1.

Running BenchmarkForAutocomplete for threeletterwords.txt, we get:

init time: 0.007561 for BruteAutocomplete

init time: 0.07555 for BinarySearchAutocomplete

init time: 0.3011 for HashListAutocomplete

init time: 0.03777 for SlowBruteAutocomplete

search size #match BruteAutoc BinarySear HashListAu SlowBruteA

17576 50 0.00535330 0.02271540 0.00003380 0.01123090

17576 50 0.00084380 0.00298920 0.00000850 0.00081940

a 676 50 0.00052700 0.00024020 0.00000640 0.00052280

a 676 50 0.00059000 0.00024520 0.00000680 0.00056680

b 676 50 0.00053320 0.00035890 0.00000740 0.00052590

c 676 50 0.00055830 0.00069070 0.00000780 0.00056190

g 676 50 0.00054510 0.00033610 0.00000790 0.00053500

ga 26 50 0.00025080 0.00006530 0.00000750 0.00021130

go 26 50 0.00024400 0.00008680 0.00000740 0.00019940

gu 26 50 0.00023220 0.00008050 0.00000800 0.00023100

x 676 50 0.00034590 0.00037310 0.00000860 0.00032440

y 676 50 0.00032000 0.00023560 0.00000670 0.00030040

z 676 50 0.00031370 0.00032170 0.00000850 0.00028320

aa 26 50 0.00022100 0.00008780 0.00000710 0.00022790

az 26 50 0.00024610 0.00011800 0.00000710 0.00021810

za 26 50 0.00025800 0.00005170 0.00000740 0.00021170

zz 26 50 0.00024690 0.00005700 0.00000720 0.00020830

zqzqwwx 0 50 0.00246650 0.00004360 0.00000340 0.00042340

size in bytes=246064 for BruteAutocomplete

size in bytes=246064 for BinarySearchAutocomplete

size in bytes=354276 for HashListAutocomplete

size in bytes=246064 for SlowBruteAutocomplete

Running BenchmarkForAutocomplete for fourletterwords.txt, we get:

init time: 0.07107 for BruteAutocomplete

init time: 0.1043 for BinarySearchAutocomplete

init time: 1.764 for HashListAutocomplete

init time: 0.04731 for SlowBruteAutocomplete

search size #match BruteAutoc BinarySear HashListAu SlowBruteA

456976 50 0.01148750 0.01548540 0.00002900 0.00623680

456976 50 0.00521230 0.00382140 0.00000890 0.00619650

a 17576 50 0.00673040 0.00036410 0.00000830 0.00841300

a 17576 50 0.00485340 0.00020200 0.00000760 0.00479040

b 17576 50 0.00471530 0.00022850 0.00000690 0.00474280

c 17576 50 0.00470260 0.00018480 0.00000700 0.00496200

g 17576 50 0.00471140 0.00031450 0.00000920 0.00484360

ga 676 50 0.00460720 0.00010420 0.00000690 0.00466930

go 676 50 0.00459840 0.00007130 0.00000610 0.00470220

gu 676 50 0.00463210 0.00007950 0.00000630 0.00470680

x 17576 50 0.00533770 0.00022640 0.00000710 0.00633850

y 17576 50 0.00727150 0.00037550 0.00000960 0.00673970

z 17576 50 0.00471340 0.00018600 0.00000760 0.00476150

aa 676 50 0.00464190 0.00007350 0.00000760 0.00491230

az 676 50 0.00463350 0.00007520 0.00000770 0.00495330

za 676 50 0.00462620 0.00006690 0.00000720 0.00469330

zz 676 50 0.00468500 0.00006590 0.00000710 0.00468520

zqzqwwx 0 50 0.00519450 0.00007580 0.00000350 0.00461160

size in bytes=7311616 for BruteAutocomplete

size in bytes=7311616 for BinarySearchAutocomplete

size in bytes=11075636 for HashListAutocomplete

size in bytes=7311616 for SlowBruteAutocomplete

Running BenchmarkForAutocomplete for alexa.txt the first time, we get:

init time: 0.5328 for BruteAutocomplete

init time: 2.059 for BinarySearchAutocomplete

init time: 18.53 for HashListAutocomplete

init time: 0.7596 for SlowBruteAutocomplete

search size #match BruteAutoc BinarySear HashListAu SlowBruteA

1000000 50 0.03373440 0.08085990 0.00004600 0.02094010

1000000 50 0.01981700 0.06137000 0.00001370 0.01666170

a 69464 50 0.01706580 0.00279630 0.00001430 0.01647680

a 69464 50 0.01685360 0.00276880 0.00001340 0.01315120

b 56037 50 0.01637420 0.00229880 0.00001240 0.01682270

c 65842 50 0.01614330 0.00150500 0.00001150 0.01497210

g 37792 50 0.01717910 0.00170740 0.00001290 0.01674060

ga 6664 50 0.01503800 0.00027560 0.00000720 0.01114720

go 6953 50 0.01781070 0.00051290 0.00000920 0.01710860

gu 2782 50 0.01662330 0.00033170 0.00001080 0.01578910

x 6717 50 0.01725890 0.00047500 0.00001030 0.01595150

y 16765 50 0.01823150 0.00092070 0.00001150 0.03855170

z 8780 50 0.01664180 0.00060900 0.00001130 0.01667270

aa 718 50 0.01754780 0.00016310 0.00000880 0.01698850

az 889 50 0.01819030 0.00019180 0.00000940 0.03142610

za 1718 50 0.01592170 0.00024430 0.00001110 0.01599230

zz 162 50 0.01587490 0.00010660 0.00000900 0.02163160

zqzqwwx 0 50 0.01869800 0.00011800 0.00001150 0.01975010

size in bytes=38204230 for BruteAutocomplete

size in bytes=38204230 for BinarySearchAutocomplete

size in bytes=98824414 for HashListAutocomplete

size in bytes=38204230 for SlowBruteAutocomplete

2.

Running BenchmarkForAutocomplete for alexa.txt the second time with #matches=10000, we get:

init time: 0.6861 for BruteAutocomplete

init time: 3.027 for BinarySearchAutocomplete

init time: 15.63 for HashListAutocomplete

init time: 0.2933 for SlowBruteAutocomplete

search size #match BruteAutoc BinarySear HashListAu SlowBruteA

1000000 10000 0.04155290 0.06553760 0.00003990 0.02731830

1000000 10000 0.03850020 0.06201380 0.00001210 0.02424050

a 69464 10000 0.02510980 0.01836860 0.00001200 0.02350750

a 69464 10000 0.02556760 0.01837980 0.00001060 0.02349350

b 56037 10000 0.02911010 0.02205690 0.00001070 0.02474300

c 65842 10000 0.02465350 0.01867400 0.00001070 0.02348360

g 37792 10000 0.02573470 0.01460460 0.00001220 0.02494210

ga 6664 10000 0.02243640 0.00402050 0.00001200 0.02549000

go 6953 10000 0.02654550 0.00374110 0.00000820 0.02274520

gu 2782 10000 0.02639570 0.00134830 0.00000810 0.02073950

x 6717 10000 0.03342270 0.00447830 0.00001150 0.02906280

y 16765 10000 0.03251160 0.00989130 0.00001070 0.02489600

z 8780 10000 0.02588950 0.00512030 0.00001040 0.02366190

aa 718 10000 0.01965930 0.00031300 0.00000820 0.01838200

az 889 10000 0.01834070 0.00039770 0.00000870 0.01861590

za 1718 10000 0.02460150 0.00080800 0.00000930 0.01869140

zz 162 10000 0.01994900 0.00008770 0.00000790 0.01766470

zqzqwwx 0 10000 0.01977480 0.00013820 0.00000460 0.01863740

size in bytes=38204230 for BruteAutocomplete

size in bytes=38204230 for BinarySearchAutocomplete

size in bytes=98824414 for HashListAutocomplete

size in bytes=38204230 for SlowBruteAutocomplete

3.

The last for loop in BruteAutocomplete.topMatches uses a LinkedList and not an ArrayList because new elements are added to the 'ret' list by the addFirst method. The addFirst method takes O(1) for a LinkedList, but O(n) for an ArrayList. This is because a LinkedList adds nodes to the beginning, but an ArrayList needs to shift all the existing nodes by one index. Thus, for simpler run-time complexity and faster execution, a LinkedList is used instead.

The PriorityQueue uses a Comparator.comparing(Term::getWeight) in order to specify what the parameter to order the elements inside the PriorityQueue should be. A PriorityQueue stores the elements in an internal order, and by invoking the Comparator statement, we instruct Java to store the Term objects by order of their weights.

4.

HashListAutocomplete is more speed-efficient, but takes far more memory than the other implementations. This is because the other three methods use only use that much memory required to store the Term objects. But, HashListAutocomplete stores list of prefixes of the Term words, along with the pointers to the Term objects. So, apart from storing the data of the Term objects, HashListAutocomplete also stores the various prefixes for all the Term objects. That makes it store more memory.