Angela Reyzelman

1.)

Threeletterwords.txt:

init time: 0.008863 for BruteAutocomplete
init time: 0.007074 for BinarySearchAutocomplete
init time: 0.1800 for HashListAutocomplete

search	size	#match	BruteAutoc	BinarySear	HashListAu
	17576	50	0.00601585	0.01146814	0.00010623
	17576	50	0.00127794	0.00608937	0.00000901
a	676	50	0.00207074	0.00038318	0.00001019
a	676	50	0.00091431	0.00036378	0.00000899
b	676	50	0.00374713	0.00036519	0.00000922
c	676	50	0.00099976	0.00034012	0.00001020
g	676	50	0.00293953	0.00032205	0.00000831
ga	26	50	0.00074180	0.00007546	0.00000684
go	26	50	0.00352066	0.00013413	0.00000885
gu	26	50	0.00072571	0.00007871	0.00000766
x	676	50	0.00086220	0.00043843	0.00000976
y	676	50	0.00082389	0.00040582	0.00001040
z	676	50	0.00123505	0.00035219	0.00001151
aa	26	50	0.00103808	0.00010259	0.00000956
az	26	50	0.00070484	0.00007872	0.00000979
za	26	50	0.00074889	0.00007121	0.00001150
zz	26	50	0.00075069	0.00007770	0.00000935
zqzqwwx	0	50	0.00123415	0.00009761	0.00001812

size in bytes=246064 for BruteAutocomplete
size in bytes=246064 for BinarySearchAutocomplete
size in bytes=1092468 for HashListAutocomplete

Fourletterwords.txt:

init time: 0.1556 for BruteAutocomplete
init time: 0.1384 for BinarySearchAutocomplete
init time: 1.780 for HashListAutocomplete

search	size	#match	BruteAutoc	BinarySear	HashListAu
	456976	50	0.02489710	0.04544217	0.00008311
	456976	50	0.01217313	0.01928710	0.00001545
a	17576	50	0.01449292	0.00088202	0.00001375
a	17576	50	0.02078109	0.00167946	0.00003160
b	17576	50	0.01237948	0.00098332	0.00001902
c	17576	50	0.00923475	0.00103178	0.00001587
g	17576	50	0.01022411	0.00156711	0.00001470

ga 676 50 0.00864368 0.00019509 0.00001072
go 676 50 0.01236808 0.00019470 0.00001533
gu 676 50 0.00906006 0.00014830 0.00090346
x 17576 50 0.00892343 0.00104062 0.00001299
y 17576 50 0.00735718 0.00044806 0.00001187
z 17576 50 0.00819325 0.00053275 0.00000939
aa 676 50 0.00844937 0.00007723 0.00000796
az 676 50 0.00989819 0.00008859 0.00000824
za 676 50 0.00907869 0.00013806 0.00003092
zz 676 50 0.00726408 0.00013262 0.00001144
zqzqwwx 0 50 0.00633792 0.00009173 0.00000295
size in bytes=7311616 for BruteAutocomplete
size in bytes=7311616 for BinarySearchAutocomplete
size in bytes=40322100 for HashListAutocomplete

Alexa.txt:

init time: 0.6887 for BruteAutocomplete
init time: 3.052 for BinarySearchAutocomplete
init time: 10.01 for HashListAutocomplete
search size #match BruteAutoc BinarySear HashListAu
1000000 50 0.05611237 0.15560741 0.00008183
1000000 50 0.02161265 0.19994939 0.00003905
a 69464 50 0.02336989 0.01058844 0.00001430
a 69464 50 0.01793743 0.00902410 0.00001935
b 56037 50 0.01798248 0.00724327 0.00001868
c 65842 50 0.01719683 0.00726141 0.00007218
g 37792 50 0.01766185 0.00339604 0.00019064
ga 6664 50 0.01887666 0.00089596 0.00001020
go 6953 50 0.01951440 0.00073315 0.00000891
gu 2782 50 0.01870469 0.00131444 0.00001510
x 6717 50 0.01866042 0.00129448 0.00001684
y 16765 50 0.01864058 0.00242418 0.00010428
z 8780 50 0.01811433 0.00194860 0.00001368
aa 718 50 0.01960558 0.00025752 0.00001450
az 889 50 0.01873837 0.00031855 0.00001234
za 1718 50 0.01575088 0.00047360 0.00001136
zz 162 50 0.01734885 0.00011820 0.00001095
zqzqwwx 0 50 0.01707456 0.00023570 0.00000392
size in bytes=38204230 for BruteAutocomplete
size in bytes=38204230 for BinarySearchAutocomplete
size in bytes=475893648 for HashListAutocomplete

2.)

Alexa.txt with MatchSize of 10000:

init time: 0.8304 for BruteAutocomplete
init time: 2.841 for BinarySearchAutocomplete
init time: 10.03 for HashListAutocomplete

search	size	#match	BruteAutoc	BinarySear	HashListAu
	1000000	10000	0.05504813	0.23924597	0.00017704
	1000000	10000	0.02936991	0.21191790	0.00001259
a	69464	10000	0.03516735	0.04030473	0.00002206
a	69464	10000	0.02717172	0.03205163	0.00001572
b	56037	10000	0.02634654	0.03075496	0.00001337
c	65842	10000	0.02841933	0.03566129	0.00001312
g	37792	10000	0.03137370	0.02903809	0.00001244
ga	6664	10000	0.02728662	0.00608567	0.00001897
go	6953	10000	0.02625126	0.00615137	0.00001194
gu	2782	10000	0.02173926	0.00225995	0.00001440
x	6717	10000	0.02401600	0.00530286	0.00001672
y	16765	10000	0.02738330	0.01537466	0.00001350
z	8780	10000	0.02738352	0.00786187	0.00001431
aa	718	10000	0.01981619	0.00055248	0.00001399
az	889	10000	0.01632400	0.00042367	0.00000870
za	1718	10000	0.01985069	0.00121951	0.00001784
zz	162	10000	0.01869774	0.00015315	0.00000983
zqzqwwx	0	10000	0.02013033	0.00014926	0.00000423

size in bytes=38204230 for BruteAutocomplete
size in bytes=38204230 for BinarySearchAutocomplete
size in bytes=475893648 for HashListAutocomplete

When the number of matches increases from 50 to 10000, the runtime for BruteAutocomplete increases from 0.6887 to 0.8304, which is a very small difference in runtimes. For BinarySearchAutocomplete, the runtime decreases from 3.052 to 2.841, which is also an inconsequentially small difference in runtimes. Finally, for HashListAutocomplete, the runtime barely increases from 10.01 to 10.03. Therefore, it can be concluded that the number of matches, matchSize, does not affect the runtime of BruteAutocomplete, BinarySearchAutocomplete, and HashListAutocomplete.

3.)

BruteAutocomplete.topMatches uses a LinkedList instead of an ArrayList in order to more efficiently add terms to the front of the list, which is what we need to do in order to return a list with the heaviest elements first, not the lighter ones. LinkedList allows terms to be added to the front of the list in a constant O(1) runtime while ArrayList has to shift all of the existing terms in the list to add a term to the front of the list, which has an O(N) runtime.

Moreover, PriorityQueue uses Comparator.comparing(Term::getWeight) to use getWeight to get the top k heaviest elements that have passed through the PriorityQueue at any time t, and ends up returning the k heaviest elements In the array of terms.

4.)
HashListAutocomplete uses more memory than the other Autocomplete implementations because in the Autocomplete implementations such as BruteAutocomplete and BinarySearchAutoComplete, we use a PriorityQueue and this PriorityQueue has a maximum of k elements and discards the rest of the terms that we don't want, so PriorityQueue will always have k terms while HashList creates a Map of keys that contain prefixes and values that contain words with the prefixes so in comparison, the Map will have a great amount of duplicate terms, which will take up a significantly greater amount of memory than a PriorityQueue.

5.)
Comment on some aspect of the article relating to privacy and DNA

I found this article kind of concerning. Although I believe that it is very useful that one DNA sample, or even lack thereof, can be used to trace relatives with chunks of matching DNA that can turn up a third cousin or closer, I believe that this is only useful in terms of criminal cases. Otherwise, if there was to be a breach of this data, I believe that it could be very harmful. The statement that the analysis can quickly lead to demographic information such as address and exact location up to 100 miles is a major invasion of privacy and data in my opinion, one that could be used detrimentally if fallen into the wrong hands.