1. Run the program BenchmarkForAutocomplete and copy/paste the results into the file you submit. You'll need to run three times, once for each of the files in the Benchmark program: threeletterwords.txt, fourletterwords.txt, and alexa.txt. On ola's computer the first few lines are what's shown below for "data/threeletterwords.txt". The unlabeled "search" is for an empty string "" which matches every string stored. These numbers are for a file of every three letter word "aaa, "aab", … "zzy", "zzz", not actual words, but 3-character strings.

init time: 0.006699 for BruteAutocomplete

init time: 0.004799 for BinarySearchAutocomplete

init time: 0.07067 for HashListAutocomplete

search size #match BruteAutoc BinarySear HashListAu

17576 50 0.00238732 0.00219437 0.00019249

17576 50 0.00056931 0.00136807 0.00000449

a 676 50 0.00044899 0.00015267 0.00000443

a 676 50 0.00042797 0.00013736 0.00000575

b 676 50 0.00051954 0.00015502 0.00000640

1. Run the program again for alexa.txt with  #matches = 10000, paste the results, and then explain to what extent the # matches affects the runtime. The # matches, matchSize, is specified in the method runAM (for run all matches)
2. Explain why the last for loop in BruteAutocomplete.topMatches uses a LinkedList (and not an ArrayList) AND why the PriorityQueue uses Comparator.comparing(Term::getWeight) to get the top k heaviest matches.

* Why last for loop in BruteAutocomplete.topMatches uses LinkedList
  + Why using ArrayList wouldn’t make sense
* Explain how Comparator.comapring(Term::getWeight) works for PriorityQueue
* What’s the purpose of getting the top k heaviest matches in terms of the problem context

1. Explain why HashListAutocomplete uses more memory than the other Autocomplete implementations. Be brief.

* Why hashlist takes up a lot of space (innerworkings in the back)
  + Maps taking up memory with keys and values explanation