P5 Analysis

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.

***result for threeletterwords.txt***

init time: 0.01630 for BruteAutocomplete

init time: 0.01007 for BinarySearchAutocomplete

init time: 0.1738 for HashListAutocomplete

search size #match BruteAutoc BinarySear HashListAu

17576 50 0.00291357 0.01192530 0.00006622

17576 50 0.00186159 0.00324465 0.00000774

a 676 50 0.00098776 0.00026731 0.00000613

a 676 50 0.00082538 0.00025807 0.00000563

b 676 50 0.00144241 0.00030145 0.00001544

c 676 50 0.00095708 0.00039470 0.00001375

g 676 50 0.00079572 0.00024062 0.00000697

ga 26 50 0.00061078 0.00007080 0.00000595

go 26 50 0.00068306 0.00008217 0.00000512

gu 26 50 0.00075481 0.00007417 0.00000564

x 676 50 0.00072020 0.00025078 0.00000557

y 676 50 0.00100431 0.00056167 0.00000648

z 676 50 0.00093860 0.00040899 0.00000633

aa 26 50 0.00103194 0.00013393 0.00000834

az 26 50 0.00102061 0.00024649 0.00002218

za 26 50 0.00031356 0.00006762 0.00000708

zz 26 50 0.00057337 0.00004992 0.00000649

zqzqwwx 0 50 0.00019095 0.00003061 0.00000240

size in bytes=246064 for BruteAutocomplete

size in bytes=246064 for BinarySearchAutocomplete

size in bytes=1092468 for HashListAutocomplete

***result for fourletterwords.txt***

init time: 0.07183 for BruteAutocomplete

init time: 0.02938 for BinarySearchAutocomplete

init time: 1.162 for HashListAutocomplete

search size #match BruteAutoc BinarySear HashListAu

456976 50 0.01064342 0.03244870 0.00007366

456976 50 0.00431931 0.00447909 0.00000755

a 17576 50 0.01004850 0.00031225 0.00001936

a 17576 50 0.00514206 0.00040506 0.00000713

b 17576 50 0.00402912 0.00026328 0.00000539

c 17576 50 0.00389821 0.00025164 0.00000557

g 17576 50 0.00346129 0.00037173 0.00000717

ga 676 50 0.00363112 0.00009578 0.00000571

go 676 50 0.00407085 0.00008119 0.00000579

gu 676 50 0.00368249 0.00008304 0.00000536

x 17576 50 0.00349981 0.00027206 0.00000628

y 17576 50 0.00417669 0.00026731 0.00000614

z 17576 50 0.00363190 0.00024278 0.00000697

aa 676 50 0.00368782 0.00007040 0.00000639

az 676 50 0.00355222 0.00007189 0.00000622

za 676 50 0.00389759 0.00012546 0.00002197

zz 676 50 0.00355957 0.00006675 0.00000594

zqzqwwx 0 50 0.00504773 0.00012692 0.00000666

size in bytes=7311616 for BruteAutocomplete

size in bytes=7311616 for BinarySearchAutocomplete

size in bytes=40322100 for HashListAutocomplete

***result for alexa.txt***

init time: 0.3503 for BruteAutocomplete

init time: 1.528 for BinarySearchAutocomplete

init time: 7.623 for HashListAutocomplete

search size #match BruteAutoc BinarySear HashListAu

1000000 50 0.02247188 0.10372771 0.00008463

1000000 50 0.01099393 0.08933350 0.00001183

a 69464 50 0.01097246 0.00750020 0.00005277

a 69464 50 0.01342277 0.00488598 0.00000868

b 56037 50 0.01087933 0.00383054 0.00000725

c 65842 50 0.01413490 0.00621941 0.00002352

g 37792 50 0.01315630 0.00292824 0.00000716

ga 6664 50 0.01053062 0.00072102 0.00000696

go 6953 50 0.01037153 0.00072344 0.00000733

gu 2782 50 0.01002810 0.00036084 0.00000660

x 6717 50 0.00972443 0.00060508 0.00000646

y 16765 50 0.00963091 0.00128811 0.00000822

z 8780 50 0.00979453 0.00073662 0.00000602

aa 718 50 0.01096038 0.00016060 0.00000684

az 889 50 0.01039988 0.00017951 0.00000633

za 1718 50 0.00959458 0.00026508 0.00000681

zz 162 50 0.01044079 0.00012322 0.00000607

zqzqwwx 0 50 0.00974707 0.00008774 0.00000302

size in bytes=38204230 for BruteAutocomplete

size in bytes=38204230 for BinarySearchAutocomplete

size in bytes=475893648 for HashListAutocomplete

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)

***result***

init time: 0.6254 for BruteAutocomplete

init time: 1.460 for BinarySearchAutocomplete

init time: 8.457 for HashListAutocomplete

search size #match BruteAutoc BinarySear HashListAu

1000000 10000 0.03248335 0.11698774 0.00008760

1000000 10000 0.01692705 0.14737910 0.00001548

a 69464 10000 0.01764203 0.02088367 0.00002777

a 69464 10000 0.01697979 0.01888484 0.00000963

b 56037 10000 0.01588361 0.01654637 0.00001063

c 65842 10000 0.01856439 0.02126216 0.00001048

g 37792 10000 0.01638702 0.01368583 0.00001048

ga 6664 10000 0.01420811 0.00355928 0.00000748

go 6953 10000 0.01578972 0.00338235 0.00000700

gu 2782 10000 0.01247519 0.00136572 0.00000897

x 6717 10000 0.01608633 0.00362610 0.00000780

y 16765 10000 0.01704877 0.00869600 0.00001033

z 8780 10000 0.01461898 0.00457405 0.00000782

aa 718 10000 0.01094509 0.00033068 0.00000688

az 889 10000 0.01036045 0.00042253 0.00000647

za 1718 10000 0.01053165 0.00078203 0.00000717

zz 162 10000 0.00954420 0.00009014 0.00000586

zqzqwwx 0 10000 0.00941757 0.00008926 0.00000355

size in bytes=38204230 for BruteAutocomplete

size in bytes=38204230 for BinarySearchAutocomplete

size in bytes=475893648 for HashListAutocomplete

***explanation***

The number of matches has some impacts on initialization time, but does not have large impact on overall runtime. The initialization time for BruteAutocomplete increases around 0.3 s, BinarySearchAutocomplete, ~ 0.1 s, and HashListAutocomplete, ~ 1 s.

The runtime differences for BruteAutocomplete are all less than 0.01 s, those for BinarySearchAutocomplete are only slightly greater than 0.01 s, which is still not a large impact, and those for HashListAutocomplete are smaller than 0. 00001 s.

1. 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.

The last for loop in BruteAutocomplete.topMatches uses a LinkedList because LinkedList allows an element to be added to the front of list with O(1) efficiency. On the other hand, adding an element to the front of an ArrayList will require every element currently in the list to be shifted 1 index behind, which is O(N) efficiency. In BruteAutocomplete.topMatches, we want to add an element of greater weight to the front of the list, so a LinkedList will be the most efficient.

The PriorityQueue uses Comparator.comparing(Term::getWeight) to get the top k heaviest matches because Term::getWeight compares the weight of each term, so when calling .remove in pq, the term that weighs the least will be removed first and the heaviest k terms will remain in pq, ensuring that by the end of the loop what’s left in pq are the top matches.

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

The initialize method of HashListAutocomplete stores not only the terms, but also all possible prefixes (up to MAX\_PREFIX number) for every term; thus, initialization takes up a lot of memory. On the other hand, the initialize method of BruteAutocomplete and BinarySearchAutocomplete only stores the terms. That’s why they use a lot less memory.