HOME WORK 3

PLEAE RUN MY PROGRAM ON A NORMAL COMPUTER ☺.

MY IMPLEMENTATION IS IN ONLY ONE FILE

1. INSERTION

The times are in average

|  |  |  |
| --- | --- | --- |
| NUMBER OF ELEMENTS | LIST TIME(SEC) | DICTIONARY TIME(SEC) |
| 2000 | 0.0 | 0.01 |
| 4000 | 0.0 | 0.03 |
| 8000 | 0.0 | 0.11 |
| 120000 | 0.0 | 0.18 |
| 131071 | 0.0 | 0.21 |
|  |  |  |

LOOKUP

The times are in average

|  |  |  |
| --- | --- | --- |
| NUMBER OF ELEMENTS | LIST TIME(SEC) | DICTIONARY TIME(SEC) |
| 2000 | 0.50 | 0.00 |
| 4000 | 1.2 | 0.00 |
| 8000 | 4.1 | 0.02 |
| 120000 | 19.0 | 1.01 |
| 131071 | 21.6 | 1.38 |

2)I realized that there is a difference in the data structures. The first difference is that, when it comes to insertion, both the dictionary and list where both really fast, but when it comes to look up, the dictionary took a good advantage of hashing and performs faster, its was fast even thought there was a lot of collisions but the probing has helped me in taking care of this issue. On the hand, the list took quite a lot of time when insertion because I had to create two loops for comparing a key with many other elements to update its character. Yes, the runtimes I measured are in line with the corresponding average-case time complexities.

3)No doubt that I realized that the more elements I put, the more collisions happen, and the collision increases really quick, which is exponential. The reason is that when there are more elements in the dictionary and a new element is to be inserted, it will most likely collide because there are already many elements in the dictionary.

4) The worst case is when looking up keys which had a lot of collisions before getting inserted, the corresponding complexity is O(n). In this case what dominates the complexity is the probing, an element will have to be probed many times.