Data Structures and Algorithms — CSCI 230
Chapter 5 — Review Solutions

1. Consider the following hashing function, similar to hash functions discussed in class.

```
unsigned int
Hash( const string & key, const int h_size )
{
  unsigned int value = 0;
  for ( int i=0; i<key.length(); i++ )
    value = (value + key[i]) << 3;  // multiply by 8
  return value % h_size;
}
```

   Is this a good hash function when `h_size = 128`? Why or why not?

   Solution: No. It is a bad hash function. It only hashes to locations that are multiples of 8.

2. Suppose you need a data structure to support several types of operations on a set of strings: insert a string, find a string, delete a string, and print the strings in order. Consider the possibilities of using either an unbalanced binary search tree, an AVL tree, or a hash table with separate chaining to represent the strings. Even under these assumptions, which data structure is best will depend on the data and on the relative frequencies of insert, find, delete and print operations.

   (a) Under what conditions should you choose a hash table? Why?

   Solution:
   When there are no print operations the hash table will be best. Since there will never be a need to access the items in order, each operation will require on average a constant amount of time, independent of the number of items stored. This is better than the $O(\log n)$ time required by each of the tree operations.

   (b) Under what conditions should you choose a binary search tree? Why?

   Solution:
   This is the best choice when there are enough print operations to offset the constant time accesses to the hash table and the input is random enough that balancing is not necessary.

(c) Under what conditions should you choose an AVL tree? Why?

Solution:
When there are print operations to offset the constant time access for insert, find of hash tables and when the input is close enough to being ordered that an unbalanced tree would result if a regular binary search tree were used.