At most 5 lines

(**Reminder**) Time complexity is a function describing the amount of time an algorithm takes in terms of the amount of input to the algorithm. ... Space complexity is a function describing the amount of memory (space) an algorithm takes in terms of the amount of input to the algorithm.

Binary tree

What is the time and the memory complexity of the algorithm?

* Assuming a tree is well balanced, then the height of the tree would be lg n (average and best case). So space and time complexity would be O(lg n) for an insert. In a worst case scenario, if the tree is just one linked list (like a sorted array), then it would be O(n) as worst case.

Red black tree

What is the time and memory complexity of the algorithm?

* This the sum of the insert function of the binary tree plus the time and memory complexity of the fixup. The tree is well balanced so the insert takes Θ(lg n) time and O(n) memory. The fixup takes at most ½O(lg n) time and memory. So the total time and memory complexity of the algorithm is Θ(lg n).

What are some alternatives that you could use?

* Hashtable (dictionary). This can also hold the same information and is even faster to look up the result. As we only need to search, insert and update values of a given key a hashtable can do all these things in O(1) instead of a red black tree that is O(lg n). If we really want a binary tree we could also use a randomly built binary tree as another tree that tries to be balanced so we do not end up in the worst case of O(n) instead of O(lg n) of a binary tree.

BK-tree

How much storage space is required for the dynamic programming version of the Levenshtein distance?

* This is solved by making a matrix [n x m] (n: string1-length+1, m: string2-length+1), so there is made space for (n+1)\*(m+1) string characters. So we get a space complexity of O(m\*n). This could be reduced to O(m) or O(n) but this would be at the cost of time and because strings usually are not big this should be fine.

What is the time and memory complexity of the BK-tree algorithm?

* The height of a BK Tree depends on the average amount of children a single node has, if we take this to be k on average we get a height of where n is the total amount of items. So in the average case, the depth of the BK Tree will be log n.

If s1 is the average length of a word in the dictionary, s2 the length of the word you are searching and the TOL the tolerance; the time complexity would be O(s1\*s2\*log n). Where we calculate TOL times per level O(log n) the distance O(s1\*s2), assuming we take TOL to be small (TOL = 2). The memory complexity is O(n) as we need to store almost every word that is inserted.

What is the time and memory complexity of comparing the query word with each word in the dictionary?

* Time complexity of O(n\*s1\*s2), with n the dictionary size, s1 average length of a word in the dictionary and s2 the length of the query word. This can be easily understood because we need to check n words and comparing 2 words takes s1\*s2 time. The memory complexity is equal to O(n) because you need a place to store the results and n comparisons give n results.

When many items are added to the tree, it becomes slow. How would you solve this?

* At the cost of some memory we can filter doubles out before trying to insert them. This doesn’t allow any doubles, reducing the size of the tree thus making it quicker.
* Use only tokens and filter doubles

Ranking

Which data structure did you use to store intermediate results in your ranker? What is the reason?

* We stored intermediate results in a dictionary (hash table) because it’s easy to check whether or not an element is already in it (O(1)). In a list this would be O(n). We can also easily fetch all the given keys in O(1) average case. This makes this python structure ideal for the intended purposes.