Andrew Ha – 40088418
Nora Houari
COMP 352
Assignment 4 - Theory

Question 1:
A)

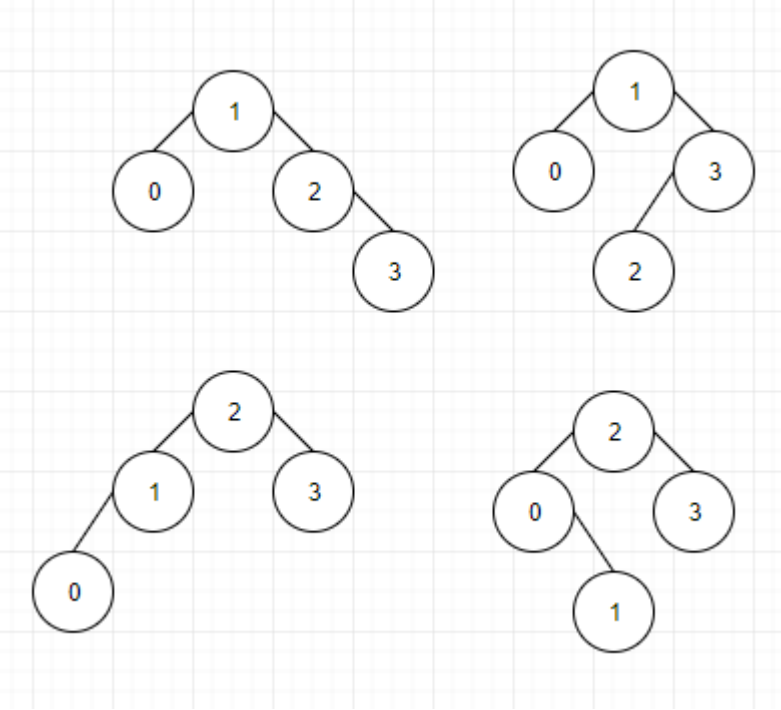
Using the Catalan number to solve this for N = 4.

Catalan Number CN = 2(n)!/(n + 1)! * n!

= 2(4)!/(4 + 1)! * 4!

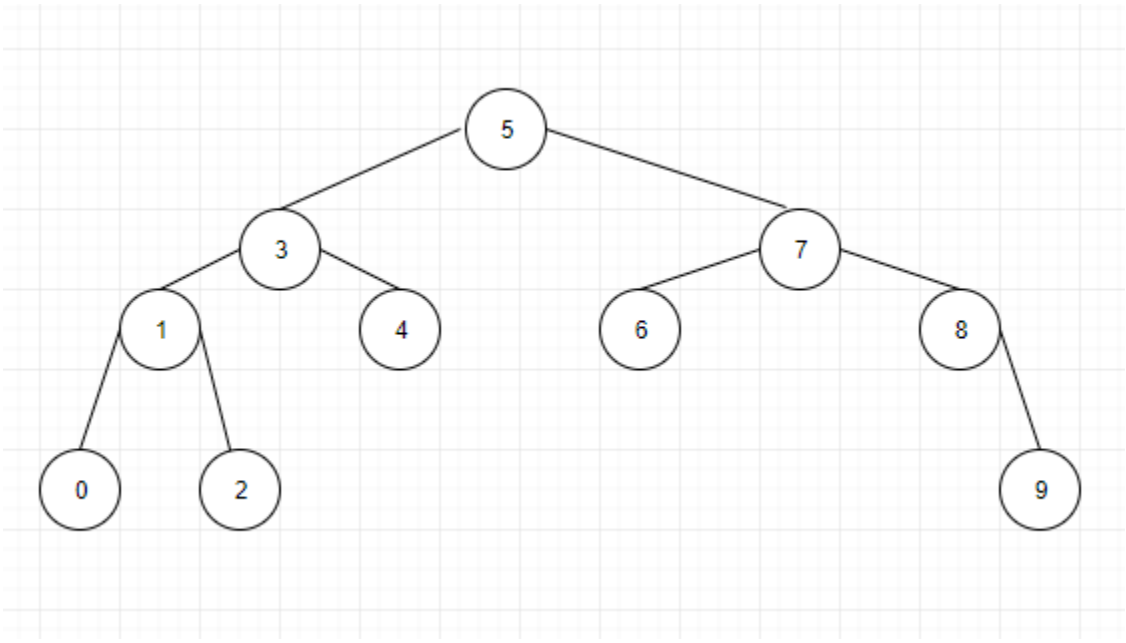
= 8! / 5! * 4!

= 14 different binary search trees that we can create



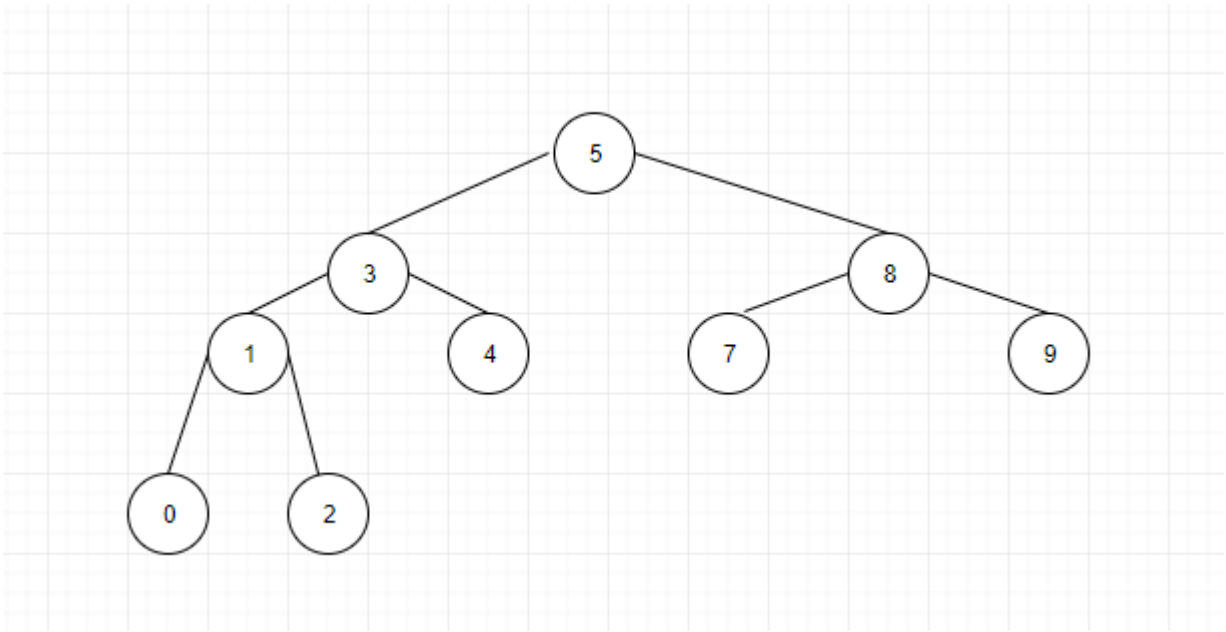
We can have 4 AVL trees.

B) If N = 10 then we have {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



The minimum height of this tree is 2 and the maximum height is 3. This is because there can only be a height difference of 1 at all times.

C:



Removing 6 will result in one rotation.

Question 2:

A) The time complexity of adding all the elements to the tree would be O(nlog(n))
B) The most efficient way to remove the elements from the tree in sorted order would be to perform an in-order traversal.
C) The time complexity of removing these elements would be O(n)
D) The total time complexity of add and removing all would be O(nlog(n)) + O(n) which makes it O(nlog(n))
E) The total memory complexity would be O(n)

Question 3:

A) Every time we use put, we will store the smallest key in a variable. And every time put is called, you will compare the key that is being inserted to your current smallest key. If it is smaller than you would need to update your variable and continue on. Essentially you would need to compare right before you put. For your remove, you would have to check if the key you want to remove is the smallest key. If it isn't then you would just remove and check for a balance. If it is the key, then you would need to set the variable to null, and remove the key, rebalance, and then look for the next smallest key, maybe using the in-order traversal.

B) I do not see the time complexity being reduced since performing an In-order traversal would yield the same result. If anything it would be about the same. You might even be performing an extra linear traversal in order to look for the smallest / bigger key. Which will still give O(nlog(n)) for the total time complexity

Question 4:

Balance:
Input: T – A specific node

T.balance ← Height(T.left) – Height(T.right)

Algorithm Height():
Input: T – A node of the AVL Tree
Output: Height of the total AVL tree

If T.left and T.right = NULL then
return 0

If T = Tree.root then
T.parent ← null

If T.balance(T.left) > 1
    return T.balance(T.right)
If T.balance(T.right) < -1
    return T.balance(T.left)

If T.balance = 0 then
return T.left
else
return T.right

The big-O complexity is going to be O(logn). This is because we will only be working with half the data most of the time. Going through and looking for whichever one is going to be the 'deepest' height and since it's a binary tree will result in O(logn) as we will only be looking at 1 child, meaning we are splitting up our data.

Question 5:
Result:

A)

Index	Key	Key: Chaining	Key: Chaining V3
0	195	91	
1			
2			
3	16	94	81
4	147		
5	265		
6	32	162	
7	189	202	
8	21		
9	48		
10	75		
11	180	37	
12	207	77	

B) The maximum number of collisions caused by the above insertions is 7.

Question 6:

Index	Key	Key: Chaining	Key: Chaining V3
0	195	180	75
1	16	91	
2	32	77	
3	48		
4	94		
5			
6	21	81	
7	202		
8			
9	189		
10	265		
11			
12	147	207	162
13			
14			

Yes this proposal does have some validity to it. BUT it would be best to use a prime number when you are calculating the key/hashing. Which might mean that you should modify the Hash Table to a size closer to your prime as well. This is why there are more collisions in this scenario compared to question 5.

Question 7:

Index	Key
0	19
1	1
2	
3	
4	42
5	
6	
7	48
8	27
9	9
10	48
11	
12	
13	
14	
15	72
16	
17	
18	18

The size of the longest cluster caused by the above insertions is 4.

There was only one collision for 48. This is assuming that we are not replacing the keys.

The current value of the table's load factor is: 9/19

Question 8:

Index	Key
0	95
1	19
2	
3	
4	
5	
6	
7	
8	
9	
10	AVAILABLE
11	30
12	12
13	32
14	AVAILABLE
15	72
16	
17	
18	

The size of the longest cluster is 3. The complexity of the above operations is O(n). This is the worse case because of the linear probing.

The number of collisions that occurred was 1.