1. Form a hypothesis about how each of the following three factors should affect the runtime of BruteGenerator and MapGenerator in big-O notation and explain your reasoning by referencing segments of your code.

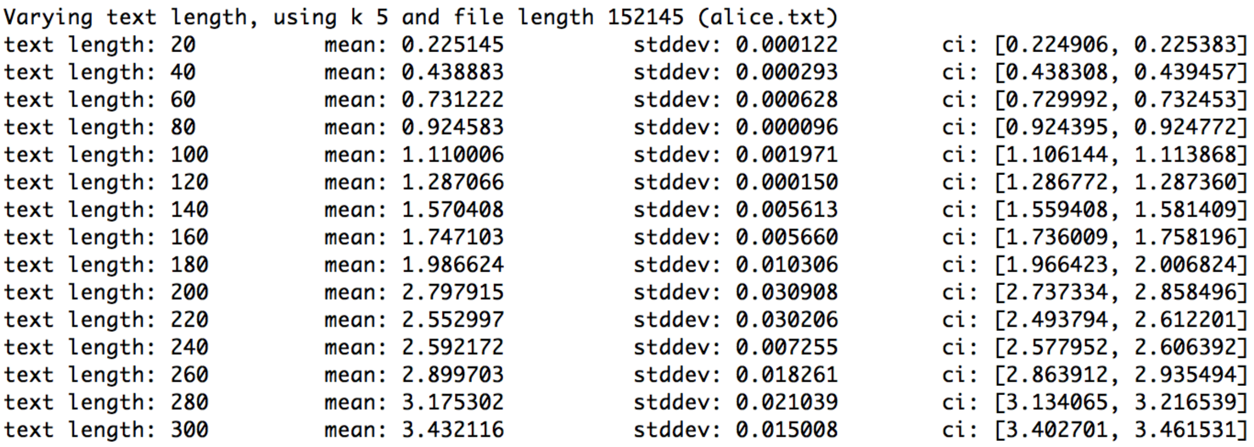
a) the length of the training text
b) the k-value or length of the word
c) the length of the random text

Run the Benchmark class on both BruteGenerator and MapGenerator to get empirical data to test your hypothesis. Running the Benchmark class once should be sufficient to generate quality data. Running Benchmark will likely take a very long time especially for BruteGenerator, so be patient. Compare your empirical results to your hypothesis.

1.
i)I hypothesize that the runtime of BruteGenerator and MapGenerator increases as the length of the training text increases, vice versa.Because This is because if length of text is n, the runtime would be O(n), since for (int j=0; j<text.size()-1;j++){ consists of one for loop iterating once through text.
ii)Assuming the k-value to be k, the runtime is O(1) because there's no for loop in TrainingText, which uses k.
iii)The runtime would be O(n) because for (int i=0;i<length;i++){ consists of one for loop iterating once.

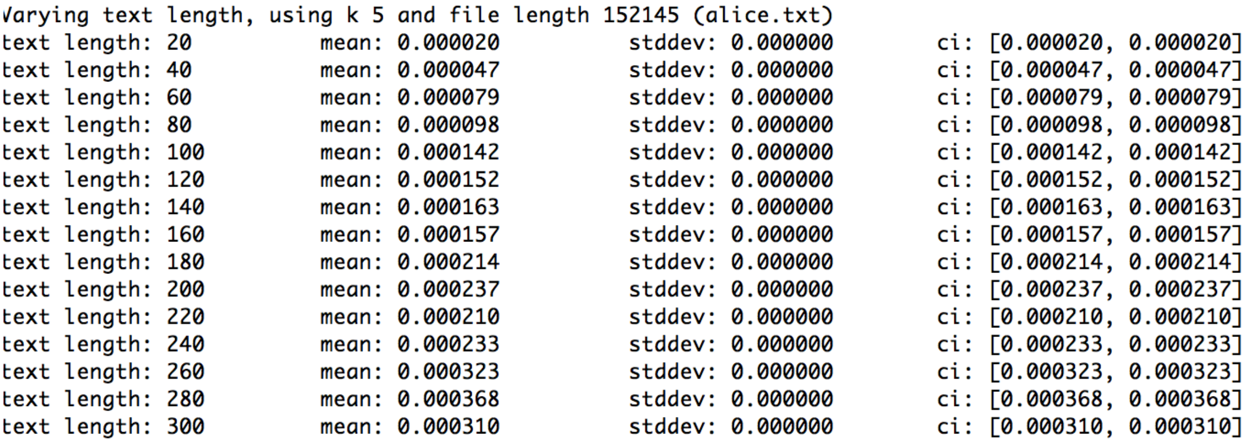
2.
i) The benchmark data in MapGenerator and BruteGenerator supports the hypothesis that big O is o(n), because the mean runtime changes in proportion to the length of the training text. For example, in Brute Generator, when the text length changes from 20 to 40, the mean runtime changes from 0.225145 second to 0.438883 second, as shown in the data table below. In MapGenerator,when text length changes from 20 to 40, the mean runtime changes from 0.000020 second to 0.000047, as shown in the data table below.

Data for BruteGenerator:



```
Varying text length, using k 5 and file length 152145 (alice.txt)
text length: 20         mean: 0.225145          stddev: 0.000122        ci: [0.224906, 0.225383]
text length: 40         mean: 0.438883          stddev: 0.000293        ci: [0.438308, 0.439457]
text length: 60         mean: 0.731222          stddev: 0.000628        ci: [0.729992, 0.732453]
text length: 80         mean: 0.924583          stddev: 0.000096        ci: [0.924395, 0.924772]
text length: 100        mean: 1.110006          stddev: 0.001971        ci: [1.106144, 1.113868]
text length: 120        mean: 1.287066          stddev: 0.000150        ci: [1.286772, 1.287360]
text length: 140        mean: 1.570408          stddev: 0.005613        ci: [1.559408, 1.581409]
text length: 160        mean: 1.747103          stddev: 0.005660        ci: [1.736009, 1.758196]
text length: 180        mean: 1.986624          stddev: 0.010306        ci: [1.966423, 2.006824]
text length: 200        mean: 2.797915          stddev: 0.030908        ci: [2.737334, 2.858496]
text length: 220        mean: 2.552997          stddev: 0.030206        ci: [2.493794, 2.612201]
text length: 240        mean: 2.592172          stddev: 0.007255        ci: [2.577952, 2.606392]
text length: 260        mean: 2.899703          stddev: 0.018261        ci: [2.863912, 2.935494]
text length: 280        mean: 3.175302          stddev: 0.021039        ci: [3.134065, 3.216539]
text length: 300        mean: 3.432116          stddev: 0.015008        ci: [3.402701, 3.461531]
```

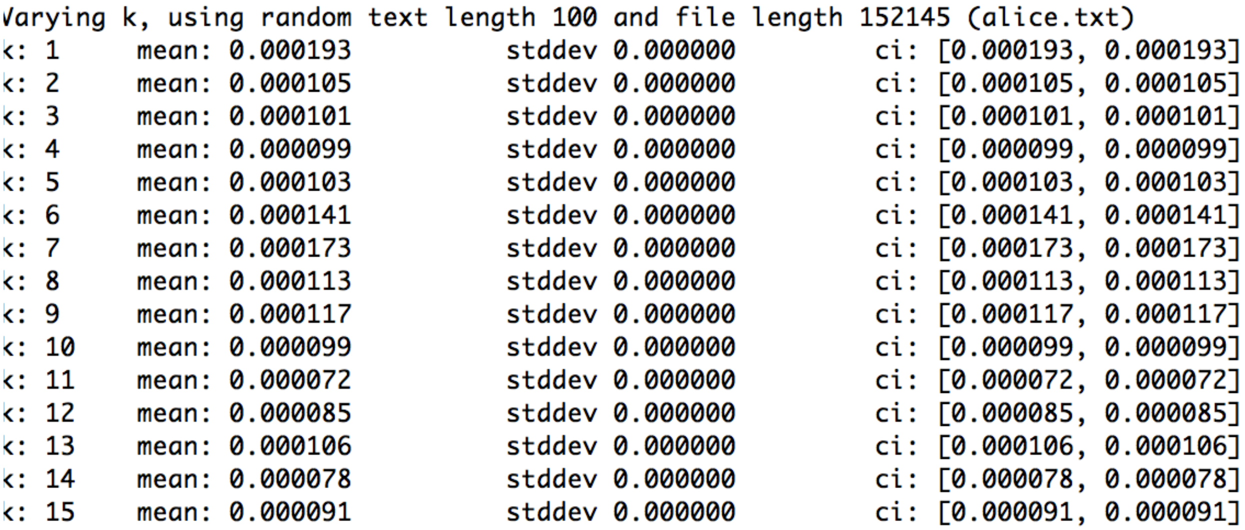
Data for MapGenerator:



```
Varying text length, using k 5 and file length 152145 (alice.txt)
text length: 20         mean: 0.000020          stddev: 0.000000        ci: [0.000020, 0.000020]
text length: 40         mean: 0.000047          stddev: 0.000000        ci: [0.000047, 0.000047]
text length: 60         mean: 0.000079          stddev: 0.000000        ci: [0.000079, 0.000079]
text length: 80         mean: 0.000098          stddev: 0.000000        ci: [0.000098, 0.000098]
text length: 100        mean: 0.000142          stddev: 0.000000        ci: [0.000142, 0.000142]
text length: 120        mean: 0.000152          stddev: 0.000000        ci: [0.000152, 0.000152]
text length: 140        mean: 0.000163          stddev: 0.000000        ci: [0.000163, 0.000163]
text length: 160        mean: 0.000157          stddev: 0.000000        ci: [0.000157, 0.000157]
text length: 180        mean: 0.000214          stddev: 0.000000        ci: [0.000214, 0.000214]
text length: 200        mean: 0.000237          stddev: 0.000000        ci: [0.000237, 0.000237]
text length: 220        mean: 0.000210          stddev: 0.000000        ci: [0.000210, 0.000210]
text length: 240        mean: 0.000233          stddev: 0.000000        ci: [0.000233, 0.000233]
text length: 260        mean: 0.000323          stddev: 0.000000        ci: [0.000323, 0.000323]
text length: 280        mean: 0.000368          stddev: 0.000000        ci: [0.000368, 0.000368]
text length: 300        mean: 0.000310          stddev: 0.000000        ci: [0.000310, 0.000310]
```

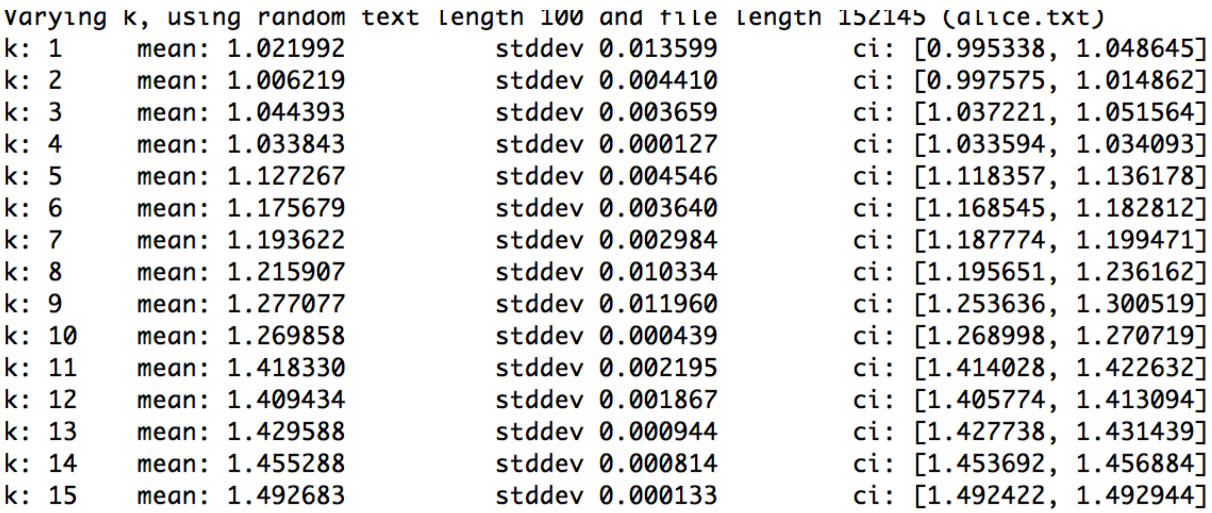
ii) The benchmark data in both MapGenerator and BruteGenerator support the hypothesis that big O is O(1), as the mean runtime changes little as K changes. For example, in MapGenerator, when k changes from 2 to 4, the mean runtime changes from 0.000105 second to 0.000099 second, which isn't a big change. Similarly, in BruteGenerator, when k changes from 1 to 4, the mean runtime changes from 1.021992 seconds to 1.033843 seconds, which isn't a big change either.

Map Generator Data:



```
Varying k, using random text length 100 and file length 152145 (alice.txt)
k: 1    mean: 0.000193          stddev 0.000000         ci: [0.000193, 0.000193]
k: 2    mean: 0.000105          stddev 0.000000         ci: [0.000105, 0.000105]
k: 3    mean: 0.000101          stddev 0.000000         ci: [0.000101, 0.000101]
k: 4    mean: 0.000099          stddev 0.000000         ci: [0.000099, 0.000099]
k: 5    mean: 0.000103          stddev 0.000000         ci: [0.000103, 0.000103]
k: 6    mean: 0.000141          stddev 0.000000         ci: [0.000141, 0.000141]
k: 7    mean: 0.000173          stddev 0.000000         ci: [0.000173, 0.000173]
k: 8    mean: 0.000113          stddev 0.000000         ci: [0.000113, 0.000113]
k: 9    mean: 0.000117          stddev 0.000000         ci: [0.000117, 0.000117]
k: 10   mean: 0.000099          stddev 0.000000         ci: [0.000099, 0.000099]
k: 11   mean: 0.000072          stddev 0.000000         ci: [0.000072, 0.000072]
k: 12   mean: 0.000085          stddev 0.000000         ci: [0.000085, 0.000085]
k: 13   mean: 0.000106          stddev 0.000000         ci: [0.000106, 0.000106]
k: 14   mean: 0.000078          stddev 0.000000         ci: [0.000078, 0.000078]
k: 15   mean: 0.000091          stddev 0.000000         ci: [0.000091, 0.000091]
```

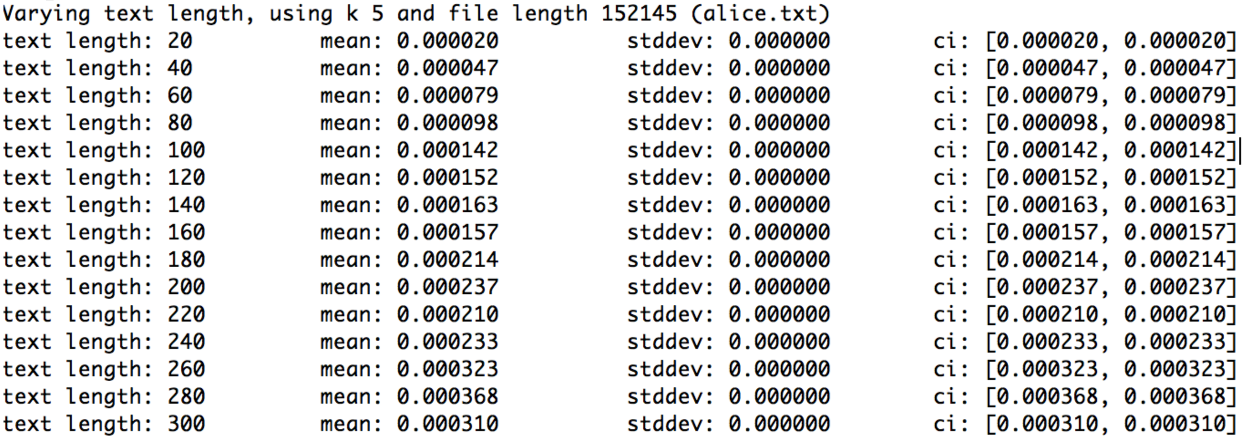
BruteGenerator Data:



```
Varying k, using random text length 100 and file length 152145 (alice.txt)
k: 1     mean: 1.021992          stddev 0.013599          ci: [0.995338, 1.048645]
k: 2     mean: 1.006219          stddev 0.004410          ci: [0.997575, 1.014862]
k: 3     mean: 1.044393          stddev 0.003659          ci: [1.037221, 1.051564]
k: 4     mean: 1.033843          stddev 0.000127          ci: [1.033594, 1.034093]
k: 5     mean: 1.127267          stddev 0.004546          ci: [1.118357, 1.136178]
k: 6     mean: 1.175679          stddev 0.003640          ci: [1.168545, 1.182812]
k: 7     mean: 1.193622          stddev 0.002984          ci: [1.187774, 1.199471]
k: 8     mean: 1.215907          stddev 0.010334          ci: [1.195651, 1.236162]
k: 9     mean: 1.277077          stddev 0.011960          ci: [1.253636, 1.300519]
k: 10    mean: 1.269858          stddev 0.000439          ci: [1.268998, 1.270719]
k: 11    mean: 1.418330          stddev 0.002195          ci: [1.414028, 1.422632]
k: 12    mean: 1.409434          stddev 0.001867          ci: [1.405774, 1.413094]
k: 13    mean: 1.429588          stddev 0.000944          ci: [1.427738, 1.431439]
k: 14    mean: 1.455288          stddev 0.000814          ci: [1.453692, 1.456884]
k: 15    mean: 1.492683          stddev 0.000133          ci: [1.492422, 1.492944]
```

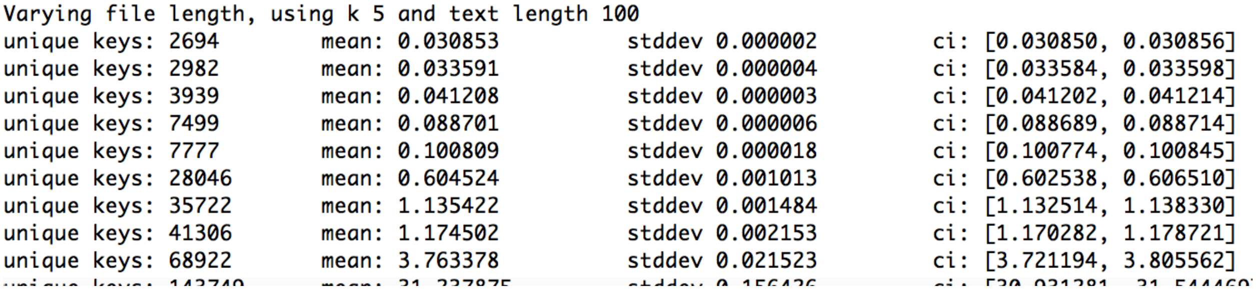
iii) The benchmark data in both MapGenerator and BruteGenerator support the hypothesis that big O is O(n), as the mean runtime approximately doubles as K doubles. For example, in MapGenerator, when the length of the random text doubles to 120 from 60, the mean runtime also approximately doubles to 0.000152 second from 0.000079 second. In BruteGenerator, when the length of the random text doubles to 40 from 20, the mean runtime also approximately doubles to 0.438883 second from 0.225145 second.

MapGenerator Data:



```
Varying text length, using k 5 and file length 152145 (alice.txt)
text length: 20          mean: 0.000020          stddev: 0.000000        ci: [0.000020, 0.000020]
text length: 40          mean: 0.000047          stddev: 0.000000        ci: [0.000047, 0.000047]
text length: 60          mean: 0.000079          stddev: 0.000000        ci: [0.000079, 0.000079]
text length: 80          mean: 0.000098          stddev: 0.000000        ci: [0.000098, 0.000098]
text length: 100         mean: 0.000142          stddev: 0.000000        ci: [0.000142, 0.000142]
text length: 120         mean: 0.000152          stddev: 0.000000        ci: [0.000152, 0.000152]
text length: 140         mean: 0.000163          stddev: 0.000000        ci: [0.000163, 0.000163]
text length: 160         mean: 0.000157          stddev: 0.000000        ci: [0.000157, 0.000157]
text length: 180         mean: 0.000214          stddev: 0.000000        ci: [0.000214, 0.000214]
text length: 200         mean: 0.000237          stddev: 0.000000        ci: [0.000237, 0.000237]
text length: 220         mean: 0.000210          stddev: 0.000000        ci: [0.000210, 0.000210]
text length: 240         mean: 0.000233          stddev: 0.000000        ci: [0.000233, 0.000233]
text length: 260         mean: 0.000323          stddev: 0.000000        ci: [0.000323, 0.000323]
text length: 280         mean: 0.000368          stddev: 0.000000        ci: [0.000368, 0.000368]
text length: 300         mean: 0.000310          stddev: 0.000000        ci: [0.000310, 0.000310]
```

BruteGenerator Data:



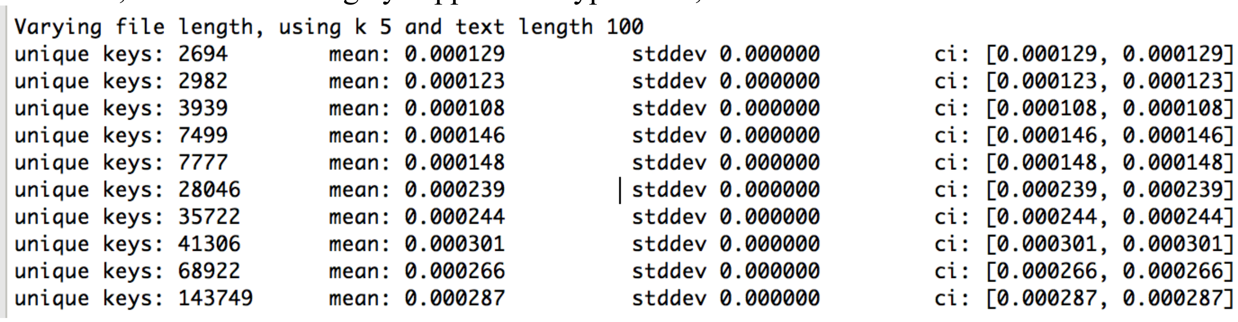
```
Varying file length, using k 5 and text length 100
unique keys: 2694        mean: 0.030853          stddev 0.000002         ci: [0.030850, 0.030856]
unique keys: 2982        mean: 0.033591          stddev 0.000004         ci: [0.033584, 0.033598]
unique keys: 3939        mean: 0.041208          stddev 0.000003         ci: [0.041202, 0.041214]
unique keys: 7499        mean: 0.088701          stddev 0.000006         ci: [0.088689, 0.088714]
unique keys: 7777        mean: 0.100809          stddev 0.000018         ci: [0.100774, 0.100845]
unique keys: 28046       mean: 0.604524          stddev 0.001013         ci: [0.602538, 0.606510]
unique keys: 35722       mean: 1.135422          stddev 0.001484         ci: [1.132514, 1.138330]
unique keys: 41306       mean: 1.174502          stddev 0.002153         ci: [1.170282, 1.178721]
unique keys: 68922       mean: 3.763378          stddev 0.021523         ci: [3.721194, 3.805562]
unique keys: 143749      mean: 31.237875         stddev 0.156426         ci: [30.931281, 31.544469]
```

i) The big O notation for Hashmap with the default hashCode function is O(n) , with n being the number of keys in the map, because every item in the map would refer to the same hashCode. The program has to go through every key in the hashmap until it finds what it's looking for.

ii) For an efficient hashCode, the big O notation is O(1) because the program could get the key and the values right away.

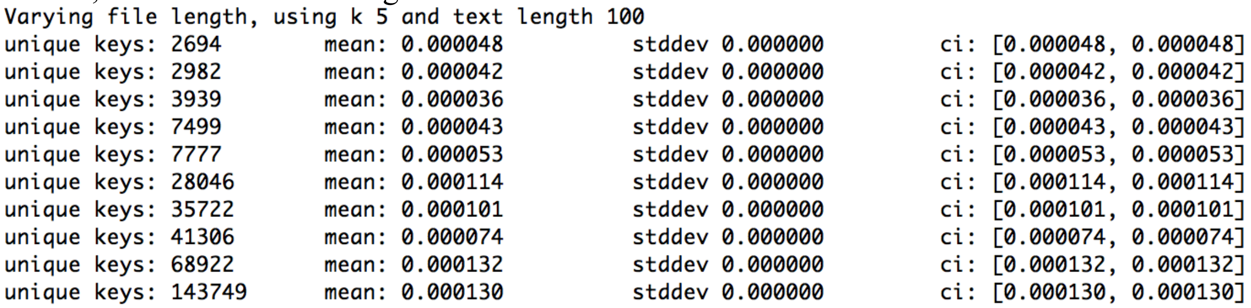
iii) The big O notation for Treemap is O(log n) with n being the number of unique keys in the map because a treemap uses binary search trees, for which the most number of searches needed would be the height of the tree, which is log(n).

4. i) The benchmark data in MapGenerator generally supports the hypothesis that big O notation is O(n), as the mean runtime changes proportionately as the number of unique keys in the map changes. For example, when the number of unique keys changes from 2694 to 2982, the mean runtime changes from 0.000129 second to 0.000123 second, with the changes being proportionate to each other. However, there are also changes that are disproportionate. For example, as the number of unique keys changes from 41306 to 68922, the mean runtime changes from 0.000301 second to 0.000266 second . This could be because that the program doesn't use a Linked List here and actually uses a binary search tree instead, in which case, the big O notation would be O(log n), with n being the number of unique keys in the map. That said, the data does largely support the hypothesis, as shown below:



```
Varying file length, using k 5 and text length 100
unique keys: 2694        mean: 0.000129        stddev 0.000000        ci: [0.000129, 0.000129]
unique keys: 2982        mean: 0.000123        stddev 0.000000        ci: [0.000123, 0.000123]
unique keys: 3939        mean: 0.000108        stddev 0.000000        ci: [0.000108, 0.000108]
unique keys: 7499        mean: 0.000146        stddev 0.000000        ci: [0.000146, 0.000146]
unique keys: 7777        mean: 0.000148        stddev 0.000000        ci: [0.000148, 0.000148]
unique keys: 28046       mean: 0.000239        stddev 0.000000        ci: [0.000239, 0.000239]
unique keys: 35722       mean: 0.000244        stddev 0.000000        ci: [0.000244, 0.000244]
unique keys: 41306       mean: 0.000301        stddev 0.000000        ci: [0.000301, 0.000301]
unique keys: 68922       mean: 0.000266        stddev 0.000000        ci: [0.000266, 0.000266]
unique keys: 143749      mean: 0.000287        stddev 0.000000        ci: [0.000287, 0.000287]
```

ii) The benchmark data in MapGenerator support the hypothesis that big O is O(1), as the mean runtime changes little as the number of keys changes. For example, when the unique keys in the map change from 2694 to 3939, the mean runtime changes from 0.000048 second to 0.000036 second, which is a small change. The benchmark data is shown below.



```
Varying file length, using k 5 and text length 100
unique keys: 2694        mean: 0.000048        stddev 0.000000        ci: [0.000048, 0.000048]
unique keys: 2982        mean: 0.000042        stddev 0.000000        ci: [0.000042, 0.000042]
unique keys: 3939        mean: 0.000036        stddev 0.000000        ci: [0.000036, 0.000036]
unique keys: 7499        mean: 0.000043        stddev 0.000000        ci: [0.000043, 0.000043]
unique keys: 7777        mean: 0.000053        stddev 0.000000        ci: [0.000053, 0.000053]
unique keys: 28046       mean: 0.000114        stddev 0.000000        ci: [0.000114, 0.000114]
unique keys: 35722       mean: 0.000101        stddev 0.000000        ci: [0.000101, 0.000101]
unique keys: 41306       mean: 0.000074        stddev 0.000000        ci: [0.000074, 0.000074]
unique keys: 68922       mean: 0.000132        stddev 0.000000        ci: [0.000132, 0.000132]
unique keys: 143749      mean: 0.000130        stddev 0.000000        ci: [0.000130, 0.000130]
```

iii) The benchmark data in MapGenerator support the hypothesis that big O is O (log n) with n being the number of unique keys in the map. For example, when the number of unique keys changes from 2982 to 3939, the mean runtimes changes from 0.000073 second to 0.000091 second, which validates the hypothesis. The benchmark data is shown below.

```
Varying file length, using k 5 and text length 100
unique keys: 2694        mean: 0.000086           stddev 0.000000         ci: [0.000086, 0.000086]
unique keys: 2982        mean: 0.000073           stddev 0.000000         ci: [0.000073, 0.000073]
unique keys: 3939        mean: 0.000091           stddev 0.000000         ci: [0.000091, 0.000091]
unique keys: 7499        mean: 0.000106           stddev 0.000000         ci: [0.000106, 0.000106]
unique keys: 7777        mean: 0.000142           stddev 0.000000         ci: [0.000142, 0.000142]
unique keys: 28046       mean: 0.000202           stddev 0.000000         ci: [0.000202, 0.000202]
unique keys: 35722       mean: 0.000233           stddev 0.000000         ci: [0.000233, 0.000233]
unique keys: 41306       mean: 0.000260           stddev 0.000000         ci: [0.000260, 0.000260]
unique keys: 68922       mean: 0.000263           stddev 0.000000         ci: [0.000263, 0.000263]
unique keys: 143749      mean: 0.000237           stddev 0.000000         ci: [0.000237, 0.000237]
```