Solutions to Week 11 Exercises

Ibrahim Chehab

March 29, 2024

1 Questions from FSG Slides

The solutions will cover the questions out of order to better represent the "problem solving" aspect of the FSGs

1.1

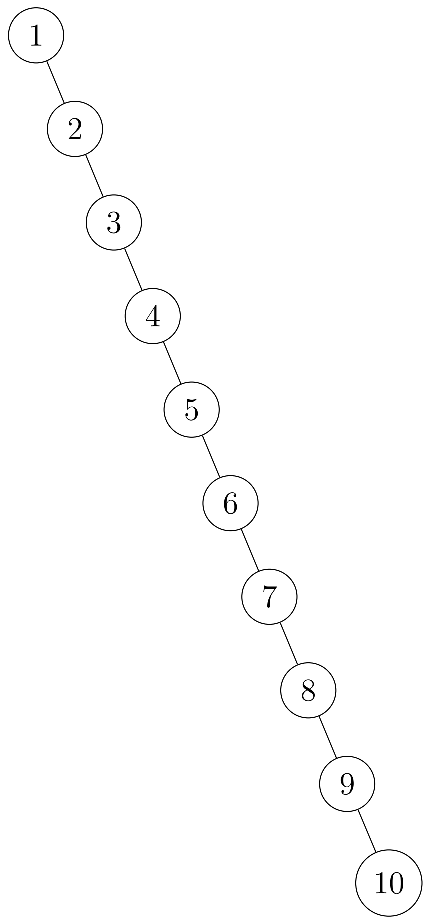
Draw the resulting BSTs for the following lists, assuming the pivot is the element you will always insert first. What is the complexity of inserting the elements in the order given? What does this mean for the BST?

- `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`
- `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`

This question is meant to *visually* show you the complexity of quicksort with different lists. This question assumes the implementation of quicksort where the pivot is the first element in the list (as it is defined in your coursenotes).

1.1.1 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

The resulting BST is as follows:

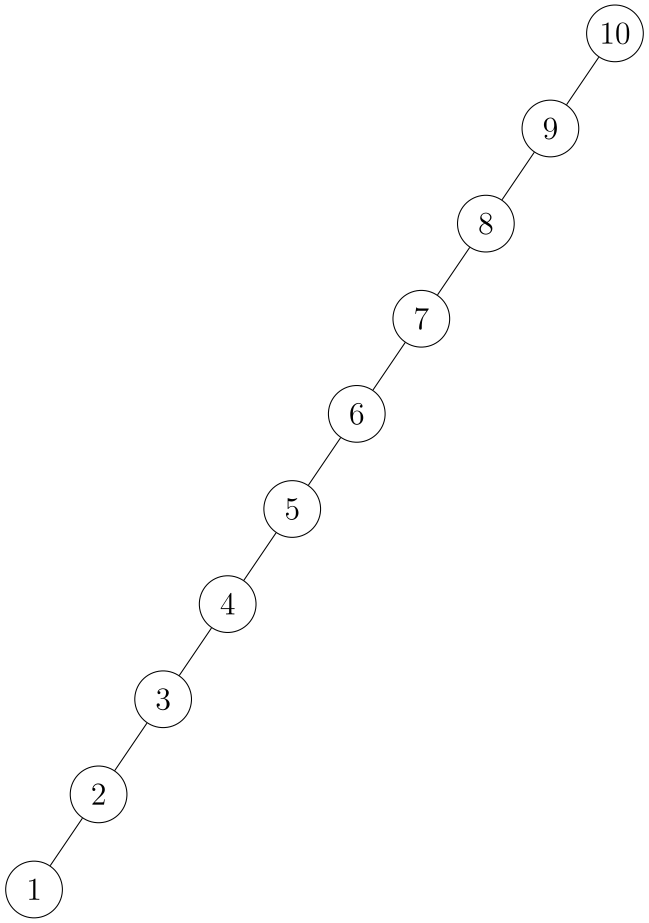


Notice, this tree is effectively a LinkedList. Hence, the insertion complexity is $O(n)$, where n is the number of elements in the list. This means that the BST is effectively a LinkedList.

This makes the total sequence complexity of inserting the elements in the order given $O(n^2)$.

1.1.2 `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`

The resulting BST is as follows:

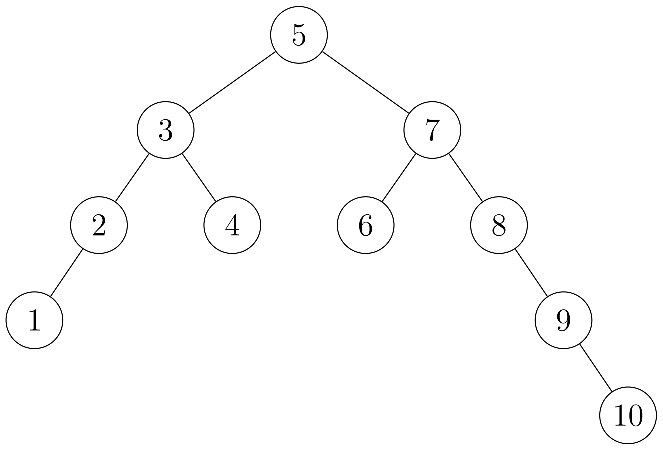


Once again, we notice that this tree is effectively a LinkedList. Hence, the insertion complexity is $O(n)$, where n is the number of elements in the list. This means that the BST is effectively a LinkedList.

This makes the total sequence complexity of inserting the elements in the order given $O(n^2)$.

1.1.3 `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`

The resulting BST is as follows:



This tree is not a LinkedList. The insertion complexity is closer to $O(\log n)$, where n is the number of elements in the list. This means that the BST is effectively a LinkedList.

This makes the total sequence complexity of inserting the elements in the order given $O(n \log n)$.

Notice, the complexity of inserting the elements in the order given is directly related to the complexity of the resulting BST. If we are given a *sorted*, or *nearly sorted* list, the resulting BST will be a LinkedList. This is because the pivot is the first element in the list, and the tree will be unbalanced.

1.2 Question 2

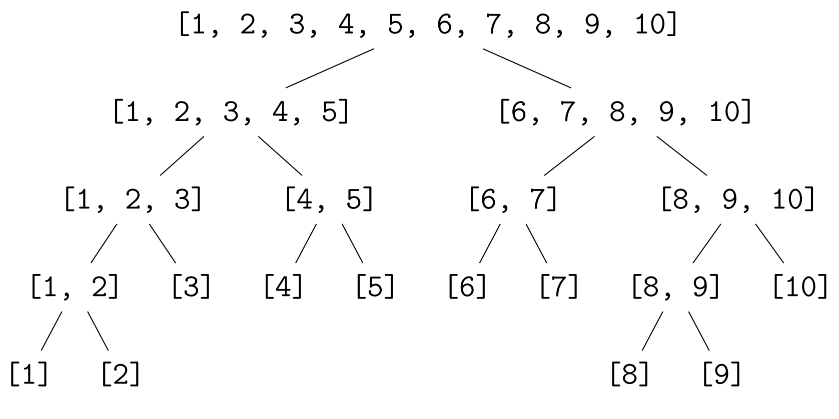
Draw the resulting Binary Tree for the following lists, following the MergeSort sorting algorithm.

- `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`
- `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`
- `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`

1.2.1 `[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]`

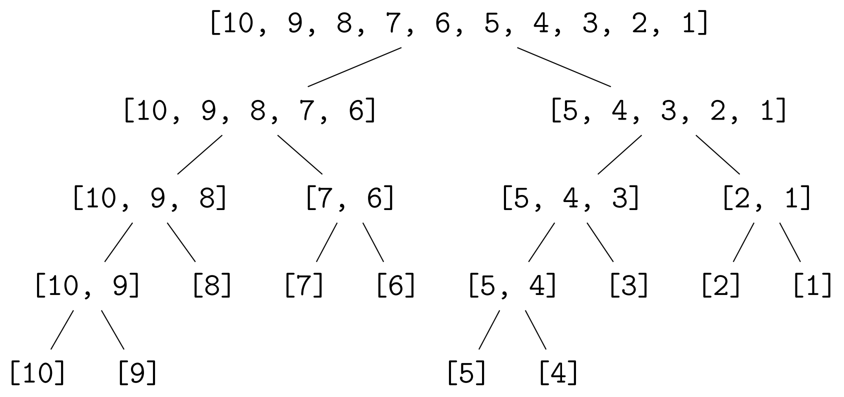
The resulting Binary Tree is as follows: *Note: This BST only represents the splitting up of the list. It does not represent the merging of the lists.*

As a challenge to the reader, try to convert the following binary trees into the merged representation (i.e: Reading the tree from down-to-up should show the process of which the sublists are merged, rather than the order which they are split up).



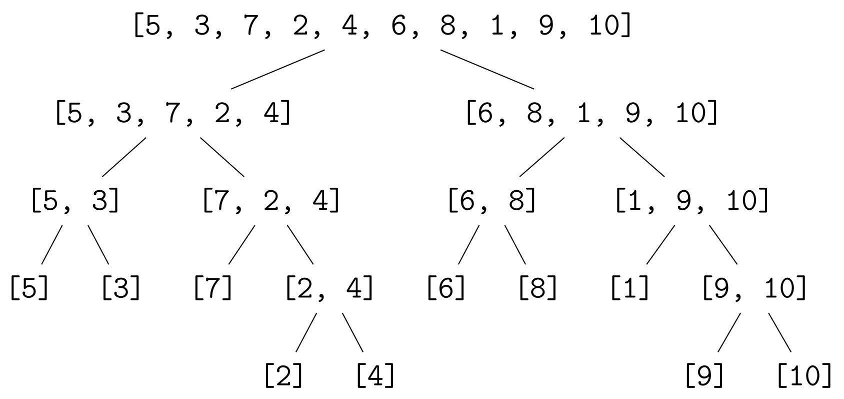
Notice, the tree is close to being balanced.

1.2.2 `[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]`



Notice, the tree is close to being balanced.

1.2.3 `[5, 3, 7, 2, 4, 6, 8, 1, 9, 10]`



Notice, the tree is close to being balanced.

We notice a pattern with *MergeSort* - It always generates a close-to-balanced tree, unlike *QuickSort* which can generate a LinkedList. Does this inherintly mean that *MergeSort* is better than *QuickSort*?

The answer is no, not necissarily. While *MergeSort* is consistently $O(n \log n)$ regardless of the input, the *linear* (i.e: $O(n)$) part of *QuickSort* is faster. This can be seen by the binary trees generated above. With *QuickSort*, we simply need an $O(n)$ in-order traversal to generate a sorted list. With *MergeSort*, we need to merge the lists, which is a $O(2n)$ operation in the worst case.
This is why choosing a sorting algorithm is not as simple as choosing the one with the best complexity; it is important to consider the input data and the variations within it wisely to choose the best algorithm.

1.3 Question 3

Dr. Halstead and Dr. Manning are arguing about the new hospital database system sorting upgrades. Dr. Manning argues Quicksort is the faster option, while Dr. Halstead is a firm believer in the efficiency mergesort. With the help of April, Dr. Choi runs some tests with both sorting algorithms on the hospital dataset. Here is what you know about the dataset:

1. 99% of the time, the hospitals dataset is mostly sorted
2. All the sorting algorithms are run in their non-in-place variants
3. Every other part of their software is as optimized as it can get - There is no other source of slowdowns other than the choice of algorithm

Note: Since these are doctors and not computer scientists, they do not know about TimSort (which would've been the ideal compromise for Will and Natalie)

Given what we know about Gaffney Chicago's database, what did Dr. Choi and April find in their tests?

Solution:

Based on our findings from the previous two questions, we can infer that *MergeSort* is the faster option for the hospital database system. This is because the dataset is mostly sorted, and *MergeSort* is consistently $O(n \log n)$ regardless of the input. As we learned above, when *Quicksort* is given a sorted (or nearly-sorted) list, the complexity shoots up to $O(n^2)$.

Hence, Dr. Halstead is correct in this case.