Riley Hockett

Autocomplete Analyis

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?

The runtime of the sort is O(log n) and then for addition matching term and each additional K they all need to be ran through. The runtime with each of those is O(m) and O(k). Thus the Order of growth of binary search is O((log n)+m+k).

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.

For BruteAutocomplete the runtime increases linearly with an increase in k assuming a fixed prefix and set of terms this is because of the runtime of BruteAutocomplete which means that it has to go through every extra K that is added.

Data and Graph for BruteAutocomplete:

Time for topKMatches("kh", 1) - 0.004063780012

Time for topKMatches("kh", 11) - 0.004158525613

Time for topKMatches("kh", 21) - 0.004092689439

Time for topKMatches("kh", 31) - 0.004136502445

Time for topKMatches("kh", 41) - 0.004069700799

Time for topKMatches("kh", 51) - 0.004062828909

Time for topKMatches("kh", 61) - 0.004207495787

Time for topKMatches("kh", 71) - 0.004123139105

Time for topKMatches("kh", 81) - 0.004448322603

Time for topKMatches("kh", 91) - 0.004263959832

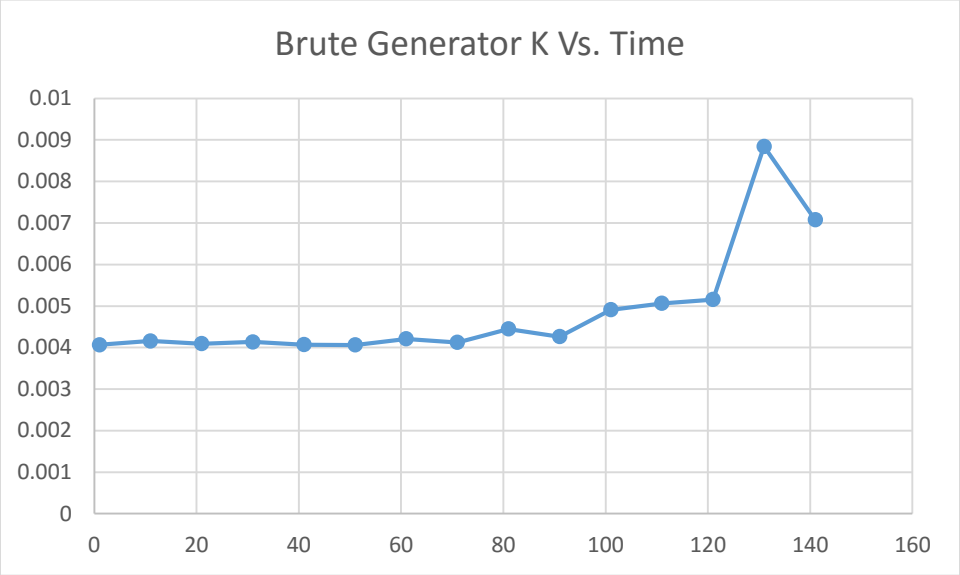
Time for topKMatches("kh", 101) - 0.004910977439

Time for topKMatches("kh", 111) - 0.00506342625708502

Time for topKMatches("kh", 121) - 0.005157845963917526

Time for topKMatches("kh", 131) - 0.008842199713780918

Time for topKMatches("kh", 141) - 0.007073461694483734



For BinarySearchAutocomplete the runtime was more varied. This is because the runtime is not directly dependent on K. When k increases, binary autocomplete just has to return more terms so there is not a strong correlation between a change in k and runtime. There is a general upward trend, but it is challenging to make out exactly what that trend is.

Data and Graph for BinarySearchAutocomplete:

Time for topKMatches("kh", 1) - 2.4798473E-5

Time for topKMatches("kh", 11) - 2.4628779E-5

Time for topKMatches("kh", 21) - 2.4850475E-5

Time for topKMatches("kh", 31) - 2.4691045E-5

Time for topKMatches("kh", 41) - 2.5499825E-5

Time for topKMatches("kh", 51) - 2.4866213E-5

Time for topKMatches("kh", 61) - 2.4889477E-5

Time for topKMatches("kh", 71) - 2.5074908E-5

Time for topKMatches("kh", 81) - 2.4434453E-5

Time for topKMatches("kh", 91) - 2.6828633E-5

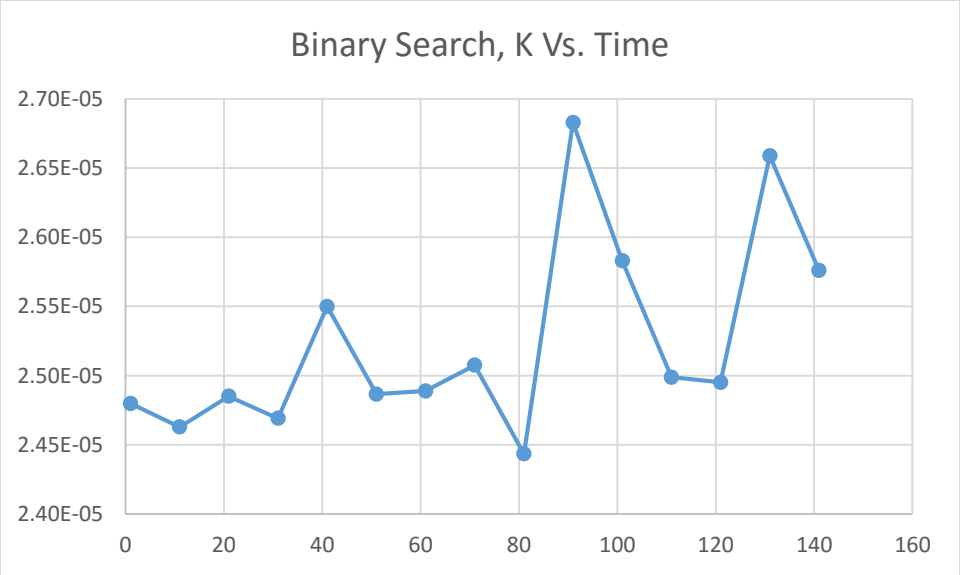
Time for topKMatches("kh", 101) - 2.5828948E-5

Time for topKMatches("kh", 111) - 2.4988693E-5

Time for topKMatches("kh", 121) - 2.4951059E-5

Time for topKMatches("kh", 131) - 2.6588462E-5

Time for topKMatches("kh", 141) - 2.5759155E-5



TrieAutocomplete is very similar in BinarySearch in terms of how the runtime correlates with the K value. The runtime jumps around a lot although it goes upwards, there is not a correlation between the two. However, there are matching spikes at certain values of k, but the rationale behind this is unclear.

Data and Graph for TrieAutocomplete:

Time for topKMatches("kh", 1) - 8.504504E-6

Time for topKMatches("kh", 11) - 8.785729E-6

Time for topKMatches("kh", 21) - 1.0291072E-5

Time for topKMatches("kh", 31) - 9.706725E-6

Time for topKMatches("kh", 41) - 6.673459E-6

Time for topKMatches("kh", 51) - 5.418551E-6

Time for topKMatches("kh", 61) - 6.687828E-6

Time for topKMatches("kh", 71) - 5.538294E-6

Time for topKMatches("kh", 81) - 5.179748E-6

Time for topKMatches("kh", 91) - 0.002273318275

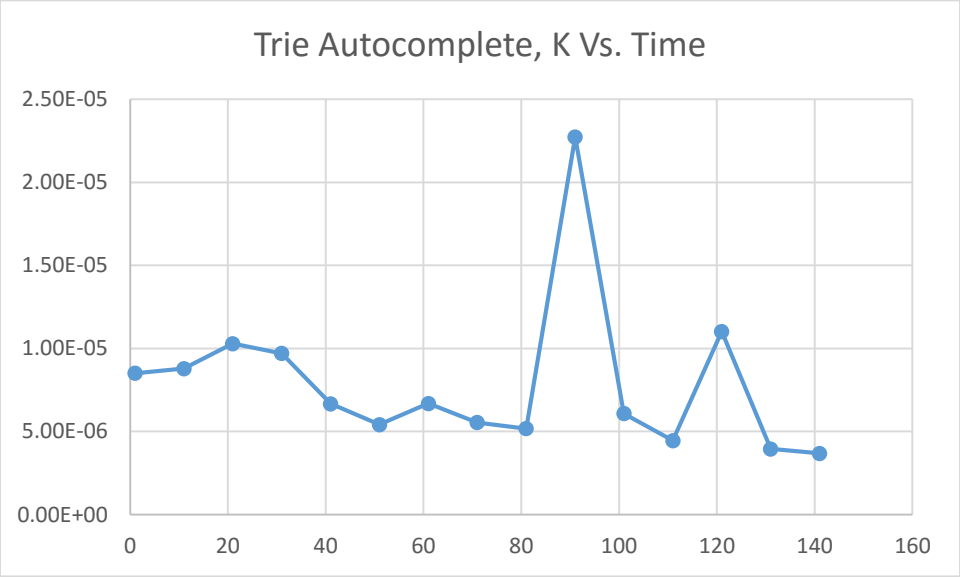
Time for topKMatches("kh", 101) - 6.080901E-6

Time for topKMatches("kh", 111) - 4.457868E-6

Time for topKMatches("kh", 121) - 1.1026637E-5

Time for topKMatches("kh", 131) - 3.948104E-6

Time for topKMatches("kh", 141) - 3.680563E-6



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.

BinarySearchAutocomplete always ends up being faster because in the test with no prefix is faster for binary than brute. This is comparing the entire array. Sort is just faster than going through every single element because it has runtime O(log n).

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.)

BruteGenerator:

For BruteGenerator increasing the source is just increasing the number of items that it has to sort through. Since its runtime is O(n), this just linearly increases the runtime. The same is true for the prefix length. The longer the prefix the more items it has to iterate at runtime O(n).

Binary:

Both the change in source and k matter with binary search but less than brute generator because the runtime of sorting is log (n). The increase in the source only increases the runtime by log n. The change in prefix, however, creates a linear increase in runtime.

Trie:

Increasing the size of the source for the TrieAutocomplete does not matter because the Trie data structure goes node by node, so the amount of source does not affect the runtime basically at all. Increasing size of prefix argument means you have to go down more nodes, but there are less words to sort through at the end. This creates an ambiguous change in runtime based on increase in prefix length, it also depends on the commonality of the prefix.