1. Pseudo codes for methods add, and search(Tuple t) methods from HashTable:

```
        add(Tuple t) {
                integer bucket = hashFunction(t.key)

                add a new list for this bucket if there isn't one

                add the tuple t to the bucket's list

                this.search(t)
                        if result is 0, there are no duplicates,
                        so increment the number of unique tuples
                        in the table

                check load factor, rehash if it is above 0.7
        }

        rehash() {
                create new table with double size

                re-add tuples

                set this table equal to that
        }

        search(Tuple t) {
                integer bucket = hashFunction(t.key)

                check if the list at this bucket exists
                        if not, return 0
                        if so, continue

                bucketLoad = table[bucket].size

                for(i = 0; i < collisions; i++) {
                        if t equals table[bucket].get(i)
                                increment count
                }
        }
```

2.
3.
4.

```
        BFS similarity: 0.00324
                runtime: 8.262 seconds
        HSS similarity: 1.040429
                runtime: 37 seconds
        HCS similarity: 1.72813
```

runtime: 9.369 seconds

5.

HCS would be fastest on a large scale. Using the small input sizes
available to me, there is not a huge difference, though. Obviously
BFS is slowest. BFS has no hashtable of key/value pairs to reference,
and therefore it must continuously loop through the arrays storing
the keys/values and compare ints and strings. With the hash table,
the hashcode can be calculated in constant time, and so it can be looked
up in constant time (usually) in the hash table. The time can be as bad as
the number of elements in the table, depending on how well distributed
tuples are between buckets. HCS would be faster than HSS because
it compares only the integers, which is quicker than comparing both
integers and strings.

6.

No. I think that HashStringSimilarity and BruteForceSimilarity
should return the same thing, based on how I implemented BruteForce.
However, I was having an error I could not find in time to submit.
The values are still similar. In BruteForce, I also checked to make sure
both the string and the hashcode were the same, like in HSS. In HCS,
it checks only the code. Therefore, collisions are counted as the same.
I had a variable amount of collisions, due to the HashFunction randomness,
so HCS's values are different each time.