly calculate the address of any object stored in memory.
Along with hashing functions hash maps have become an essential part in the world of computer science. Hashing is used in database indexing, compiler design, caching, password authentication, software integrity verification, error-checking and multiple other applications that are vital to our modern systems and applications.

2. How good is the provided hash function -- are we really getting constant time operations with our hash map?
   There are two components in our current hash function. We have an strlen() calculation and a modulus operation. After some research I found that both operations are mainly Big-O(1). According to some resources strlen() of any string runs in constant time because the pointer has a starting point and an ending point that can reveal the size of the string.
   Furthermore, it is widely considered that a mod operation would take close to O(1).

   Therefore our hash function is extremely fast running almost at constant time. Downsides are that there are a great deal of collisions that occur when two keys have the same length, which is not that uncommon in the English language. Maybe it would take longer since we would have to iterate through a linked list over and over again.

   note: some implementations might yield different results.

3. What is one other way you could implement the hash function? Anything creative is acceptable in this answer.
   Use the numeric value of each of the characters inside the string. Use all the digits to form a new integer number. This number has to be larger than a certain value, for example: 9999999. If it is smaller, then we add more digits (contained in our key) until it is equal or bigger.
   If we are still using an array of linked list , we use the mod operator on that integer value and calculate where it should be stored.

   This reduces the chances of a collision greatly, given enough buckets but now it runs on Big-O(n).

4. If I have to resize the hash map to add more buckets, what is the Big-Oh complexity?

   Assuming that we need to reorganize each element in another bucket and even handle new collisions. One could estimate that for each element there is going to be an insert process to relocate this element into a different bucket. Therefore, on average It would take O(n).

5. What is "open addressing" in regards to hash maps and hash tables?

This is used in an alternate implementation of a hash map, in which instead of having an array of linked lists, we use a simple array that can store many key-value pairs.

When two keys are entered into a hashing function and the outcome is the same, we have a collision. We can fix this collision by using something called a probing sequence function to find a new slot in the array.
This probing function can be anything from a linear equation, a quadratic equation or even another hashing function (double hashing). If the slot that is assigned by the probing function is occupied, then we just iterate through the array until it finds an empty spot. (using the result of the probing function as our starting point).

There is a relationship between how many items are currently stored and how many slots there are in total. The lower the number, statistically the more time it takes to find a new slot, this can prove to be very inefficient when handling many collisions.