Jair Ulises Nino-Espino

Analysis for Autocomplete

Run AutocompletorBenchmark

Objective: Run the **AutocompletorBenchmark** file for each of three implementations of the Autocompletor interface:

- **BruteAutocomplete**
- **BinarySearchAutocomplete**
- **TrieAutocomplete**

Run the benchmark on two files: **fourletterwords.txt** and **fourletterwordshalf.txt**

BruteAutocomplete

fourletterwords.txt	**fourletterwordshalf.txt**
Benchmarking BruteAutocomplete...	Benchmarking BruteAutocomplete...
Found 456976 words	Found 228488 words
Time to initialize - 0.145640719	Time to initialize - 0.048832027
Time for topMatch("") - 0.001983269184	Time for topMatch("") - 6.88384607E-4
Time for topMatch("nenk") - 0.004535006108	Time for topMatch("aenk") - 0.005683502375
Time for topMatch("n") - 0.003497374283	Time for topMatch("a") - 7.82121352E-4
Time for topMatch("ne") - 0.004232379476	Time for topMatch("ae") - 8.94537357E-4
Time for topMatch("notarealword") - 0.004509614259	Time for topMatch("notarealword") - 0.003698296763
Time for topKMatches("", 1) - 0.00677043838	Time for topKMatches("", 1) - 0.00474939255
Time for topKMatches("", 2) - 0.00698339403	Time for topKMatches("", 2) - 0.00423127578
Time for topKMatches("", 4) - 0.00595257895	Time for topKMatches("", 4) - 0.00437112254
Time for topKMatches("", 8) - 0.00555258885	Time for topKMatches("", 8) - 0.00410604982
Time for topKMatches("", 16) - 0.00394274206	Time for topKMatches("", 16) - 0.00429974079
Time for topKMatches("", 32) - 0.00389505803	Time for topKMatches("", 32) - 0.00416202746
Time for topKMatches("", 64) - 0.00445290118	Time for topKMatches("", 64) - 0.00424456354
Time for topKMatches("", 128) - 0.00649602166	Time for topKMatches("", 128) - 0.00431160712
Time for topKMatches("", 256) - 0.00686253532	Time for topKMatches("", 256) - 0.00457646377
Time for topKMatches("nenk", 1) - 0.004530229	Time for topKMatches("aenk", 1) - 0.00432284577
Time for topKMatches("nenk", 2) - 0.00459328528	Time for topKMatches("aenk", 2) - 0.00414342219
Time for topKMatches("nenk", 4) - 0.0043377707	Time for topKMatches("aenk", 4) - 0.00436645465
Time for topKMatches("nenk", 8) - 0.00535923377	Time for topKMatches("aenk", 8) - 0.00423405204
Time for topKMatches("nenk", 16) - 0.00498258842	Time for topKMatches("aenk", 16) - 0.00421789991
Time for topKMatches("nenk", 32) - 0.00395628976	Time for topKMatches("aenk", 32) - 0.00428915429
Time for topKMatches("nenk", 64) - 0.00409138234	Time for topKMatches("aenk", 64) - 0.00415507263
Time for topKMatches("nenk", 128) - 0.00404151722	Time for topKMatches("aenk", 128) - 0.00426644948
Time for topKMatches("nenk", 256) - 0.00514810884	Time for topKMatches("aenk", 256) - 0.00416692098
Time for topKMatches("n", 1) - 0.00421913427	Time for topKMatches("a", 1) - 0.00432489593
Time for topKMatches("n", 2) - 0.00400254902	Time for topKMatches("a", 2) - 0.00417612513
Time for topKMatches("n", 4) - 0.00379413128	Time for topKMatches("a", 4) - 0.00416582265
Time for topKMatches("n", 8) - 0.00372316363	Time for topKMatches("a", 8) - 0.00425768347
Time for topKMatches("n", 16) - 0.00381885615	Time for topKMatches("a", 16) - 0.00414707905
Time for topKMatches("n", 32) - 0.00382476901	Time for topKMatches("a", 32) - 0.00438125443
Time for topKMatches("n", 64) - 0.00389446385	Time for topKMatches("a", 64) - 0.00420017457
Time for topKMatches("n", 128) - 0.00383629837	Time for topKMatches("a", 128) - 0.00439726795
Time for topKMatches("n", 256) - 0.00385627885	Time for topKMatches("a", 256) - 0.00444010907
Time for topKMatches("ne", 1) - 0.00375680665	Time for topKMatches("ae", 1) - 0.00419016178
Time for topKMatches("ne", 2) - 0.00371830853	Time for topKMatches("ae", 2) - 0.00431955574
Time for topKMatches("ne", 4) - 0.00372160906	Time for topKMatches("ae", 4) - 0.00421562088
Time for topKMatches("ne", 8) - 0.00380255738	Time for topKMatches("ae", 8) - 0.0042381992
Time for topKMatches("ne", 16) - 0.00367370971	Time for topKMatches("ae", 16) - 0.00409622991
Time for topKMatches("ne", 32) - 0.00322848967	Time for topKMatches("ae", 32) - 0.00426790051
Time for topKMatches("ne", 64) - 0.00326743906	Time for topKMatches("ae", 64) - 0.00428386214
Time for topKMatches("ne", 128) - 0.00329691602	Time for topKMatches("ae", 128) - 0.00421583678
Time for topKMatches("ne", 256) - 0.00328377875	Time for topKMatches("ae", 256) - 0.00435326766
Time for topKMatches("notarealword", 1) - 0.00266806615	Time for topKMatches("notarealword", 1) - 0.00334826997
Time for topKMatches("notarealword", 2) - 0.00272494314	Time for topKMatches("notarealword", 2) - 0.00324216988
Time for topKMatches("notarealword", 4) - 0.00262895922	Time for topKMatches("notarealword", 4) - 0.00335906701
Time for topKMatches("notarealword", 8) - 0.00265251208	Time for topKMatches("notarealword", 8) - 0.00330656737

Time for topKMatches("notarealword", 16) - 0.002670901 Time for topKMatches("notarealword", 32) - 0.00308450495 Time for topKMatches("notarealword", 64) - 0.00341339656 Time for topKMatches("notarealword", 128) - 0.0033202478 Time for topKMatches("notarealword", 256) - 0.00335469993	Time for topKMatches("notarealword", 16) - 0.00330345539 Time for topKMatches("notarealword", 32) - 0.00330592556 Time for topKMatches("notarealword", 64) - 0.00331122162 Time for topKMatches("notarealword", 128) - 0.00362447589 Time for topKMatches("notarealword", 256) - 0.00167757394

BinarySearchAutocomplete

fourletterwords.txt	fourletterwordshalf.txt
Benchmarking BinarySearchAutocomplete... Found 456976 words Time to initialize - 0.047752735 Time for topMatch("") - 0.001594267355 Time for topMatch("nenk") - 4.269655E-6 Time for topMatch("n") - 3.0213528E-5 Time for topMatch("ne") - 2.118295E-6 Time for topMatch("notarealword") - 3.869317E-6 Time for topKMatches("", 1) - 0.00601823719 Time for topKMatches("", 2) - 0.00702785099 Time for topKMatches("", 4) - 0.010301432 Time for topKMatches("", 8) - 0.01246310157 Time for topKMatches("", 16) - 0.0150247712 Time for topKMatches("", 32) - 0.01768085141 Time for topKMatches("", 64) - 0.02071609782 Time for topKMatches("", 128) - 0.02386099254 Time for topKMatches("", 256) - 0.02640055212 Time for topKMatches("nenk", 1) - 3.00803E-6 Time for topKMatches("nenk", 2) - 1.85091E-6 Time for topKMatches("nenk", 4) - 1.43256E-6 Time for topKMatches("nenk", 8) - 1.36126E-6 Time for topKMatches("nenk", 16) - 1.39845E-6 Time for topKMatches("nenk", 32) - 1.37341E-6 Time for topKMatches("nenk", 64) - 1.38582E-6 Time for topKMatches("nenk", 128) - 1.3604E-6 Time for topKMatches("nenk", 256) - 1.43168E-6 Time for topKMatches("n", 1) - 2.1278095E-4 Time for topKMatches("n", 2) - 2.3784843E-4 Time for topKMatches("n", 4) - 3.7547032E-4 Time for topKMatches("n", 8) - 5.3853038E-4 Time for topKMatches("n", 16) - 5.6717293E-4 Time for topKMatches("n", 32) - 7.0776538E-4 Time for topKMatches("n", 64) - 8.1200715E-4 Time for topKMatches("n", 128) - 9.4702614E-4 Time for topKMatches("n", 256) - 0.00108742959 Time for topKMatches("ne", 1) - 9.38041E-6 Time for topKMatches("ne", 2) - 1.104953E-5 Time for topKMatches("ne", 4) - 1.479929E-5 Time for topKMatches("ne", 8) - 1.843913E-5 Time for topKMatches("ne", 16) - 2.185562E-5 Time for topKMatches("ne", 32) - 3.460756E-5 Time for topKMatches("ne", 64) - 3.105857E-5 Time for topKMatches("ne", 128) - 4.030366E-5 Time for topKMatches("ne", 256) - 5.152481E-5 Time for topKMatches("notarealword", 1) - 9.6153E-7 Time for topKMatches("notarealword", 2) - 5.9699E-7 Time for topKMatches("notarealword", 4) - 6.8525E-7 Time for topKMatches("notarealword", 8) - 6.7786E-7 Time for topKMatches("notarealword", 16) - 7.0457E-7 Time for topKMatches("notarealword", 32) - 5.4418E-7 Time for topKMatches("notarealword", 64) - 5.2171E-7 Time for topKMatches("notarealword", 128) - 4.6598E-7 Time for topKMatches("notarealword", 256) - 4.639E-7	Benchmarking BinarySearchAutocomplete... Found 228488 words Time to initialize - 0.047140982 Time for topMatch("") - 3.1694346E-4 Time for topMatch("aenk") - 1.911184E-6 Time for topMatch("a") - 1.8948363E-5 Time for topMatch("ae") - 2.180903E-6 Time for topMatch("notarealword") - 4.255832E-6 Time for topKMatches("", 1) - 0.00274895106 Time for topKMatches("", 2) - 0.0034120484 Time for topKMatches("", 4) - 0.00467305079 Time for topKMatches("", 8) - 0.00569993735 Time for topKMatches("", 16) - 0.01208889339 Time for topKMatches("", 32) - 0.01051502687 Time for topKMatches("", 64) - 0.013881295 Time for topKMatches("", 128) - 0.01274380494 Time for topKMatches("", 256) - 0.01479419398 Time for topKMatches("aenk", 1) - 2.90111E-6 Time for topKMatches("aenk", 2) - 1.76579E-6 Time for topKMatches("aenk", 4) - 1.19638E-6 Time for topKMatches("aenk", 8) - 1.31639E-6 Time for topKMatches("aenk", 16) - 1.16773E-6 Time for topKMatches("aenk", 32) - 1.21179E-6 Time for topKMatches("aenk", 64) - 1.78131E-6 Time for topKMatches("aenk", 128) - 1.25592E-6 Time for topKMatches("aenk", 256) - 2.37512E-6 Time for topKMatches("a", 1) - 2.3899064E-4 Time for topKMatches("a", 2) - 2.6147925E-4 Time for topKMatches("a", 4) - 3.6096484E-4 Time for topKMatches("a", 8) - 5.6932242E-4 Time for topKMatches("a", 16) - 7.450616E-4 Time for topKMatches("a", 32) - 8.5428186E-4 Time for topKMatches("a", 64) - 0.0010443209 Time for topKMatches("a", 128) - 0.00116784503 Time for topKMatches("a", 256) - 0.00144462256 Time for topKMatches("ae", 1) - 1.143184E-5 Time for topKMatches("ae", 2) - 1.502979E-5 Time for topKMatches("ae", 4) - 2.337739E-5 Time for topKMatches("ae", 8) - 3.061688E-5 Time for topKMatches("ae", 16) - 2.690451E-5 Time for topKMatches("ae", 32) - 4.38652E-5 Time for topKMatches("ae", 64) - 3.803425E-5 Time for topKMatches("ae", 128) - 4.831992E-5 Time for topKMatches("ae", 256) - 7.69605E-5 Time for topKMatches("notarealword", 1) - 1.15843E-6 Time for topKMatches("notarealword", 2) - 1.08897E-6 Time for topKMatches("notarealword", 4) - 1.03693E-6 Time for topKMatches("notarealword", 8) - 9.8529E-7 Time for topKMatches("notarealword", 16) - 8.3999E-7 Time for topKMatches("notarealword", 32) - 7.962E-7 Time for topKMatches("notarealword", 64) - 7.4773E-7 Time for topKMatches("notarealword", 128) - 8.1788E-7 Time for topKMatches("notarealword", 256) - 7.7332E-7

TrieAutocomplete - fourletterwords.txt

fourletterwords.txt	fourletterwordshalf.txt
Benchmarking TrieAutocomplete... Found 456976 words Time to initialize - 0.17897016 Created 475255 nodes	Benchmarking TrieAutocomplete... Found 228488 words Time to initialize - 0.072958179 Created 237628 nodes

```
Time for topMatch("") - 3.155393E-6
Time for topMatch("nenk") - 2.04709E-7
Time for topMatch("n") - 1.997462E-6
Time for topMatch("ne") - 1.34094E-6
Time for topMatch("notarealword") - 2.07081E-7
Time for topKMatches("", 1) -  9.74095E-5
Time for topKMatches("", 2) -  4.01309E-5
Time for topKMatches("", 4) -  4.371523E-5
Time for topKMatches("", 8) -  1.8607452E-4
Time for topKMatches("", 16) -  2.0372477E-4
Time for topKMatches("", 32) -  1.9421007E-4
Time for topKMatches("", 64) -  4.0671996E-4
Time for topKMatches("", 128) -  7.7988481E-4
Time for topKMatches("", 256) -  0.00131244435
Time for topKMatches("nenk", 1) -  3.45866E-6
Time for topKMatches("nenk", 2) -  1.50512E-6
Time for topKMatches("nenk", 4) -  2.81148E-6
Time for topKMatches("nenk", 8) -  2.25642E-6
Time for topKMatches("nenk", 16) -  2.24539E-6
Time for topKMatches("nenk", 32) -  2.29096E-6
Time for topKMatches("nenk", 64) -  2.33607E-6
Time for topKMatches("nenk", 128) -  2.41124E-6
Time for topKMatches("nenk", 256) -  2.54451E-6
Time for topKMatches("n", 1) -  1.087533E-5
Time for topKMatches("n", 2) -  8.79917E-6
Time for topKMatches("n", 4) -  1.160391E-5
Time for topKMatches("n", 8) -  3.411751E-5
Time for topKMatches("n", 16) -  4.865664E-5
Time for topKMatches("n", 32) -  8.494023E-5
Time for topKMatches("n", 64) -  1.4158927E-4
Time for topKMatches("n", 128) -  1.9704523E-4
Time for topKMatches("n", 256) -  3.306589E-4
Time for topKMatches("ne", 1) -  1.31861E-6
Time for topKMatches("ne", 2) -  1.46078E-6
Time for topKMatches("ne", 4) -  2.3181E-6
Time for topKMatches("ne", 8) -  3.99153E-6
Time for topKMatches("ne", 16) -  7.09654E-6
Time for topKMatches("ne", 32) -  1.254451E-5
Time for topKMatches("ne", 64) -  1.801703E-5
Time for topKMatches("ne", 128) -  3.034259E-5
Time for topKMatches("ne", 256) -  6.762641E-5
Time for topKMatches("notarealword", 1) -  2.10059E-6
Time for topKMatches("notarealword", 2) -  7.606E-7
Time for topKMatches("notarealword", 4) -  1.66368E-6
Time for topKMatches("notarealword", 8) -  2.47503E-6
Time for topKMatches("notarealword", 16) -  8.0663E-7
Time for topKMatches("notarealword", 32) -  6.8411E-7
Time for topKMatches("notarealword", 64) -  7.4803E-7
Time for topKMatches("notarealword", 128) -  8.5524E-7
Time for topKMatches("notarealword", 256) -  8.719E-7
```

```
Time for topMatch("") - 3.208463E-6
Time for topMatch("aenk") - 4.02081E-7
Time for topMatch("a") - 3.228709E-6
Time for topMatch("ae") - 2.296157E-6
Time for topMatch("notarealword") - 2.42461E-7
Time for topKMatches("", 1) -  9.552767E-5
Time for topKMatches("", 2) -  4.43174E-5
Time for topKMatches("", 4) -  1.0790461E-4
Time for topKMatches("", 8) -  1.7359673E-4
Time for topKMatches("", 16) -  1.2611312E-4
Time for topKMatches("", 32) -  1.6780971E-4
Time for topKMatches("", 64) -  2.4513405E-4
Time for topKMatches("", 128) -  5.1857052E-4
Time for topKMatches("", 256) -  9.6269294E-4
Time for topKMatches("aenk", 1) -  4.22264E-6
Time for topKMatches("aenk", 2) -  2.60645E-6
Time for topKMatches("aenk", 4) -  3.80357E-6
Time for topKMatches("aenk", 8) -  2.4312E-6
Time for topKMatches("aenk", 16) -  2.34159E-6
Time for topKMatches("aenk", 32) -  3.23715E-6
Time for topKMatches("aenk", 64) -  2.48676E-6
Time for topKMatches("aenk", 128) -  2.54528E-6
Time for topKMatches("aenk", 256) -  3.5069E-6
Time for topKMatches("a", 1) -  6.37488E-6
Time for topKMatches("a", 2) -  1.051109E-5
Time for topKMatches("a", 4) -  1.449254E-5
Time for topKMatches("a", 8) -  2.121797E-5
Time for topKMatches("a", 16) -  4.283189E-5
Time for topKMatches("a", 32) -  8.825876E-5
Time for topKMatches("a", 64) -  1.6970359E-4
Time for topKMatches("a", 128) -  1.8669398E-4
Time for topKMatches("a", 256) -  3.6044753E-4
Time for topKMatches("ae", 1) -  1.15725E-6
Time for topKMatches("ae", 2) -  1.50588E-6
Time for topKMatches("ae", 4) -  2.42643E-6
Time for topKMatches("ae", 8) -  4.58549E-6
Time for topKMatches("ae", 16) -  1.020985E-5
Time for topKMatches("ae", 32) -  1.440151E-5
Time for topKMatches("ae", 64) -  2.375109E-5
Time for topKMatches("ae", 128) -  3.243647E-5
Time for topKMatches("ae", 256) -  6.710209E-5
Time for topKMatches("notarealword", 1) -  1.29941E-6
Time for topKMatches("notarealword", 2) -  5.617E-7
Time for topKMatches("notarealword", 4) -  5.7595E-7
Time for topKMatches("notarealword", 8) -  4.7414E-7
Time for topKMatches("notarealword", 16) -  5.7517E-7
Time for topKMatches("notarealword", 32) -  5.0781E-7
Time for topKMatches("notarealword", 64) -  5.7864E-7
Time for topKMatches("notarealword", 128) -  7.2257E-7
Time for topKMatches("notarealword", 256) -  8.2048E-7
```

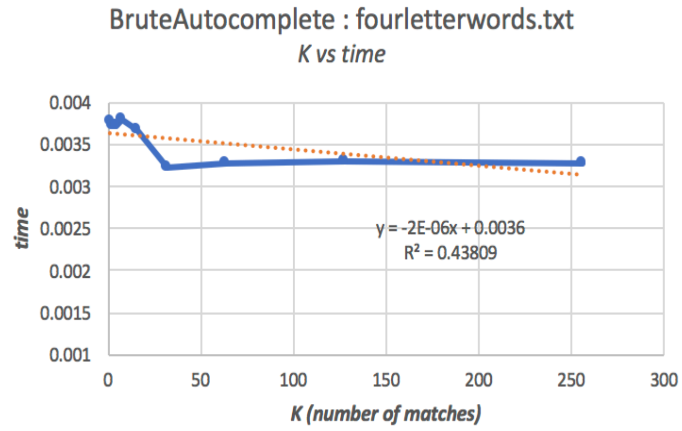
Run Time for Each Implementation

Objective: Using the results from the benchmark program:

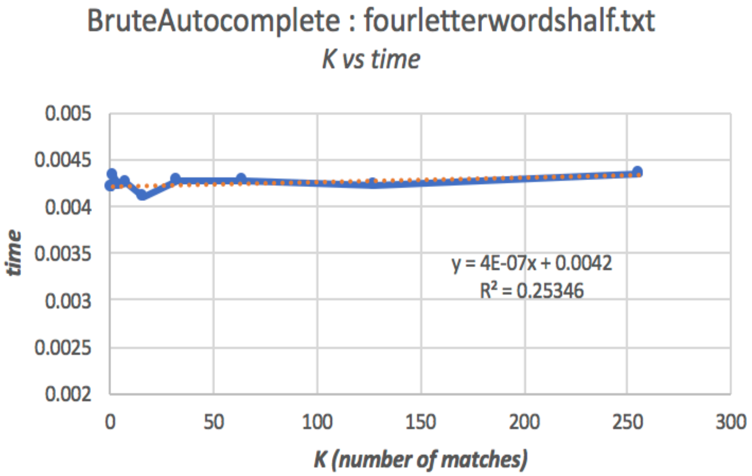
- Make conclusions about **the size of the prefix** and how it **affects runtimes** for each of the Autocompletor implementations.
- How the number of matches **k** that's a parameter to topMatches **affects runtimes**
- Since the number of word/weight pairs in these files differ by a factor of two you should be able to relate these **conclusions to N**, the total number of entries stored in each implementation.

BruteAutocomplete

BruteAutocomplete – fourletterwords.txt	
k	time
1	0.003756807
2	0.003718309
4	0.003721609
8	0.003802557
16	0.00367371
32	0.00322849
64	0.003267439
128	0.003296916
256	0.003283779

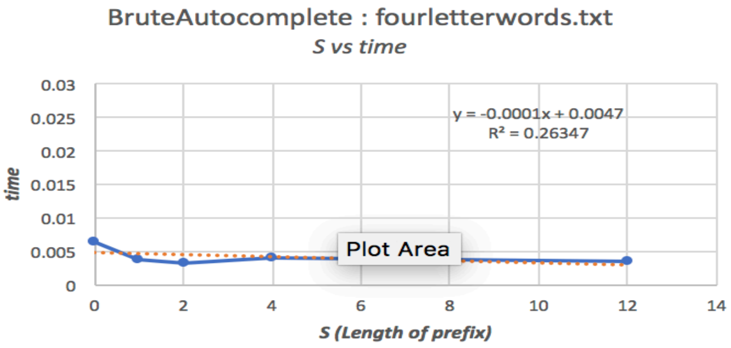


BruteAutocomplete – fourletterwordshalf.txt	
k	time
1	0.004190162
2	0.004319556
4	0.004215621
8	0.004238199
16	0.00409623
32	0.004267901
64	0.004283862
128	0.004215837
256	0.004353268

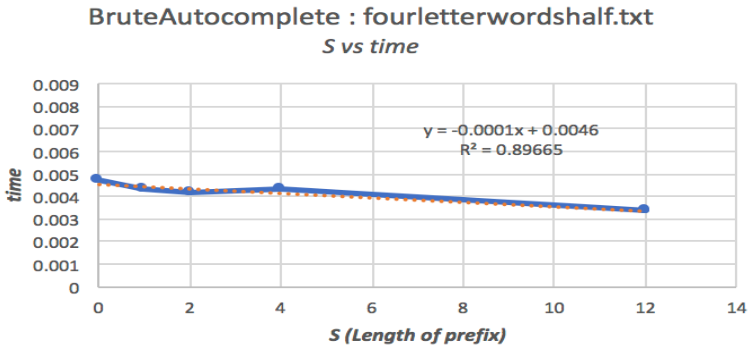


To obtain this data, the length of prefix was kept constant while k (number of matches) was incremented by a factor of two all up to 256. From this data, we can see there is a constant relationship between k and runtime. The R^2 value was determined using a linear trend line which resulted in 0.4809 / 0.25346, these values are not close to one which indicates that the relation between k and runtime is not linear but rather constant. Furthermore, the line of best fit indicates that's it slope is -2.0 x 10^{6} / 4.0 x 10^{-7} which is really small, nearly equal to 0. Therefore, we can say that the big Oh for k and runtime is O(1).

BruteAutocomplete – fourletterwords.txt	
S	time
0	0.006496022
1	0.003836298
2	0.003296916
4	0.004041517
12	0.0033547

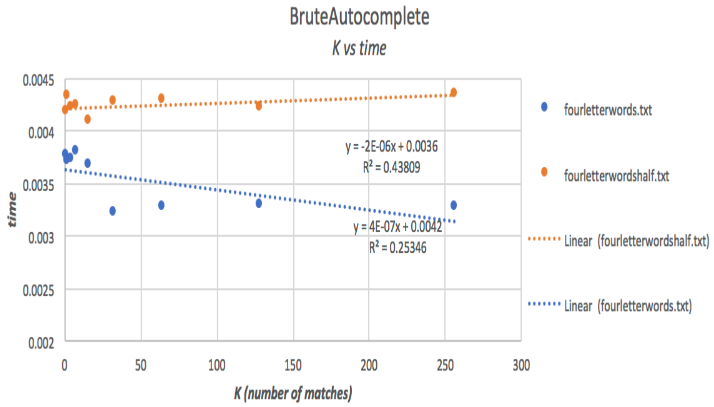


BruteAutocomplete – fourletterwordshalf.txt	
S	time
0	0.004749393
1	0.004324896
2	0.004190162
4	0.004322846
12	0.00334827



To obtain this data, the number of matches, k, was kept constant while S (length of prefix) was incremented by from 0 to 12. From this data, we can see there is a constant relationship between s and runtime. The R^2 value was determined using a linear trend line which resulted in 0.25347 / 0.89665, this value is not close to one which indicates that the relation between S and runtime is not linear but rather constant. Furthermore, the line of best fit for both files had a slope of -0.0001 which is really small, nearly equal to 0. Therefore, we can say that the big Oh for s and runtime is O(1).

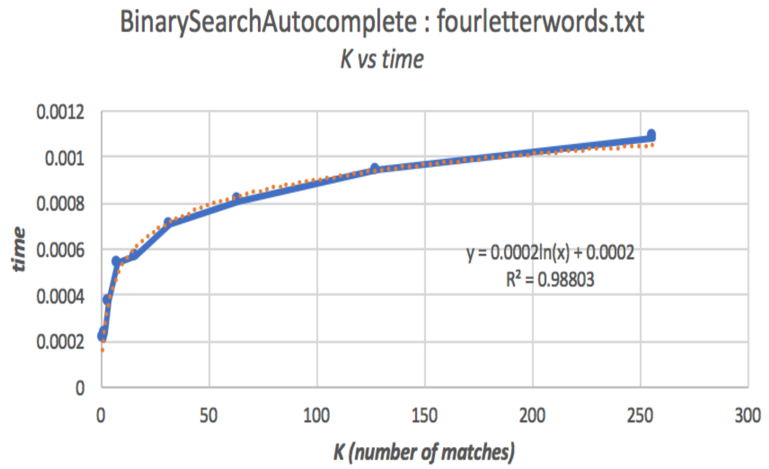
BruteAutocomplete		
k	fourletterwords.txt	fourletterwordshalf.txt
1	0.003756807	0.004190162
2	0.003718309	0.004319556
4	0.003721609	0.004215621
8	0.003802557	0.004238199
16	0.00367371	0.00409623
32	0.00322849	0.004267901
64	0.003267439	0.004283862
128	0.003296916	0.004215837
256	0.003283779	0.004353268



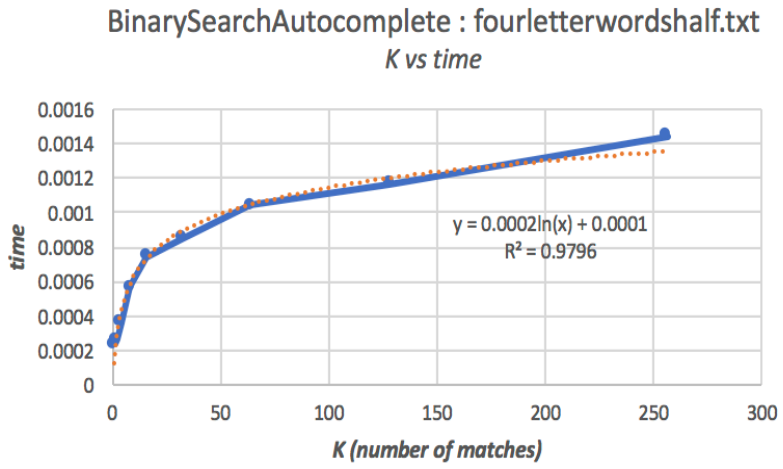
To determine the effect of N (number of entries) on runtime the two files, fourletterwords and fourletterwordshalf, were ran. Fourletterwords had twice as many entries than fourletterwordshalf. The length of the prefix was kept constant, with k incrementing from 1 to 256. From the graph above it is clear that doubling the number of entries does effect the runtime by a small factor. To calculate the factor, all the runtimes from fourletterwords and fourletterwordshalf were averaged and then divided. The calculated factor was 0.83156267 which is close to the value of one. The data above proves that N $\propto$ time and the big Oh is O(1).

BinaryAutocomplete

BinarySearchAutocomplete – fourletterwords.txt	
k	**time**
1	0.000212781
2	0.000237848
4	0.00037547
8	0.00053853
16	0.000567173
32	0.000707765
64	0.000812007
128	0.000947026
256	0.00108743

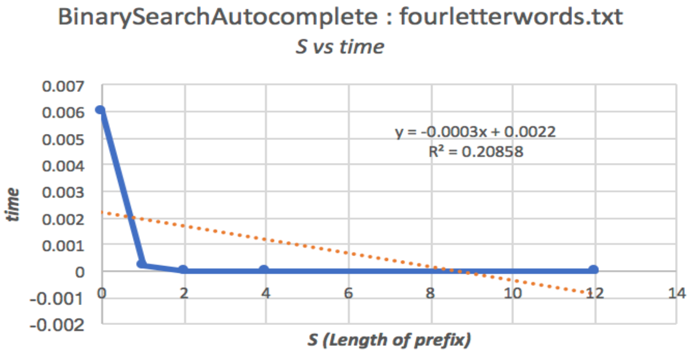


BinarySearchAutocomplete – fourletterwordshalf.txt	
k	**time**
1	0.000238991
2	0.000261479
4	0.000360965
8	0.000569322
16	0.000745062
32	0.000854282
64	0.001044321
128	0.001167845
256	0.001444623

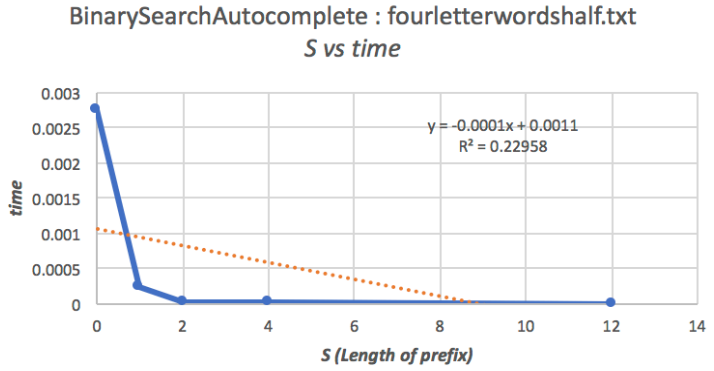


To obtain this data, the length of prefix was kept constant while k (number of matches) was incremented by a factor of two all up to 256. From this data, we can see there is a logarithmic relationship between k and runtime. The R^2 value from these files are 0.98803 / 0.9796 which is close to 1, indicating the reliability of this relationship between k and time. Therefore, the big Oh between k and runtime is O(log k).

BinarySearchAutocomplete – fourletterwords.txt	
S	time
0	0.006018237
1	2.13E-04
2	9.38E-06
4	3.01E-06
12	9.62E-07

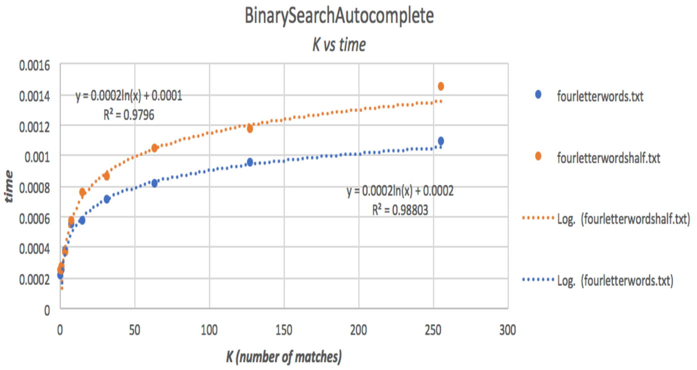


BinarySearchAutocomplete – fourletterwordshalf.txt	
S	time
0	0.002748951
1	2.39E-04
2	1.14E-05
4	2.90E-06
12	1.16E-06



To obtain this data, the number of matches, k, was kept constant while S (length of prefix) was incremented by from 0 to 12. From this data, we can see there is a constant relationship between s and runtime; it is evident that runtime starts high, then drastically drops as prefix length increases and then level off into a straight horizontal line. The R^2 value was determined using a linear trend line which resulted in 0.20858 / 0.22958, this value is not close to one which indicates that the relation between S and runtime is not linear but rather constant. Furthermore, the line of best fit for these files had slopes of -0.003 / -0.0001 which is really small, nearly equal to 0. Therefore, we can say that the big Oh for s and runtime is O (1).

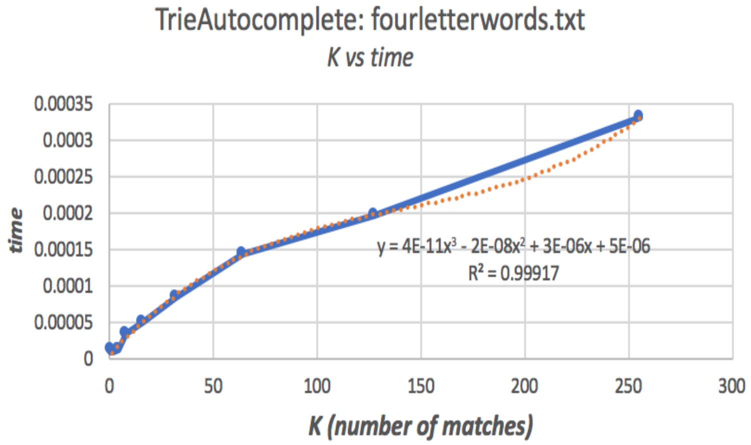
BinarySearchAutocomplete		
k	fourletterwords.txt	fourletterwordshalf.txt
1	0.000212781	0.000238991
2	0.000237848	0.000261479
4	0.00037547	0.000360965
8	0.00053853	0.000569322
16	0.000567173	0.000745062
32	0.000707765	0.000854282
64	0.000812007	0.001044321
128	0.000947026	0.001167845
256	0.00108743	0.001444623



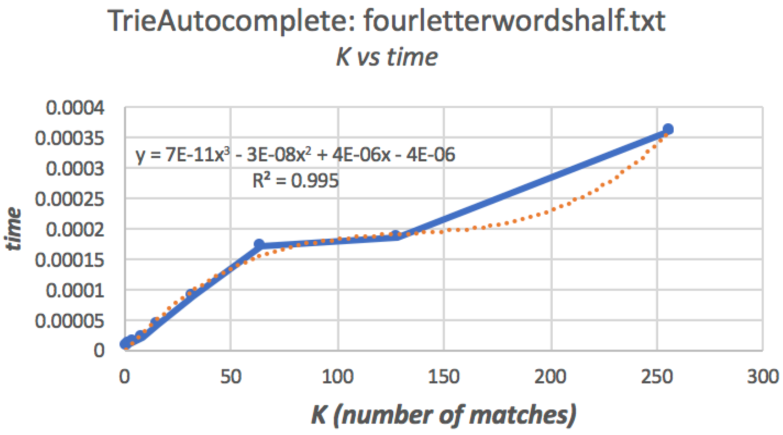
To determine the effect of N (number of entries) on runtime the two files, fourletterwords and fourletterwordshalf, were ran. Fourletterwords was twice as many entries than fourletterwordshalf. The length of the prefix was kept constant, with k incrementing from 1 to 256. From the graph above it is clear that doubling the number of entries does effect the runtime by a small factor. To calculate the factor, the runtimes from fourletterwords and fourletterwordshalf were averaged and then divided. The calculated factor was 0.820416069 which is close to the value of one. The data above proves that N$\propto$time, thus the big Oh is O(1).

TrieAutocomplete

TrieAutocomplete – fourletterwords.txt	
k	**time**
1	1.08753E-05
2	8.79917E-06
4	1.16039E-05
8	3.41175E-05
16	4.86566E-05
32	8.49402E-05
64	0.000141589
128	0.000197045
256	0.000330659

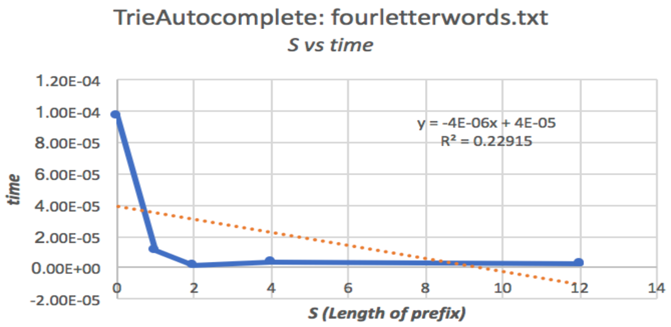


TrieAutocomplete – fourletterwordshalf.txt	
k	**time**
1	6.37488E-06
2	1.05111E-05
4	1.44925E-05
8	2.1218E-05
16	4.28319E-05
32	8.82588E-05
64	0.000169704
128	0.000186694
256	0.000360448

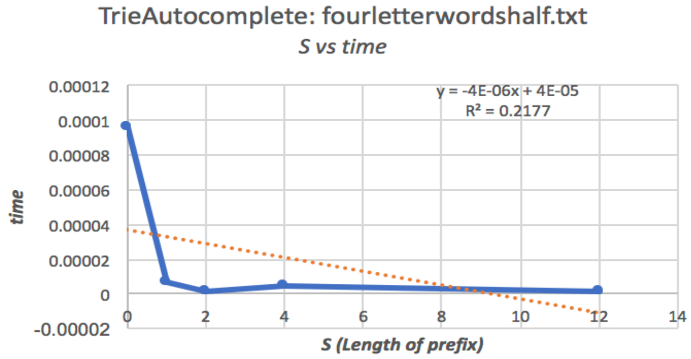


To obtain this data, the length of prefix was kept constant while k (number of matches) was incremented by a factor of two all up to 256. From this data, we can see there is a polynomial relationship between k and runtime. The R^2 value from these files are 0.99917 / 0.995 which is close to 1, indicating the reliability of this relationship between k and time. Thus, this data shows that the big Oh between k and runtime is O(k^3). Although this big Oh would make TrieAutocomplete the slowest algorithm, the runtimes were actually smallest in comparison to the other two implementations, yet the data reveals this relationship. Using greater values for k could aid in finding a more accurate big Oh that may depict that using the Trie algorithm is actually much faster than the other two algorithms.

TrieAutocomplete – fourletterwords.txt	
S	time
0	9.74E-05
1	1.08753E-05
2	1.32E-06
4	3.46E-06
12	2.10E-06

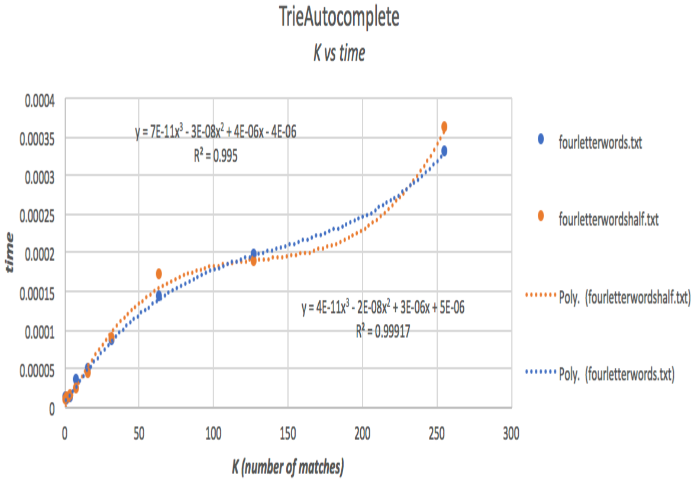


TrieAutocomplete – fourletterwordshalf.txt	
S	time
0	9.55E-05
1	6.37E-06
2	1.16E-06
4	4.22E-06
12	1.30E-06



To obtain this data, the number of matches, k, was kept constant while S (length of prefix) was incremented by from 0 to 12. From this data, we can see there is a constant relationship between s and runtime; it is evident that runtime starts high, then drastically drops as prefix length increases and then level off into a straight horizontal line. The R^2 value was determined using a linear trend line which resulted in 0.22915 / 0.2177, this value is not close to one which indicates that the relation between S and runtime is not linear but rather constant. Furthermore, the line of best fit for both these files had a slope of -4.0 x 10^6 which is really small, nearly equal to 0. Therefore, we can say that the big Oh for s and runtime is O(1).

TrieAutocomplete		
k	**fourletterwords.txt**	**fourletterwordshalf.txt**
1	1.08753E-05	6.37488E-06
2	8.79917E-06	1.05111E-05
4	1.16039E-05	1.44925E-05
8	3.41175E-05	2.1218E-05
16	4.86566E-05	4.28319E-05
32	8.49402E-05	8.82588E-05
64	0.000141589	0.000169704
128	0.000197045	0.000186694
256	0.000330659	0.000360448



To determine the effect of N (number of entries) on runtime the two files, fourletterwords and fourletterwordshalf, were ran. Fourletterwords was twice as many entries than fourletterwordshalf. The length of the prefix was kept constant, with k incrementing from 1 to 256. From the graph above it is clear that doubling the number of entries doesn't increase the runtime since the line of best fit overlaps. Furthermore, the factor difference between both text files were calculate by having the runtimes from fourletterwords and fourletterwordshalf being averaged and then divided. The calculated factor was 0.964192242 which is close to the value of one. The data above proves that $N \propto$ time and thus the relationship between N and time is O(1).

In TrieAutocomplete, as the number of words double in the file, the number of nodes created also doubles. In fourletterwords.txt, there are 475255 nodes created and in fourletterowrdshald.txt, there are 237628 nodes created. When you divide these values by each other you can see that doubling the words in the file, increases the number of nodes being created by a factor of 1.99999579.