Analysis

Question 1:

Threeletters.txt

```
init time: 0.003647 for BruteAutocomplete
init time: 0.004130 for BinarySearchAutocomplete
init time: 0.07435 for HashListAutocomplete
search size #match BruteAutoc BinarySear HashListAu
        17576  50 0.00330252 0.00298379 0.00006914
        17576  50 0.00109189 0.00136815 0.00000494
a         676   50 0.00048818 0.00013836 0.00000542
a         676   50 0.00107276 0.00020760 0.00001343
b         676   50 0.00085764 0.00014383 0.00000475
c         676   50 0.00093915 0.00015582 0.00000546
g         676   50 0.00072643 0.00013299 0.00000542
ga        26    50 0.00039567 0.00004355 0.00000508
go        26    50 0.00108770 0.00008616 0.00000545
gu        26    50 0.00033540 0.00005454 0.00000413
x         676   50 0.00041880 0.00016639 0.00000527
y         676   50 0.00038484 0.00013754 0.00000473
z         676   50 0.00034265 0.00012408 0.00000385
aa        26    50 0.00041012 0.00004764 0.00000449
az        26    50 0.00015281 0.00003257 0.00000429
za        26    50 0.00033266 0.00004209 0.00000512
zz        26    50 0.00026916 0.00002705 0.00000398
zqzqwwx 0 50 0.00115711 0.00007517 0.00000764
size in bytes=246064 for BruteAutocomplete
size in bytes=246064 for BinarySearchAutocomplete
size in bytes=1092468 for HashListAutocomplete
```

Fourletters.txt

```
init time: 0.04427 for BruteAutocomplete
init time: 0.02541 for BinarySearchAutocomplete
init time: 0.7340 for HashListAutocomplete
search size #match BruteAutoc BinarySear HashListAu
        456976 50 0.01291630 0.02302300 0.00005957
        456976 50 0.00624955 0.00277507 0.00000640
a       17576 50 0.01201844 0.00023391 0.00000885
a       17576 50 0.00702325 0.00020103 0.00000590
b       17576 50 0.00561472 0.00018401 0.00000528
c       17576 50 0.00651395 0.00019993 0.00000839
```

g 17576 50 0.00507764 0.00020330 0.00000605
ga 676 50 0.00565613 0.00009986 0.00000651
go 676 50 0.00521642 0.00007599 0.00000541
gu 676 50 0.00529705 0.00008447 0.00000572
x 17576 50 0.00513353 0.00023855 0.00000545
y 17576 50 0.00513284 0.00021546 0.00000626
z 17576 50 0.00524106 0.00021583 0.00000584
aa 676 50 0.00499993 0.00006971 0.00000549
az 676 50 0.00502677 0.00007123 0.00000882
za 676 50 0.00527186 0.00006976 0.00000598
zz 676 50 0.00500339 0.00007055 0.00000578
zqzqwwx 0 50 0.00419960 0.00007472 0.00000282
size in bytes=7311616 for BruteAutocomplete
size in bytes=7311616 for BinarySearchAutocomplete
size in bytes=40322100 for HashListAutocomplete

Alexa.txt
init time: 0.3398 for BruteAutocomplete
init time: 1.276 for BinarySearchAutocomplete
init time: 4.298 for HashListAutocomplete
search size #match BruteAutoc BinarySear HashListAu
1000000 50 0.03027760 0.03520746 0.00007270
1000000 50 0.01900188 0.00508029 0.00000648
a 69464 50 0.01677146 0.00060303 0.00000649
a 69464 50 0.01581786 0.00052078 0.00000662
b 56037 50 0.01658977 0.00044772 0.00000700
c 65842 50 0.02389029 0.00054073 0.00000764
g 37792 50 0.01524772 0.00038810 0.00000629
ga 6664 50 0.02379048 0.00019856 0.00000691
go 6953 50 0.01513734 0.00015908 0.00000598
gu 2782 50 0.01595229 0.00011767 0.00000993
x 6717 50 0.01487891 0.00018028 0.00000652
y 16765 50 0.01491492 0.00023798 0.00000663
z 8780 50 0.01481942 0.00017001 0.00000670
aa 718 50 0.01584651 0.00008111 0.00000626
az 889 50 0.01557841 0.00008319 0.00000663
za 1718 50 0.01492639 0.00010549 0.00000645
zz 162 50 0.01481666 0.00005974 0.00000639
zqzqwwx 0 50 0.01501771 0.00008893 0.00000265
size in bytes=38204230 for BruteAutocomplete
size in bytes=38204230 for BinarySearchAutocomplete
size in bytes=475893648 for HashListAutocomplete

Question 2:

init time: 0.4535 for BruteAutocomplete
init time: 1.255 for BinarySearchAutocomplete
init time: 4.225 for HashListAutocomplete

search	size	#match	BruteAutoc	BinarySear	HashListAu
	1000000	10000	0.02671835	0.05135850	0.00005766
	1000000	10000	0.01939324	0.04681738	0.00000825
a	69464	10000	0.01754503	0.01721178	0.00000684
a	69464	10000	0.01846890	0.01753888	0.00000668
b	56037	10000	0.01730886	0.01612780	0.00000669
c	65842	10000	0.01833432	0.01739433	0.00000769
g	37792	10000	0.01738952	0.01339686	0.00000653
ga	6664	10000	0.01515341	0.00341972	0.00000656
go	6953	10000	0.01484476	0.00365974	0.00000672
gu	2782	10000	0.01224011	0.00131335	0.00000606
x	6717	10000	0.01399196	0.00345828	0.00000654
y	16765	10000	0.01630545	0.00839222	0.00000653
z	8780	10000	0.01605922	0.00470833	0.00000880
aa	718	10000	0.01127073	0.00030752	0.00000635
az	889	10000	0.01181815	0.00037194	0.00000686
za	1718	10000	0.01043637	0.00062433	0.00000614
zz	162	10000	0.01014984	0.00007961	0.00000563
zqzqwwx	0	10000	0.01068031	0.00008501	0.00000259

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

The number of matches does not change the runtimes of BruteAutocomplete or HashListAutocomplete by much, however, BinarySearchAutocomplete runs slower by approximately by a factor of 10 when the number of matches increases by a factor of 200. Ultimately, there is little change overall in runtime in regard to BruteAutocomplete or HashListAutocomplete, unlike how an increase of matches slows the runtime for BinarySearchAutocomplete.

Question 3:

BruteAutocomplete.topMatches uses a LinkedList instead of an ArrayList because terms are needed to be added to the beginning of the list, which are done faster (constant speed with LinkedLists, while ArrayLists are linear) and easier by LinkedLists (using addFirst method). The terms need to be added to the beginning of the list because PriorityQueues remove elements that are most minimal (elements that have the least weight). So, in order for there to be a list of

the top matches in descending order of weight, elements with the least weight are removed and continually added to the front so that last element added to the list has the most weight of all. The PriorityQueue uses the given Comparator method because PriorityQueues function in that elements are added and sorted in descending so that the elements removed are those that are most minimal. Therefore, the Comparator method changes the default comparing algorithm so that elements within the PriorityQueue are sorted based off the value of a Term's myWeight instance variable.

Question 4:

HashListAutocomplete uses more memory than other implementations of Autocomplete because HashListAutocomplete is the only one of these implementations that uses Maps. Maps take up more memory than Lists because Maps contains keys and values, while Lists just contain values. Therefore, Maps require memory for both a key and value and, therefore, require more memory than Lists. Lists do have indexes; however, those are not physically referenced when using Lists as Lists are ordered and so the indexes are known based off the order of the elements within the List.