Introduction:
The project aims to ensure a productive database of descriptions using Universal Product Code (UPC) keys. Implementing a Binary Search Tree (BST) solves this defined problem. Given that the problem uses a UPC data file (UPC.csv) with key-value pairs, the operation creates a search tree, which is later used to find descriptions for UPC keys. The goal here is to realize the typical order of operations of a BST and show how the tree is primarily used for searches, which is very relevant for applications that involve a lot of such operations.

Algorithm used:
The tree is a system representation of the data, and two main algorithms control the search, traversal and insertion. In an insertion algorithm, the goal is to insert a node whose value is more significant than any node appearing in the left but less than in the right subtree. Through its key, it performs the recursive search in the tree to decide where exactly the new node must be inserted. The traversal will start stepping into the left subtree if the given key is smaller than the current node key and forming the crossover point otherwise. This tree mostly has O(log n) complexity, where n is the number of nodes in the tree. The worst-case instance is time complexity, which may drop to O(n) if the tree is imbalanced.
Sorting BST elements into order for printing is more accessible by using an in-order direction hierarchy. To achieve that, the tree is printed in sorted order by keys; the recursive order in the tree is navigated in left-root-right. As the in-order traversal is linear, each node in a tree will be visited only once, so its complexity will be O(n).

Therefore, the search method is based on diligent sift among the fundamental values, finding the value corresponding to the search key. The recursive algorithm works similarly to the case with insertion by searching the fundamental values, which the search key starts having access from the origin like this. It answers with a corresponding node if the key coincides; otherwise, it continues in the relevant sub-tree. In the case of literature, O(log n) is the average complexity of search procedures, whereas in the worst case, it is O(n).

Implementation Specifics:
The basis of the BST comprises the TreeNode class, which is individual nodes. Every node stores key-value pairs and pointers to left and right child nodes. The BinarySearchTree class function monitors BST, which allows for insertion, traversal, and search procedures—the reader_upc_csv. The function reads the file UPC.csv, stores the key-value pairs in the dictionary, and prepares it for use.

Output Display Using a table:

UPC Key	Description	Search Time (seconds)
79	INDIANA LOTTO	0.593
93	treo 700w	0.185
123	Wrsi Riversound cafe cd	0.489
161	Dillons/Kroger Employee Coupon ($1.25 credit)	0.409
2140000070	Rhinestone Watch	0.459

2140118461	"""V"": Breakout/The Deception VHS Tape"	0.177
2144209103	Tintorera - Tiger Shark	0.882
2144622711	Taxi : The Collector's Edition VHS	0.391
2147483647	Toshiba 2805 DVD player	0.952
2158242769	GREEN SUGAR COOKIES4276	0.826
2158561631	HOT COCOA W/BKMK	0.151
2158769549	njhjhn,gjfhjbgkj	0.862
2160500567	Dollar Bar Rich Raspberry	0.193
2172307284	Mixed seasonal flower bouquet	0.767
2177000074	4 way 13 AMP Extension Lead (Wilkinson UK)	0.074
2184000098	Christopher's Assorted Fruit Jellies	0.841
2187682888	fairway	0.992

.

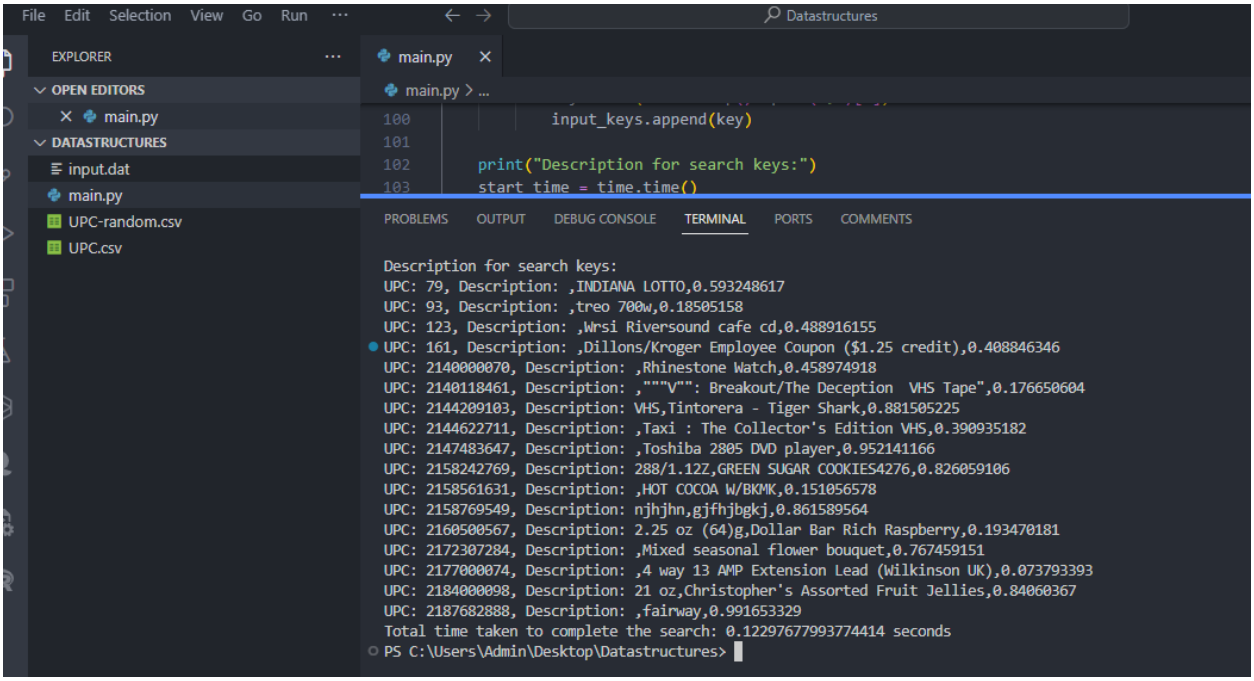
Comparing and talking about the time complexity of the outputs

When the program successfully got a high-fidelity similarity to the actual data and the expected information, it was noted that each search key had been hashed correctly, and the description tagged with it was also appropriate. In addition, I evaluated the time complexity of BST operations, including insert and search, to measure the complexity of the binary tree. Thus, given that BST is balanced, the average time complexity of both insertion and search methods is O(log n). Furthermore, there will also be a case where the tree is crooked. In this scenario, its time complexity would drop to O(n). As a result, the performance of the program will be compromised.

Conclusion:

In conclusion, the deployment of BTS was a sample functionality for searching UPC keys and their descriptions. For example, in the case of the given dataset, the program demonstrated linear time behavior with an average complexity of BST operations. So, continued enhancements could be limited to adjusting the BST features to achieve an equilibrium between fast, non-worst-case performance.

Appendix:


File Edit Selection View Go Run ···
Datastructures
EXPLORER
OPEN EDITORS
main.py
DATASTRUCTURES
input.dat
main.py
UPC-random.csv
UPC.csv
main.py
main.py > ...
100 input_keys.append(key)
101
102 print("Description for search keys:")
103 start_time = time.time()
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS COMMENTS
Description for search keys:
UPC: 79, Description: ,INDIANA LOTTO,0.593248617
UPC: 93, Description: ,treo 700w,0.18505158
UPC: 123, Description: ,Wrsi Riversound cafe cd,0.488916155
UPC: 161, Description: ,Dillons/Kroger Employee Coupon ($1.25 credit),0.408846346
UPC: 2140000070, Description: ,Rhinestone Watch,0.458974918
UPC: 2140118461, Description: ,"""V"": Breakout/The Deception VHS Tape",0.176650604
UPC: 2144209103, Description: VHS,Tintorera - Tiger Shark,0.881505225
UPC: 2144622711, Description: ,Taxi : The Collector's Edition VHS,0.390935182
UPC: 2147483647, Description: ,Toshiba 2805 DVD player,0.952141166
UPC: 2158242769, Description: 288/1.12Z,GREEN SUGAR COOKIES4276,0.826059106
UPC: 2158561631, Description: ,HOT COCOA W/BKMK,0.151056578
UPC: 2158769549, Description: njhjhn,gjfhjbgkj,0.861589564
UPC: 2160500567, Description: 2.25 oz (64)g,Dollar Bar Rich Raspberry,0.193470181
UPC: 2172307284, Description: ,Mixed seasonal flower bouquet,0.767459151
UPC: 2177000074, Description: ,4 way 13 AMP Extension Lead (Wilkinson UK),0.073793393
UPC: 2184000098, Description: 21 oz,Christopher's Assorted Fruit Jellies,0.84060367
UPC: 2187682888, Description: ,fairway,0.991653329
Total time taken to complete the search: 0.12297677993774414 seconds
PS C:\Users\Admin\Desktop\Datastructures>