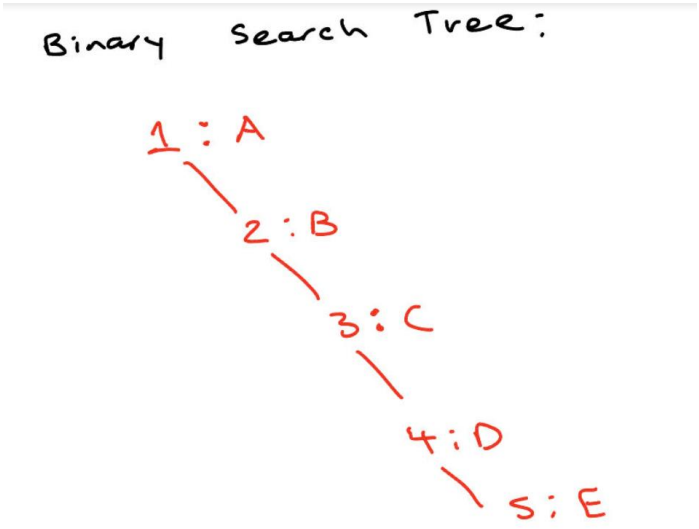
Mechtron 2MD3 Assignment 6 – Michael Giancola

1. [short answer, 3 marks]: Show that, given only the less-than operator (<) and the boolean operators *and* (&&), *or* (||), and *not* (!), it is possible to implement all of the other comparators: `>, <=, >=, ==, !=`

```
a > b = b < a

a <= b = (a < b && !(a < b || b < a))

a >= b = (b < a && !(a < b || b < a))

a == b = !(a < b || b < a)

a != b = (a < b || b < a)
```

2. [short answer, 3 marks]: If we insert the entries (1,*A*), (2,*B*), (3,*C*), (4,*D*), and (5,*E*), in this order, into an initially empty binary search tree, what will it look like?



The above binary search tree is constructed so the left subtree contains only nodes with keys less than the parent node's key and the right subtree contains only nodes with keys greater than the parent node's key (no left in this case). Each node has a key-value pair where the key is the number before the colon and the value is the letter after the colon.; The root node has key 1 and value A, and the right child has key 2 and value B, and so on. In this case, the tree forms a linear structure due to the ascending order of the keys being inserted.

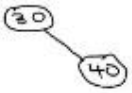
3. [short answer, 3 marks]: Insert, into an empty binary search tree, entries with keys 30, 40, 24, 58, 48, 26, 11, 13 (in this order). Draw the tree after each insertion.

3.

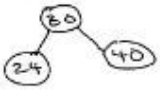
step 1:

*If key is less than root node, descend to left
If key is greater than root node, descend to right

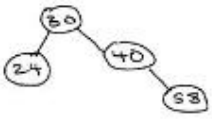
step 2:



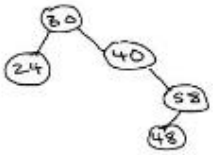
step 3:



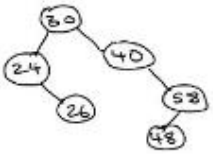
step 4:



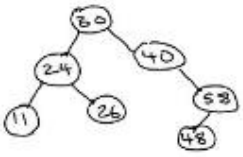
step 5:



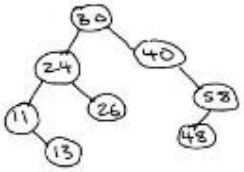
step 6:



step 7:

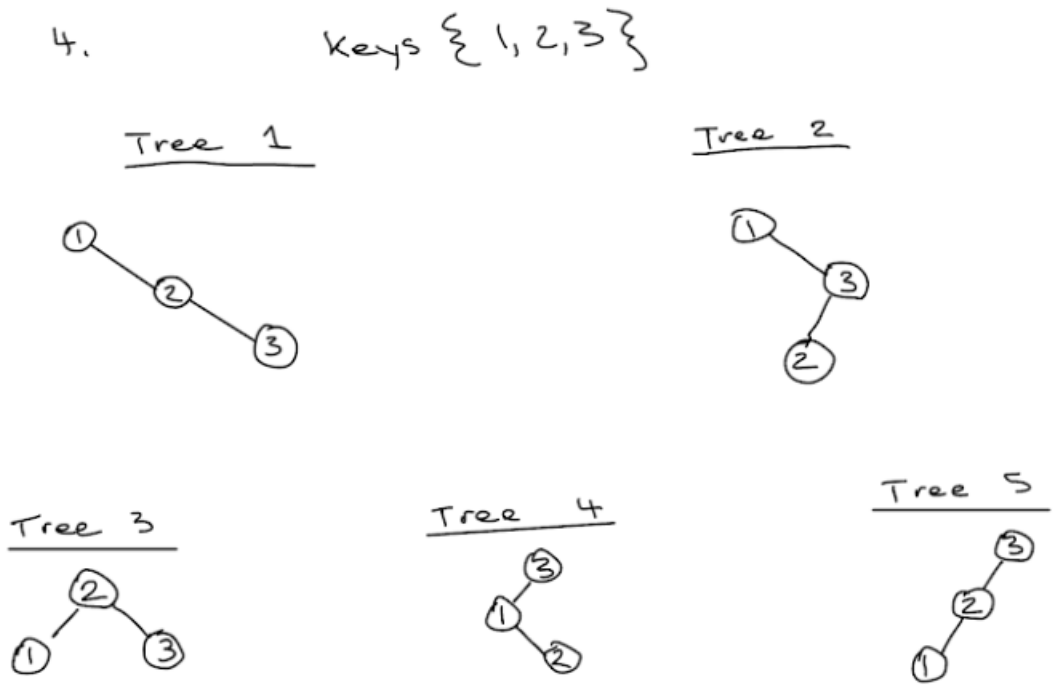


step 8:



4. [short answer, 3 marks]: How many different binary search trees can store the keys {1,2,3}?

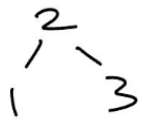
In a binary search tree, the node greater than the root node branches off to the right, and the node less than the root node branches off to the left.



As shown in the diagram above, there are two different variations of binary search trees to store the keys {1,2,3}.

5. [short answer, 3 marks]: Farid claims that the order in which a fixed set of entries is inserted into a binary search tree does not matter—the same tree results every time. Give a small example that proves they are wrong.

If we insert the elements in the order 2,1,3, the resulting binary search tree would look like,

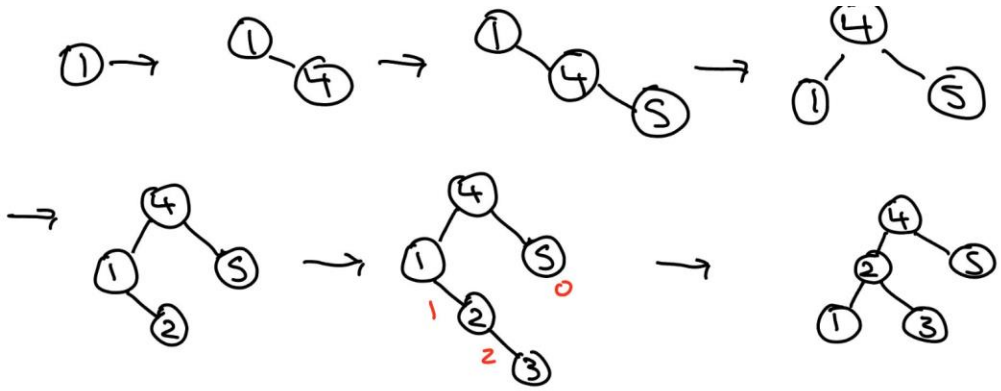


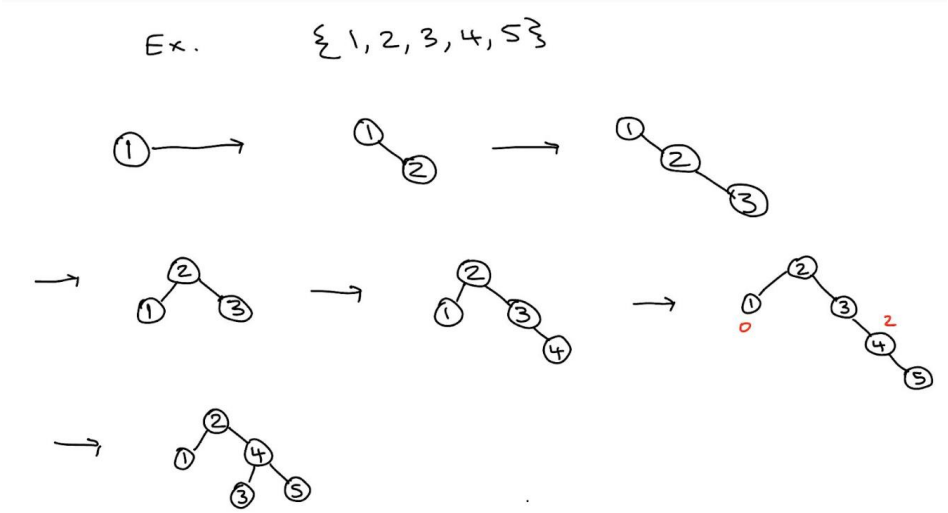
If we insert the elements in the order 1,2,3, the resulting binary search tree would look like,

Therefore, Farid is wrong and the order in which the entries are inserted doesn't result in the same tree every time.

6. [short answer, 3 marks]: Nur claims that the order in which a fixed set of entries is inserted into an AVL tree does not matter—the same AVL tree results every time. Give a small example that proves they are wrong.

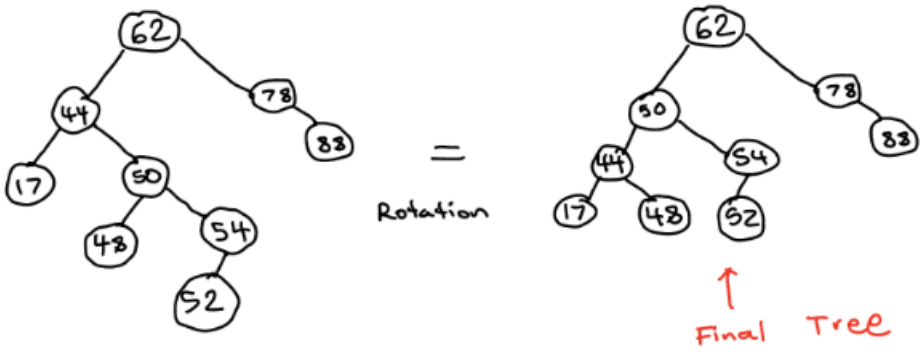
Ex. {1,4,5,2,3}





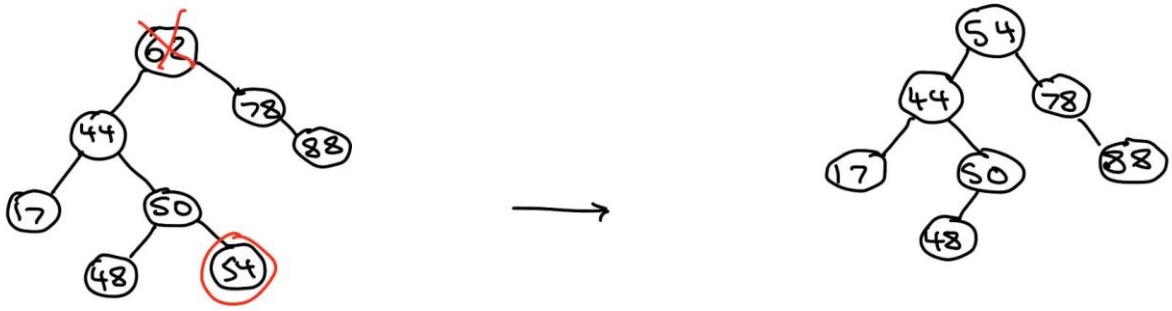
Therefore, Nur's assumption is false as the order in which a fixed set of entries are inserted into an AVL tree does matter.

7. [short answer, 3 marks]: Draw the AVL tree resulting from the insertion of an entry with key 52 into the AVL tree shown on page 3.



The tree rotation moves one node up in the tree and one node down. It's used to change the shape of the tree and to decrease its height by moving the smaller subtree down and the larger subtree up as shown above when inserting the entry with key 52.

8. [short answer, 3 marks]: Draw the AVL tree resulting from the removal of the entry with key 62 from the AVL tree on page 3.



9. [short answer, 3 marks]: Suppose *S* is a list of *n* bits, that is, *n* 0's and 1's. How long will it take to sort *S* with the merge-sort algorithm? What about quick-sort? Provide Big-O for each.

Merge sort will take O(nlogn) time to sort the list S as the merge sort algorithm will split the S into two parts, an n amount of 0's and 1's. The merging process takes O(n) time and the number of times the list id divided due to merging is log(n). When using the quick-sort algorithm to sort list S, a pivot element is chosen from the list, the list is split into two, and the sublists are recursively sorted. It can be O(n^2) or O(nlogn) depending on the implementation. A worst case occurs when the pivot is a unique max or min number of if the pivot is selected second to the front or back of the initial sequence. In this case, there are no unique max or min numbers (they're all 1 or 0) so the run time is also O(nlogn).

10. [short answer, 3 marks]: Suppose *S* is a list of *n* bits, that is, *n* 0's and 1's. How long will it take to sort *S* stably with the bucket-sort algorithm?

The only possible values that each bit can take in list S are 0 and 1 so to use bucket sort for this list we can create two buckets, one for the 0's and one for 1's. Then we can iterate through S and place each element in the corresponding bucket, and after all of the elements have been placed, we can concatenate the two buckets to get the sorted list. Since there are n elements in the list, and we need to iterate through each element once to place it in its bucket, the time complexity of the bucket sort is O(n).

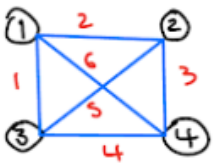
11. [short answer, 3 marks]: Is the bucket-sort algorithm in-place? Why or why not? Give its space complexity in big-O notation.

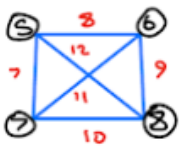
Bucket-sort is not an in-place sorting algorithm as it requires extra space to store the buckets used to sort the elements. The space complexity of bucket sort is O(n+k) where n is the number of elements in the array and k is the number of buckets formed. The space taken by each bucket is O(k) and inside each bucket we have n elements scattered, resulting in a space complexity of O(n+k). In most cases, k is much smaller than n so the space complexity simplifies to O(n).

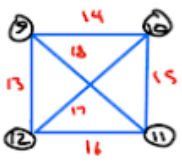
12. [short answer, 3 marks]: Give an example input list that requires merge-sort and heap-sort to take $O(n\log n)$ time to sort, but insertion-sort runs in $O(n)$ time. What if you reverse this list?

This can be shown using the example L = {1,2,3,4}. If the input is already sorted, merge sort and heap sort will take O(nlogn) time because they will always be executed the same way regardless of the order of the list. This will cause insertion sort to take O(n) time because if the list is already sorted, fewer comparisons will take place and this is the best-case scenario for insertion sort. When this input is reversed, the runtime for merge sort and heap sort still takes O(nlogn) time because these sorting algorithms don't depend on the initial arrangement of the list. The runtime for insertion sort will now be O(n^2) because this is the worst-case scenario for insertion sort as it must now perform the max number of comparisons.

13. [short answer, 3 marks]: Draw a simple undirected graph G that has 12 vertices, 18 edges, and 3 connected components. Why would it be impossible to draw G with 3 connected components if G had 66 edges?

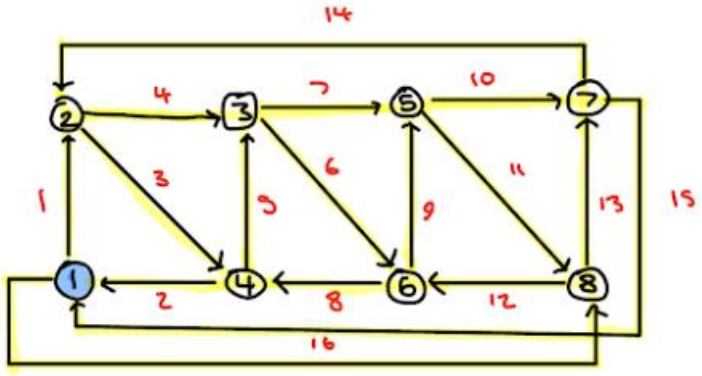






It would not be possible to have 12 vertices and 66 edges because the property of graphs states that the number of edges is less than or equal to n(n-1)/2, where n is the number of vertices. The max number of edges would be if there was only a single component but since there are three connected components, the max number of edges goes down.

14. [short answer, 3 marks]: Draw a simple connected directed graph with 8 vertices and 16 edges such that the in-degree and out-degree of each vertex is 2. Show that there is a single (nonsimple) cycle that includes all the edges of your graph, that is, you can trace all the edges in their respective directions without ever lifting your pencil. (Such a cycle is called a Euler tour.)



15. [short answer, 6 marks]: Let *G* be a graph whose vertices are the integers 1 through 8, and let the adjacent vertices of each vertex be given by the table below:

Vertex	Adjacent Vertices
1	(2, 3, 4)
2	(1, 3, 4)
3	(1, 2, 4)
4	(1, 2, 3, 6)
5	(6, 7, 8)
6	(4, 5, 7)
7	(5, 6, 8)
8	(5, 7)

1. **Draw G:**

2. **Give the sequence of vertices of G visited using DFS traversal starting at vertex 1**

DFS: 1, 2, 3, 4, 6, 5, 7, 8

3. **Give the sequence of vertices visited using BFS traversal starting at vertex 1**

BFS: 1, 2, 3, 4, 6, 5, 7, 8