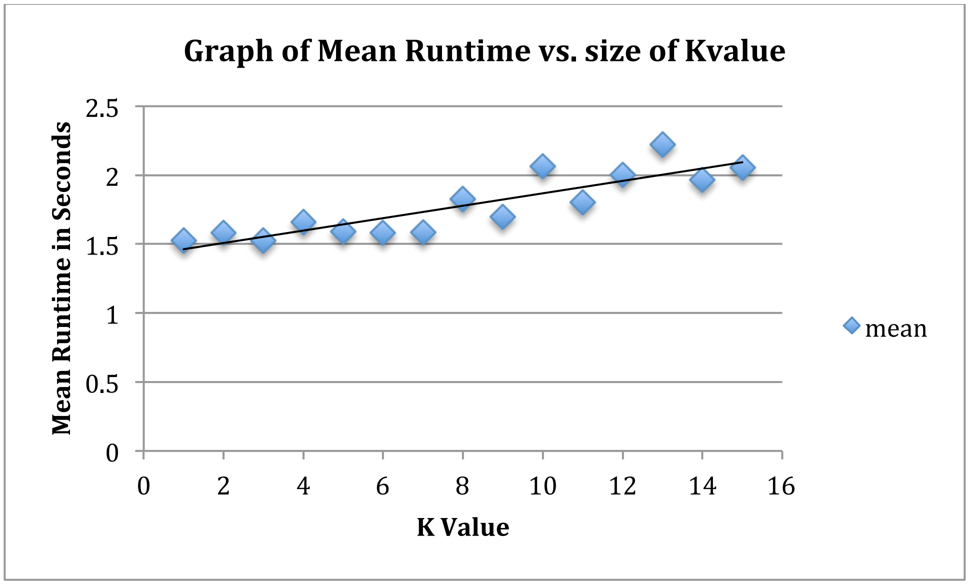
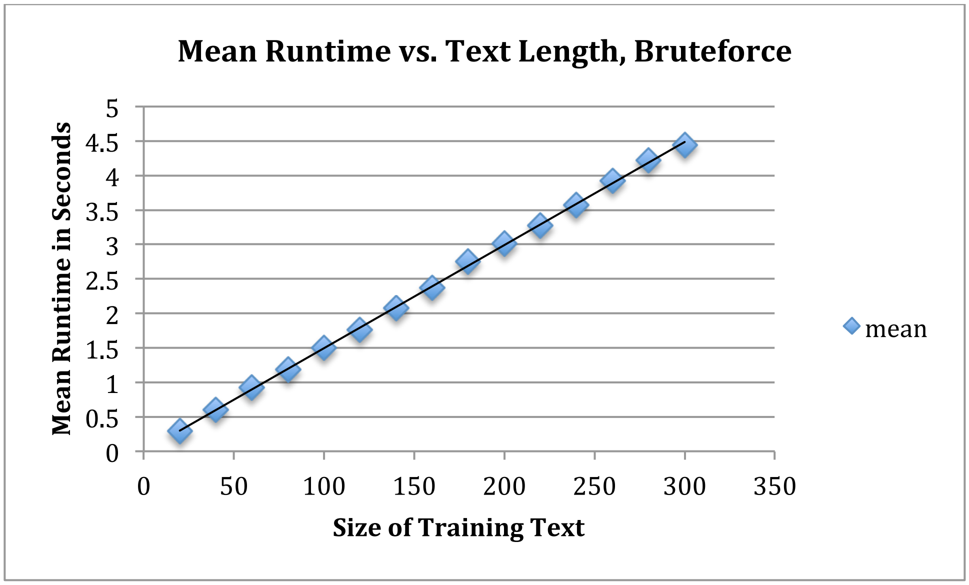
Rich von der Schmidt
CS201-Spring-Azhar
TA: Efe Aras
Assignment: Markov
Analysis
3-8-16

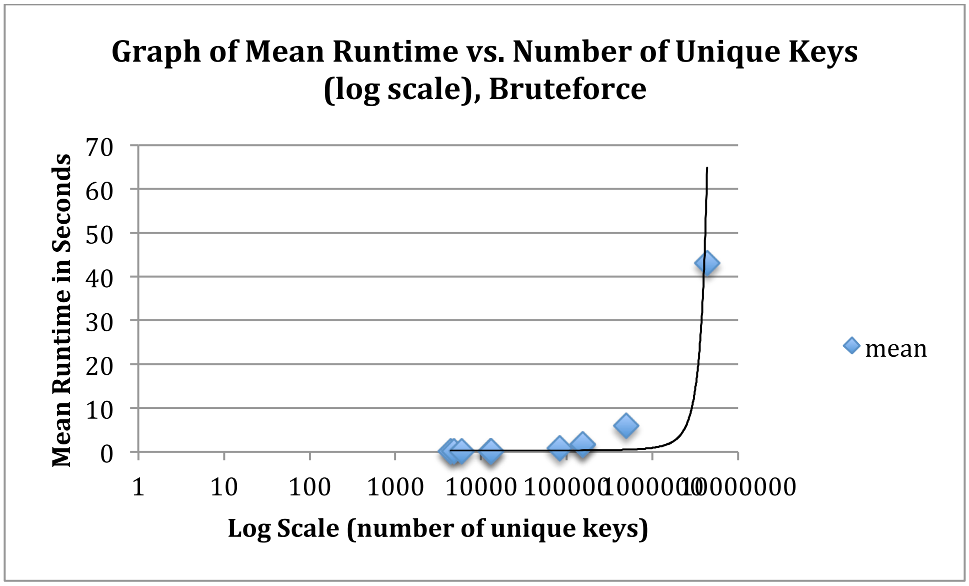
- Question 1:
    - Brute Force
        - Length of Training text:
            - O(n)
        - K-value/ Length of word
            - O(k)
        - Length of Random text to be generated
            - O(n)^l
            - Must loop through the entire text each time we are trying to generate one bit of output for l because we generate the arraylist on the fly
    - Map
        - Length of Training text
            - O(n)
        - K-value/ Length of word
            - O(1): lookup is a constant time operation, and putting them in the Hashmap is not greatly affected by the size of the string we are inputting
        - Length of Random text to be generated
            - O(l)
    - Explanation:
        - In words. The bruteforce algorithm loops through the text each time to generate the arraylist of ngrams that follow a given seed. As a result, it takes damn near forever to run. As the length of the training text increases, so should the run time. As the length of the output text increases, again, so should the run time. When we increase the size of the random output, we have to loop through the text again each time, so these are piggybacked. As the length of the k value increases, in bruteforce, we also expect to see marginal increases in runtime because of the cost of adding these bigger values.
        - For the mapgenerator, we only have to loop through the text once to generate the maps. The runtime should increase with the size of the random text to be output because this is a linear relationship. The size of the k value is not as crucial here because with a good hashcode (with few collisions) lookup and adding are roughly constant time operations. See graphs.

- Question 2:
  - Benchmark on Brute Generator

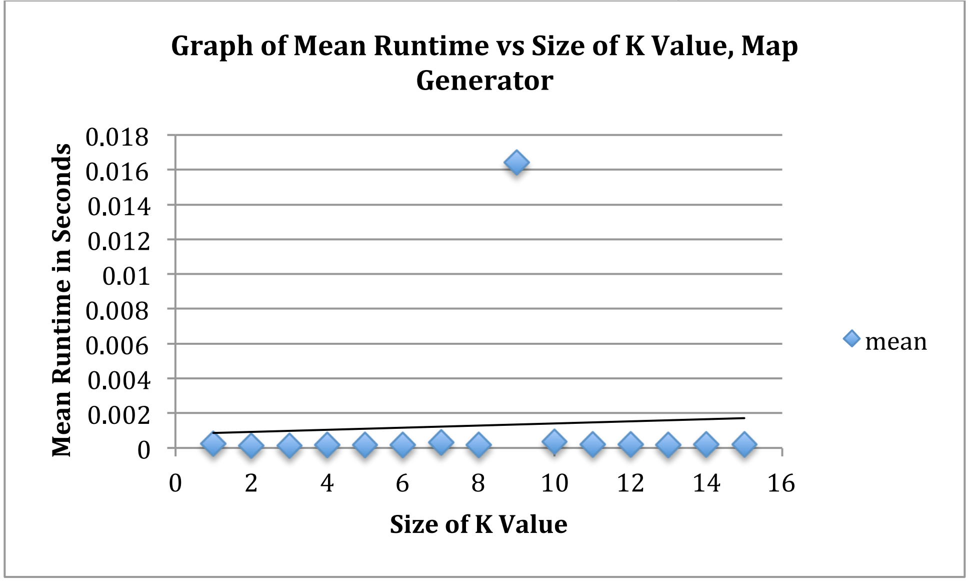




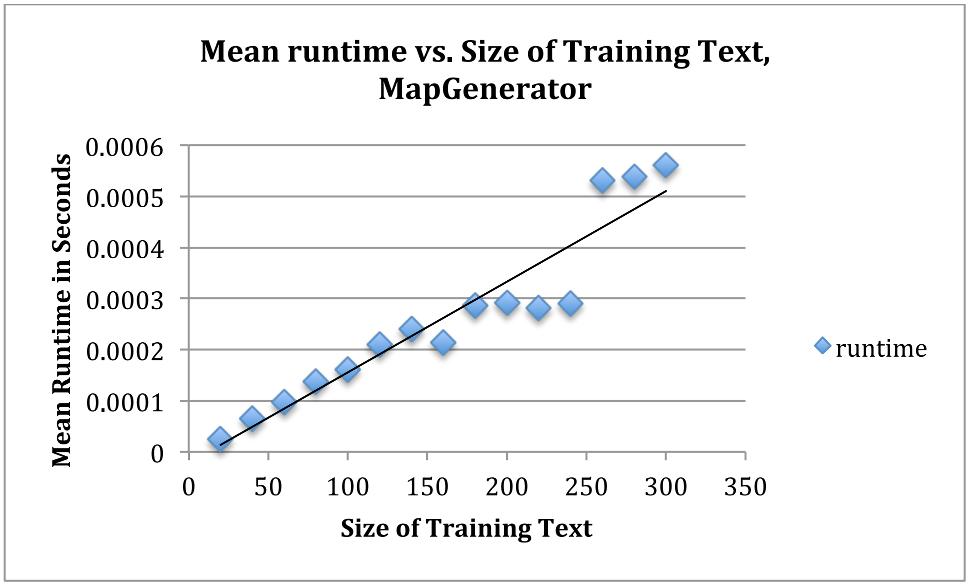
- Question 2:
  - Benchmark on Brute Generator

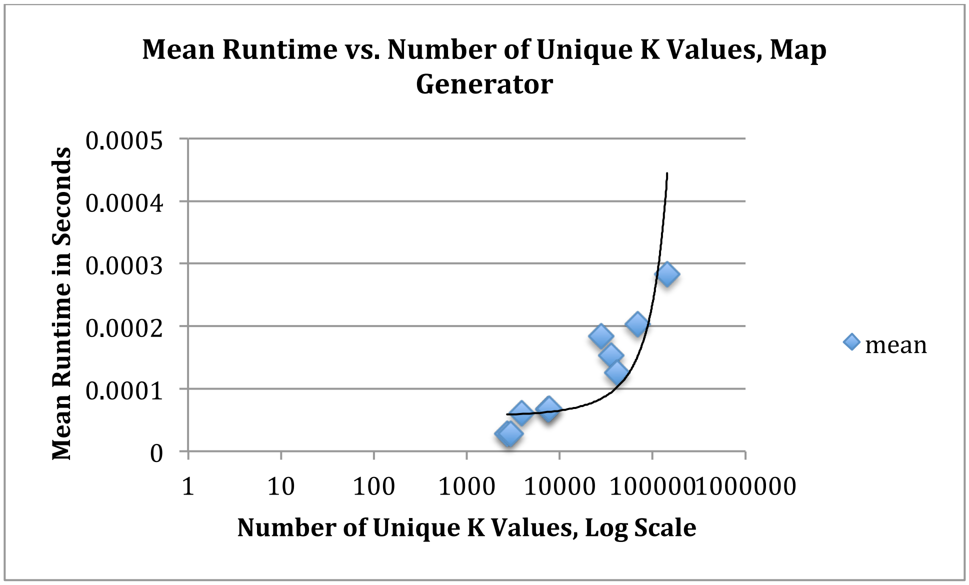


- Question 2: Benchmark on MapGenerator with a good HashCode



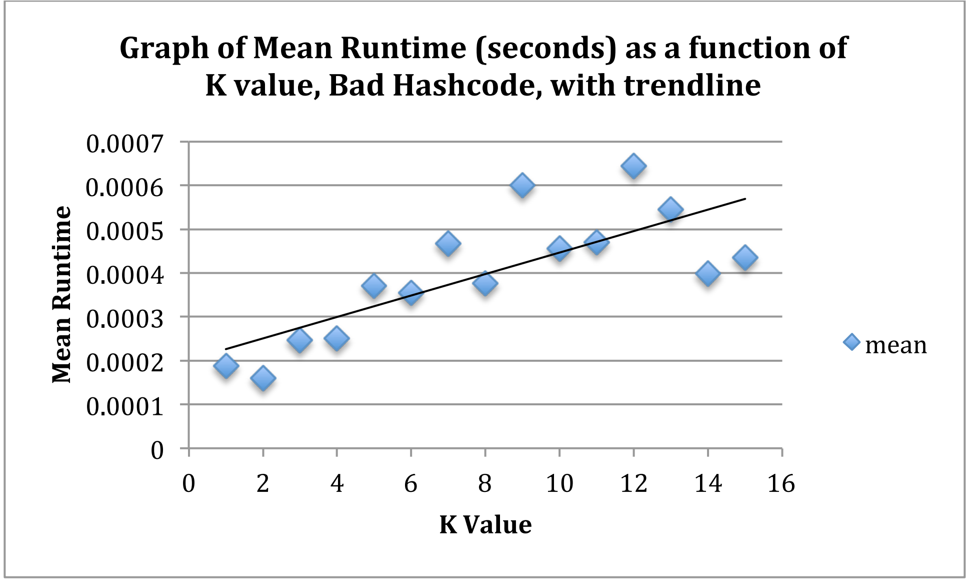
- Question 2: Benchmark on MapGenerator with a good HashCode



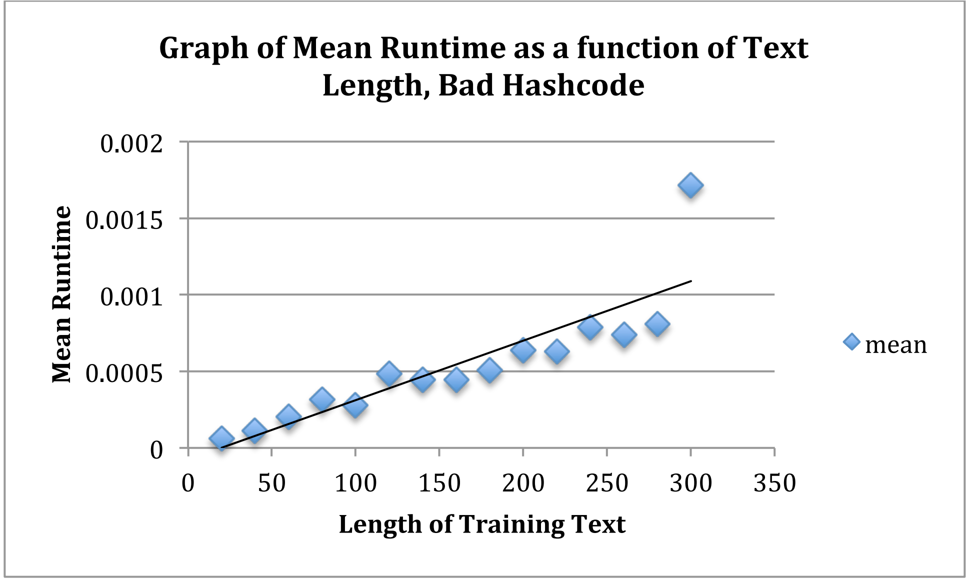


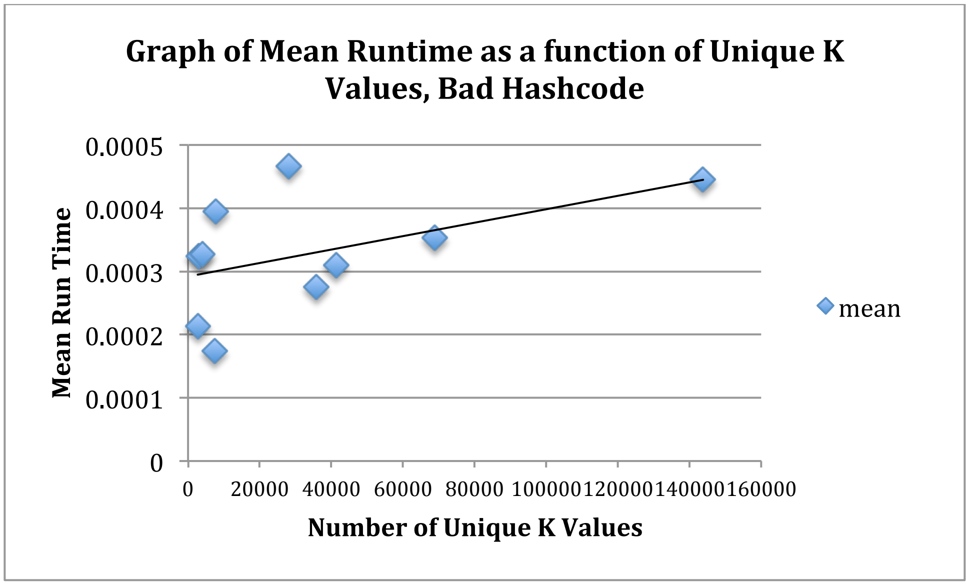
- Question 3:
    - How the number of keys affects…
        - Hashmap with a good hash function
            - The number of keys should not affect the performance of the hashmap with a good hash function, if the entries are spaced out and we avoid collisions, adding and lookup are constant time operations.
        - Hashmap with a bad hash function
            - With a bad hash function, (returning the constant) all the values are stored in the overflow array list for the one bucket that catches them all. Now, lookup and adding are no longer constant time operations. In the worst case this resembles O(n) if all keys are in the same bucket for adding, same for lookup since we don't know the exact index of the key we are looking for. (That is the point of hash functions after all)
        - Treemap
            - The order of growth for the number of keys for TreeMap should be O(log n) since lookup requires that we traverse the tree (for generating random text) and adding also requires that we find the right spot to add, which also costs O(log n)

- Question 4:
    - Use benchmark to examine data (map with good hashcode graphs are above)
    - Below: Hashmap with bad hash function

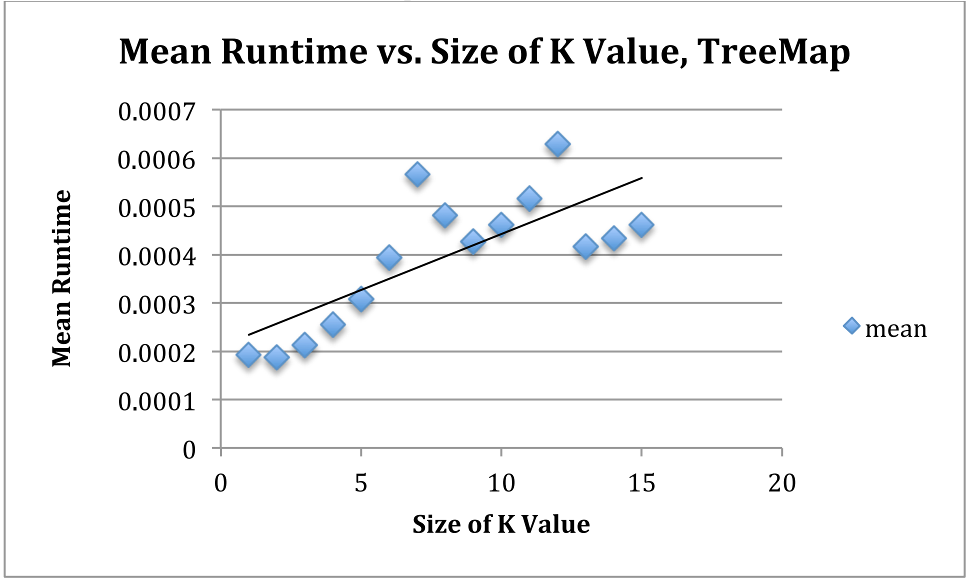


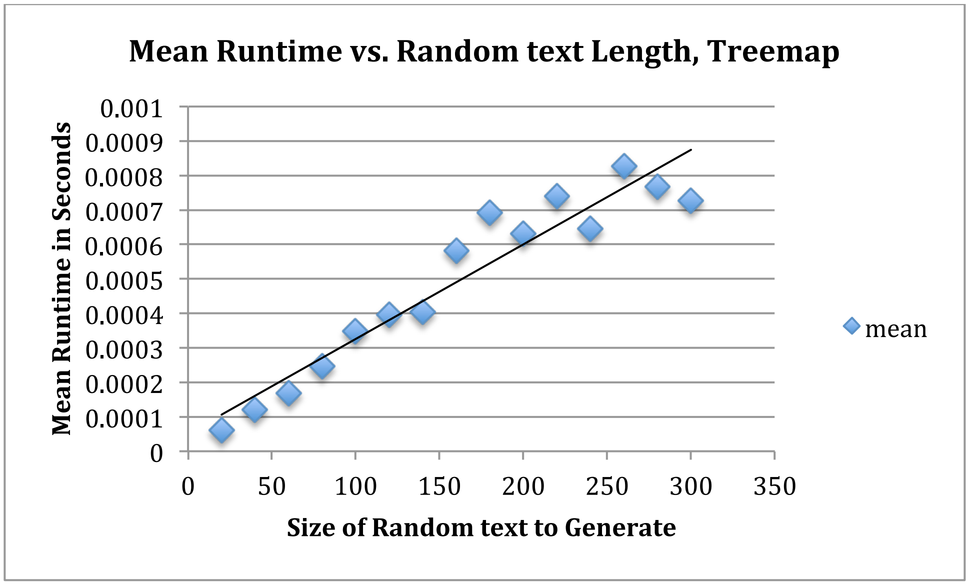
- Question 4:
  - Below: Hashmap with bad hash function



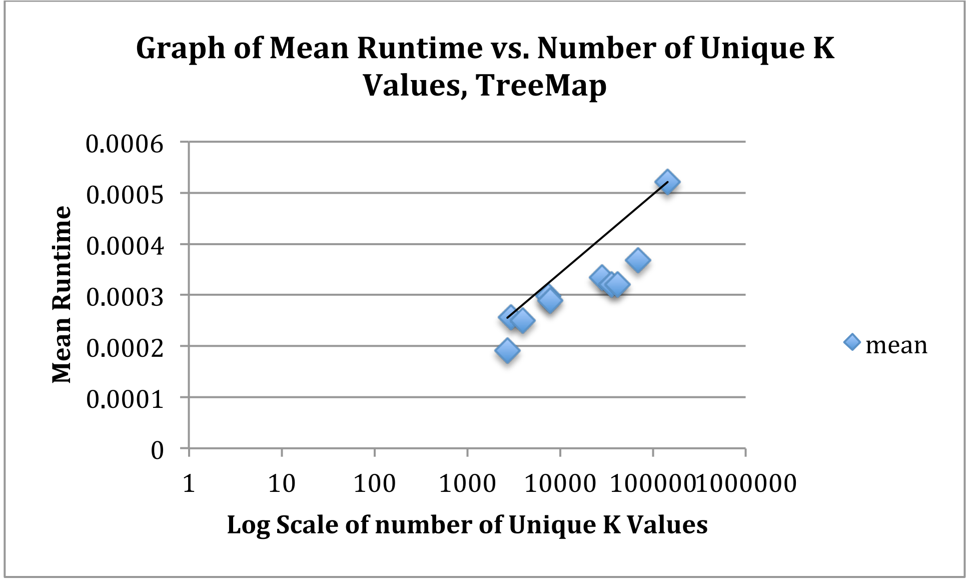


- Question 4:
  - Below: TreeMap





- Question 4:
  - Below: TreeMap


Graph of Mean Runtime vs. Number of Unique K Values, TreeMap
Mean Runtime
Log Scale of number of Unique K Values
◆ mean