1.1: Python dict uses open addressing to resolve hash collisions. In open addressing, hash collisions are resolved by probing. The hash table is just a contiguous block of memory (like an array, so you can do O(1) lookup by index).

1.2: Suffix arrays can be constructed by performing a depth-first traversal of a suffix tree. The suffix array corresponds to the leaf-labels given in the order in which these are visited during the traversal, if edges are visited in the lexicographical order of their first character.

2.1: `def respelling(string): respellings = { "teh": "the", "relevent": "relevant", "lite": "light", "lol": "haha" } res = string.split() for i in xrange(len(res)): if res[i] in respellings: res[i] = respellings[res[i]] return ' '.join(res)`

2.2: The efficiency is O(logn) compared to O(n^2)

2.3: We could store the common misspelling in a function and call that function whenever we need to do correction on misspelling.

3.1: Cuckoo hashing utilizes 2 hash functions in order to minimize collisions. The hash table in this particular implementation contains 2 lists, each one using a different hash function. When inserting into a cuckoo hash table, hash the key twice to identify 2 possible "nests" (or "buckets") for the key/data pair.

3.2: The key a is inserted in the Cuckoo Hash table below. Keys get kicked around until a free slot is found. If the number of displacements reaches a certain threshold (for instance due to a cycle among the inserted keys) rehashing takes place. Rehashing is a linear operation, so worst-case complexity is O(n).