Autocomplete Analysis

1. What is the order of growth (big-Oh) of the number of compares (in the worst case) that each of the operations in the Autocomplete data type make, as a function of the number of terms N, the number of matching terms M, and k, the number of matches returned by topKMatches for BinarySearchAutocomplete?

Solution:

(worst cases)	BruteAutocomplete	BinarySearh…	Triautocomple…
topMatch	O(N)	O(logN+M)	O(M)
topKMatchs	O(N+MlogM)	O(logN+MlogM)	O(M)

2. How does the runtime of topKMatches() vary with k, assuming a fixed prefix and set of terms? Provide answers for BruteAutocomplete, BinarySearchAutocomplete and TrieAutocomplete. Justify your answer, with both data and algorithmic analysis.

Solution:

(*baby-names.text*)

	BruteAutocomplete	BinarySearh…	Triautocomple…
Running time to k	the running time doesn't affected too much by k	The running time increase, then decrease as the k increase	Running time decrease as k increase
Analysis	Since the running time is so long, the effect from k variation is not obvious. But, running time increase as k increase		Since the running time is O(M), then the running time won't be affected too much by k
Data supporting	`Time for topKMatches("vinny", 1) - 2.85333222E-4 Time for topKMatches("vinny", 4) - 2.82414559E-4 Time for topKMatches("vinny", 7) - 2.83170282E-4`	`Time for topKMatches("vinny", 1) - 9.79574E-7 Time for topKMatches("vinny", 4) - 1.60458E-6 Time for topKMatches("vinny", 7) - 9.06559E-7`	`Time for topKMatches("vinny", 1) - 9.49756E-7 Time for topKMatches("vinny", 4) - 9.10077E-7 Time for topKMatches("vinny", 7) - 5.32879E-7`

3. Look at the methods topMatch and topKMatches in BruteAutocomplete and BinarySearchAutocomplete and compare both their theoretical and empirical runtimes. Is BinarySearchAutocomplete always guaranteed to perform better than BruteAutocomplete? Justify your answer.

Solution: (*baby-names.text*) (blue colored means less running time)

	BruteAutocomplete	BinarySearh…
topMatch		
topKMatches		
Data supporting (topMatch)	`Time for topMatch("") - 5.6572998E-5 Time for topMatch("vinny") - 6.2967267E-5 Time for topMatch("v") - 4.1044562E-5 Time for topMatch("vi") - 4.3046229E-5 Time for topMatch("notarealword") -`	`Time for topMatch("") - 1.01489183E-4 Time for topMatch("vinny") - 4.696844E-6 Time for topMatch("v") - 4.031902E-6 Time for topMatch("vi") - 5.913708E-6`

	1.69356482E-4	Time for topMatch("notarealword") - 3.468496E-6
Data supporting (topKMatches)	Time for topKMatches("", 1) - 2.52496764E-4 Time for topKMatches("", 4) - 2.14803345E-4 Time for topKMatches("", 7) - 2.17333939E-4 Time for topKMatches("vinny", 1) - 2.74175772E-4 Time for topKMatches("vinny", 4) - 2.66233453E-4 Time for topKMatches("vinny", 7) - 2.69060361E-4 Time for topKMatches("v", 1) - 2.29045866E-4 Time for topKMatches("v", 4) - 2.31638624E-4 Time for topKMatches("v", 7) - 2.31633579E-4 Time for topKMatches("vi", 1) - 2.27504282E-4 Time for topKMatches("vi", 4) - 2.35033015E-4 Time for topKMatches("vi", 7) - 2.37254711E-4 Time for topKMatches("notarealword", 1) - 1.80171164E-4 Time for topKMatches("notarealword", 4) - 1.64783791E-4 Time for topKMatches("notarealword", 7) - 1.70765549E-4	Time for topKMatches("", 1) - 2.32181587E-4 Time for topKMatches("", 4) - 1.28439186E-4 Time for topKMatches("", 7) - 1.27373602E-4 Time for topKMatches("vinny", 1) - 8.5445E-7 Time for topKMatches("vinny", 4) - 8.48299E-7 Time for topKMatches("vinny", 7) - 7.80347E-7 Time for topKMatches("v", 1) - 2.351333E-6 Time for topKMatches("v", 4) - 2.609337E-6 Time for topKMatches("v", 7) - 2.321981E-6 Time for topKMatches("vi", 1) - 1.463417E-6 Time for topKMatches("vi", 4) - 1.645958E-6 Time for topKMatches("vi", 7) - 1.475084E-6 Time for topKMatches("notarealword", 1) - 3.18439E-6 Time for topKMatches("notarealword", 4) - 9.55781E-7 Time for topKMatches("notarealword", 7) - 9.40383E-7

Generally, the running time for "BinarySearch…" is much shorter than "BruteAutocomplete". However, when "topMatch(""), the running time for BruteAutocomplete is shorter.

4. For all three of the Autocompletor implementations, how does increasing the size of the source and increasing the size of the prefix argument affect the runtime of topMatch and topKMatches? (Tip: Benchmark each implementation using fourletterwords.txt, which has all four-letter combinations from aaaa to zzzz, and fourletterwordshalf.txt, which has all four-letter word combinations from aaaa to mzzz. These datasets provide a very clean distribution of words and an exact 1-to-2 ratio of words in source files.)

Solution:

	BruteAutocomplete	BinarySearh…	Triautocomple…
Size of source	Increase source size increase the running time for topMatch , but only slightly increase the running time for topKMatches	The running time for topMatch increases as logN; For topMatches, the running time slightly decrease as N increase	Double the size of source, only slightly increase the running time for topMatch and topKMatches
Size of prefix	The running time of topMatch is like O(log(size of prefix)); Not really affect the running time for topKMatches	The increase of size of prefix dramatically decrease the running time for topMatch and topKMatches	Increase the size of prefix slightly decrease the running time for topMatch (as M decrease); while, topKMatches O(1/size of prefix)
topMatch (fourletter	Found 456976 words Time to initialize - 0.052088531	Found 456976 words Time to initialize - 0.035904729	Found 456976 words Time to initialize - 0.152425668

words) topKMatches (fourletter words)	Time for topMatch("") - 8.80295788E-4 Time for topMatch("nenk") - 0.003869805383 Time for topMatch("n") - 0.00286337357 Time for topMatch("ne") - 0.002973904277 Time for topMatch("notarealword") - 0.002726494755 Time for topKMatches("", 1) - 0.003307386334 Time for topKMatches("", 4) - 0.003249354748 Time for topKMatches("", 7) - 0.003343131615 Time for topKMatches("nenk", 1) - 0.00413196092 Time for topKMatches("nenk", 4) - 0.004214358376 Time for topKMatches("nenk", 7) - 0.004088805376 Time for topKMatches("n", 1) - 0.004076797685 Time for topKMatches("n", 4) - 0.004126821953 Time for topKMatches("n", 7) - 0.004238359689 Time for topKMatches("ne", 1) - 0.004060680165 Time for topKMatches("ne", 4) - 0.004200610182 Time for topKMatches("ne", 7) - 0.004104267814 Time for topKMatches("notarealword", 1) - 0.003701267206 Time for topKMatches("notarealword", 4) - 0.00364311602 Time for topKMatches("notarealword", 7) - 0.003929586078	Time for topMatch("") - 8.60829621E-4 Time for topMatch("nenk") - 3.650933E-6 Time for topMatch("n") - 2.4921387E-5 Time for topMatch("ne") - 3.132082E-6 Time for topMatch("notarealword") - 4.808827E-6 Time for topKMatches("", 1) - 0.00279084056 Time for topKMatches("", 4) - 0.002545338756 Time for topKMatches("", 7) - 0.002717578203 Time for topKMatches("nenk", 1) - 8.64265E-7 Time for topKMatches("nenk", 4) - 7.35708E-7 Time for topKMatches("nenk", 7) - 8.9496E-7 Time for topKMatches("n", 1) - 7.7897661E-5 Time for topKMatches("n", 4) - 7.8463734E-5 Time for topKMatches("n", 7) - 6.9801218E-5 Time for topKMatches("ne", 1) - 3.880117E-6 Time for topKMatches("ne", 4) - 3.754896E-6 Time for topKMatches("ne", 7) - 4.074931E-6 Time for topKMatches("notarealword", 1) - 7.03143E-7 Time for topKMatches("notarealword", 4) - 7.75043E-7 Time for topKMatches("notarealword", 7) - 7.49017E-7	Created 475255 nodes Time for topMatch("") - 5.326175E-6 Time for topMatch("nenk") - 3.43504E-7 Time for topMatch("n") - 3.10955E-6 Time for topMatch("ne") - 2.904207E-6 Time for topMatch("notarealword") - 5.81238E-7 Time for topKMatches("", 1) - 9.4534671E-5 Time for topKMatches("", 4) - 1.9472316E-5 Time for topKMatches("", 7) - 1.5649005E-5 Time for topKMatches("nenk", 1) - 4.44887E-7 Time for topKMatches("nenk", 4) - 4.28333E-7 Time for topKMatches("nenk", 7) - 3.81524E-7 Time for topKMatches("n", 1) - 1.1349643E-5 Time for topKMatches("n", 4) - 1.2736326E-5 Time for topKMatches("n", 7) - 1.7275409E-5 Time for topKMatches("ne", 1) - 7.097192E-6 Time for topKMatches("ne", 4) - 7.137607E-6 Time for topKMatches("ne", 7) - 4.652698E-6 Time for topKMatches("notarealword", 1) - 3.38827E-7 Time for topKMatches("notarealword", 4) - 5.44207E-7 Time for topKMatches("notarealword", 7) - 4.99105E-7
topMatch (fourletter wordshalfa) topKMatches (fourletter wordshalf)	Found 228488 words Time to initialize - 0.016935089 Time for topMatch("") - 4.29562217E-4 Time for topMatch("aenk") - 0.003301617296 Time for topMatch("a") - 4.0926232E-4 Time for topMatch("ae") - 4.23136513E-4 Time for topMatch("notarealword") - 0.002374691437 Time for topKMatches("", 1) - 0.003194195393 Time for topKMatches("", 4) - 0.00320177382 Time for topKMatches("", 7) - 0.003154902172 Time for topKMatches("aenk", 1) - 0.004113450217 Time for topKMatches("aenk", 4) - 0.004106731255 Time for topKMatches("aenk", 7)	Found 228488 words Time to initialize - 0.027061687 Time for topMatch("") - 4.20255875E-4 Time for topMatch("aenk") - 1.925042E-6 Time for topMatch("a") - 2.3044664E-5 Time for topMatch("ae") - 2.556992E-6 Time for topMatch("notarealword") - 5.489022E-6 Time for topKMatches("", 1) - 9.03437466E-4 Time for topKMatches("", 4) - 8.83742672E-4 Time for topKMatches("", 7) - 8.42432955E-4 Time for topKMatches("aenk", 1) - 1.034848E-6 Time for topKMatches("aenk", 4) - 1.121571E-6 Time for topKMatches("aenk",	Found 228488 words Time to initialize - 0.085703608 Created 237628 nodes Time for topMatch("") - 5.120163E-6 Time for topMatch("aenk") - 6.02786E-7 Time for topMatch("a") - 3.243103E-6 Time for topMatch("ae") - 2.876092E-6 Time for topMatch("notarealword") - 2.37959E-7 Time for topKMatches("", 1) - 5.5823884E-5 Time for topKMatches("", 4) - 2.4245797E-5 Time for topKMatches("", 7) - 1.3826653E-5 Time for topKMatches("aenk", 1) - 4.70268E-7 Time for

	- 0.004099416288 Time for topKMatches("a", 1) - 0.004177357768 Time for topKMatches("a", 4) - 0.004125547482 Time for topKMatches("a", 7) - 0.00415536611 Time for topKMatches("ae", 1) - 0.004197036243 Time for topKMatches("ae", 4) - 0.004143359134 Time for topKMatches("ae", 7) - 0.004217356071 Time for topKMatches("notarealword", 1) - 0.003853385465 Time for topKMatches("notarealword", 4) - 0.003928291609 Time for topKMatches("notarealword", 7) - 0.003801869213	7) - 7.63409E-7 Time for topKMatches("a", 1) - 6.3264372E-5 Time for topKMatches("a", 4) - 6.4160605E-5 Time for topKMatches("a", 7) - 7.0095148E-5 Time for topKMatches("ae", 1) - 4.643635E-6 Time for topKMatches("ae", 4) - 4.508337E-6 Time for topKMatches("ae", 7) - 4.664022E-6 Time for topKMatches("notarealword", 1) - 6.23816E-7 Time for topKMatches("notarealword", 4) - 6.04521E-7 Time for topKMatches("notarealword", 7) - 7.75465E-7	topKMatches("aenk", 4) - 4.40747E-7 Time for topKMatches("aenk", 7) - 3.75134E-7 Time for topKMatches("a", 1) - 1.0202926E-5 Time for topKMatches("a", 4) - 1.019546E-5 Time for topKMatches("a", 7) - 1.1403154E-5 Time for topKMatches("ae", 1) - 6.218047E-6 Time for topKMatches("ae", 4) - 6.314512E-6 Time for topKMatches("ae", 7) - 7.28215E-6 Time for topKMatches("notarealword", 1) - 1.33538E-7 Time for topKMatches("notarealword", 4) - 1.66707E-7 Time for topKMatches("notarealword", 7) - 1.80663E-7