Richard von der Schmidt
4-11-16
CS-201: Spring, Azhar
TA: Efe Aras
Autocomplete Analysis:

- Question 1: Number of Compares for each Algorithm:
  - Binary Search:
    - Finding the first term: O(log N)
    - Finding the last term: O(log N)
    - (Binary search cuts down on half the items that need to be compared)
    - If we have M matching terms, then finding them in the priority queue costs O(log M), and we must do this for each of the M matching terms, so the total cost of traversing the PQ is O(M log M)
    - O(2 log N + M log M) ➔ O(log N + M log M)
    - The value of k only matters for culling the list we get after traversing the priority queue and is of insignificant cost compared to the above

- Question 2: Runtime of top K matches varying with value of k
  - For each of the three methods, this is only a very small part of the runtime for these algorithms. The major costs are borne in the searching for matches; k only enters the equation when the matches are culled into an output list of size k at the very end
  - So for each one, increasing k will increase the run time of the algorithm in linear fashion, but this effect will be negligible compared to the cost of finding the number of matching terms
  - (tl;dr: no, it shouldn't affect it in any way that we humans can notice)

- Question 3: Is binary search always guaranteed to outperform brute for topKmatches and topMatch?
  - No. When the prefixes are very short, the number of matches is much greater, so binary search must look through all the matches. Brute on the other hand, does not need to examine all the matching terms since it traverses the entire list of terms

- Question 4: Increasing the size of the source, and increasing the size of the prefix argument, effects on topKMatches and topMatch (in theory)
  - For all of these implementations, as the prefix size gets larger, the number of matches should get smaller. Conversely, as the prefix size gets smaller, the number of matches is greater.

- Brute
  - TopMatch: O(N) - (no matter the size of the prefix argument, each of these operations must scan the entire input source)
  - TopKMatches: O(N + M log M)- the M log M comes from traversing the priority queue
- Binary Search
  - TopMatch:
    - O(log n + m)
  - TopKMatches:
    - O(log n + m log m)
- Trie
  - TopMatch and TopKMatches:
    - The size of the input source should not affect the runtime of this algorithm
    - As the size of the prefix increases, the runtime of the algorithm should decrease
    - (They are inversely proportional- not sure how to show this in big O, but if I triple the prefix time, the algorithm should take only 1/3 of the time in principle)