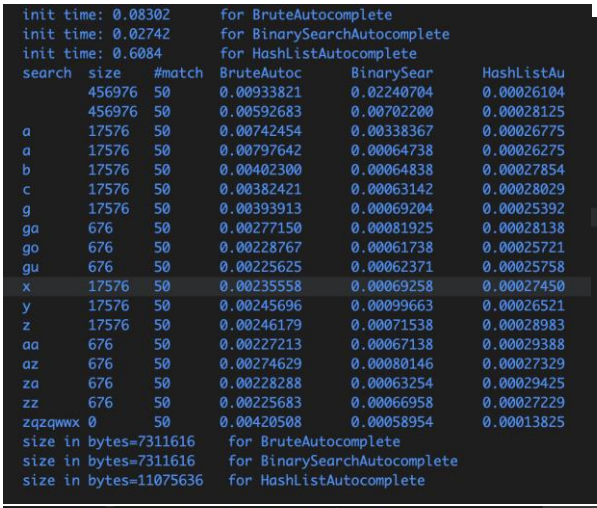
Question 1.



```
init time: 0.08302      for BruteAutocomplete
init time: 0.02742      for BinarySearchAutocomplete
init time: 0.6084       for HashListAutocomplete
search  size    #match  BruteAutoc      BinarySear      HashListAu
        456976  50      0.00933821      0.02240704      0.00026104
        456976  50      0.00592683      0.00702200      0.00028125
a       17576   50      0.00742454      0.00338367      0.00026775
a       17576   50      0.00797642      0.00064738      0.00026275
b       17576   50      0.00402300      0.00064838      0.00027854
c       17576   50      0.00382421      0.00063142      0.00028029
g       17576   50      0.00393913      0.00069204      0.00025392
ga      676     50      0.00277150      0.00081925      0.00028138
go      676     50      0.00228767      0.00061738      0.00025721
gu      676     50      0.00225625      0.00062371      0.00025758
x       17576   50      0.00235558      0.00069258      0.00027450
y       17576   50      0.00245696      0.00099663      0.00026521
z       17576   50      0.00246179      0.00071538      0.00028983
aa      676     50      0.00227213      0.00067138      0.00029388
az      676     50      0.00274629      0.00080146      0.00027329
za      676     50      0.00228288      0.00063254      0.00029425
zz      676     50      0.00225683      0.00066958      0.00027229
zqzqwwx 0       50      0.00420508      0.00058954      0.00013825
size in bytes=7311616    for BruteAutocomplete
size in bytes=7311616    for BinarySearchAutocomplete
size in bytes=11075636   for HashListAutocomplete
```

```
init time: 0.1944       for BruteAutocomplete
init time: 1.834        for BinarySearchAutocomplete
init time: 4.907        for HashListAutocomplete
search  size    #match  BruteAutoc      BinarySear      HashListAu
        1000000 50      0.01970975      0.04240971      0.00019700
        1000000 50      0.01137600      0.00775813      0.00019975
a       69464   50      0.00963738      0.00305425      0.00023929
a       69464   50      0.00887838      0.00111796      0.00024421
b       56037   50      0.00740146      0.00046046      0.00025863
c       65842   50      0.00753346      0.00051217      0.00025296
g       37792   50      0.00713983      0.00046238      0.00025000
ga      6664    50      0.00715163      0.00050317      0.00024925
go      6953    50      0.00717183      0.00040392      0.00025371
gu      2782    50      0.00851000      0.00034300      0.00024963
x       6717    50      0.00797892      0.00039708      0.00025142
y       16765   50      0.00712913      0.00111925      0.00025721
z       8780    50      0.00715304      0.00038396      0.00026646
aa      718     50      0.00819446      0.00028825      0.00026417
az      889     50      0.00812788      0.00028996      0.00027175
za      1718    50      0.00705350      0.00029058      0.00026021
zz      162     50      0.00680583      0.00029475      0.00026433
zqzqwwx 0       50      0.01055008      0.00037763      0.00005692
size in bytes=38204230   for BruteAutocomplete
size in bytes=38204230   for BinarySearchAutocomplete
size in bytes=98824414   for HashListAutocomplete
```

```
init time: 0.007707     for BruteAutocomplete
init time: 0.03707      for BinarySearchAutocomplete
init time: 0.1016       for HashListAutocomplete
search  size    #match  BruteAutoc      BinarySear      HashListAu
        17576   50      0.00384663      0.00976246      0.00004746
        17576   50      0.00264050      0.00267417      0.00017196
a       676     50      0.00107579      0.00057742      0.00043308
a       676     50      0.00081092      0.00056896      0.00031842
b       676     50      0.00095050      0.00059238      0.00025538
c       676     50      0.00082954      0.00067275      0.00024625
g       676     50      0.00138246      0.00056096      0.00024679
ga      26      50      0.00076571      0.00047263      0.00024308
go      26      50      0.00079325      0.00049458      0.00024725
gu      26      50      0.00083367      0.00067854      0.00024888
x       676     50      0.00094813      0.00061279      0.00023863
y       676     50      0.00083029      0.00058875      0.00024746
z       676     50      0.00111508      0.00073454      0.00024525
aa      26      50      0.00073883      0.00051696      0.00025158
az      26      50      0.00072729      0.00064175      0.00023829
za      26      50      0.00074129      0.00056758      0.00025208
zz      26      50      0.00092746      0.00057029      0.00024975
zqzqwwx 0       50      0.00315129      0.00019946      0.00012404
size in bytes=246064     for BruteAutocomplete
size in bytes=246064     for BinarySearchAutocomplete
size in bytes=354276     for HashListAutocomplete
```

Question 2. Let N be the total number of terms, let M be the number of terms that prefix-match a given search term (the size column above), and let k be the number of highest weight terms returned by topMatches (the #match column above). The runtime complexity of BruteAutocomplete is O(N log(k)). The runtime complexity of BinarySearchAutocomplete is O(log(N) + M log(k)). Yet you should notice (as seen in the example timing above) that BruteAutocomplete is similarly efficient or even slightly more efficient than BinarySearchAutocomplete on the empty search String "". Answer the following:

- For the empty search String "", does BruteAutocomplete seem to be asymptotically more efficient than BinarySearchAutocomplete with respect to N, or is it just a constant factor more efficient? To answer, consider the different data sets you benchmarked with varying size.
    - **It does appear that BruteAutocomplete is asymptotically more efficient than BinarySearchAutocomplete with respect to N. When size increases, the time for BinarySearchAutocomplete increases fatster relative to BruteAutocomplete not at a constant rate.**

- Explain why this observation (that BruteAutocomplete is similarly efficient or even slightly more efficient than BinarySearchAutocomplete on the empty search String "") makes sense given the values of N and M.
    - **When the prefix is just the empty string "", all terms are prefix matches, so M=N. This means the big O runtime complexity of BruteAutoComplete is O(Nlog(k)) and for BinarySearchAutocomplete is O(Nlog(N) + Nlog(k)).**
- With respect to N and M, when would you expect BinarySearchAutocomplete to become more efficient than BruteAutocomplete? Does the data validate your expectation? Refer specifically to your data in answering.
    - **BinarySearchAutocomplete becomes more efficient with N is large and M is relatively small, because BinarySearchAutocomplete has near linear runtime complexity to M, and BruteAutocomplete has near linear runtime complexity to N. Looking at the zz prefix for my top left screenshot of data, you can see that BinarySearchAutocomplete runs about 3.3 times faster than BruteAutocomplete. This does validate my expectation.**

Question 3. Run the BenchmarkForAutocomplete again using alexa.txt but doubling matchSize to 100 (matchSize is specified in the runAM method). Again copy and paste your results. Recall that matchSize determines k, the number of highest weight terms returned by topMatches (the #match column above). Do your data support the hypothesis that the dependence of the runtime on k is logarithmic for BruteAutocomplete and BinarySearchAutocomplete?

```
init time: 0.2120       for BruteAutocomplete
init time: 1.937        for BinarySearchAutocomplete
init time: 4.621        for HashListAutocomplete
search  size    #match  BruteAutoc      BinarySear      HashListAu
        1000000 100     0.02279225      0.03442163      0.00021854
        1000000 100     0.00880604      0.01174246      0.00053300
a       69464   100     0.01053925      0.00299683      0.00025292
a       69464   100     0.00886763      0.00068671      0.00024208
b       56037   100     0.00739050      0.00060267      0.00024279
c       65842   100     0.00756508      0.00064992      0.00024354
g       37792   100     0.00715158      0.00064900      0.00024717
ga      6664    100     0.00715063      0.00068454      0.00023779
go      6953    100     0.00709808      0.00053242      0.00024413
gu      2782    100     0.00779563      0.00047567      0.00024658
x       6717    100     0.00678417      0.00100850      0.00025163
y       16765   100     0.00686008      0.00085142      0.00024254
z       8780    100     0.00675546      0.00052900      0.00027229
aa      718     100     0.00754271      0.00044579      0.00026504
az      889     100     0.00752617      0.00044479      0.00026342
za      1718    100     0.00683804      0.00046846      0.00026529
zz      162     100     0.00684988      0.00041579      0.00026004
zqzqwwx 0       100     0.00735921      0.00049942      0.00012346
size in bytes=38204230   for BruteAutocomplete
size in bytes=38204230   for BinarySearchAutocomplete
size in bytes=98824414   for HashListAutocomplete
```

Looking at the data for BruteAutocomplete supports that the runtime is logarithmic with respect to k. We expect the runtimes to increase by log(100)/log(50)=1.17 times. In the first data entry for prefix "", BruteAutocomplete is 1.15 times greater for k = 100 than for k = 50. This aligns well with your expectation. Looking at the second row of data for BinarySearchAutocomplete, we expect BinarySearchAutocomplete with k = 100 to take (log(1,000,000) + 1,000,000log(100))/ log(1,000,000) + 1,000,000log(50)=1.17 times longer than BinarySearchAutocomplete with k = 50. The data shows that it actually takes 1.5 times longer, which is roughly in line with the expectation.

Question 4. Briefly explain why HashListAutocomplete is much more efficient in terms of the empirical runtime of topMatches, but uses more memory than the other Autocomplete implementations.

The Big O asymptotic runtime complexity of TopMatches for HashListAutocomplete is O(1), whereas for BinarySearchAutocomplete it is O(logN + Mlog(k)), and for BruteAutocomplete is O(Nlog(k)). This is reflected in the empirical runtimes, where HashList is typically 2 orders of magnitude less than the other two implementations. However, due to the use of a HashMap for storing and sorting the prefixes and corresponding term lists, as opposed to an array and priority queue, more memory is used. This is seen at the bottom of the pictures, where the program shows how much more data in bytes is used for HashListAutocomplete than for the two other implementations.