COMP 6481

Assignment 3: Theory Part

Amro Elbahrawy

40221760

Question 1)

So the only way for duplicates to happen in a binary search tree is either an internal node is equal to its parent, or an internal node is equal to its sibling.

The algorithm would be:

1)Start at the root node and mark it as visited.

2)Check if its left child is equal to the parent node, return the value if they are.

3)If not, then mark the left child as visited and check its sibling (the right child of the parent node) if it's equal to it. Return the value if they are.

4)Repeat steps 1-3 recursively.

If we reach the rightmost leaf of the BST, then it has no duplicates.

The algorithm would run at O(n) time since we only need to traverse the entire tree once.

Question 2)

i) In terms of space complexity, it should be equivalent to the AVAILABLE method. It is faster when it comes to time complexity though, because the negative value indicates that the key is not in the hash table and immediately terminates instead of linearly probing and looking for other keys like it would in the AVAILABLE method.

A behavioral sproblem that might occur is that if multiple values have duplicate keys, and the one that was inserted first was removed. So when we search for key(16) and the first entry of the hash table shows -16, we terminate when there exists a value in the hash table with the key 16.

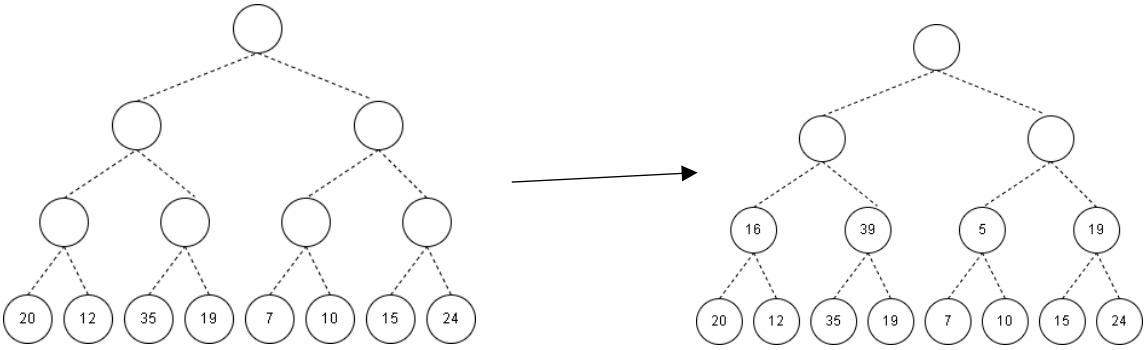
ii) This approach should be better in terms of space complexity, since we don't have any AVAILABLE entries and we sort the entries accordingly whenever we remove a value from a hash entry.

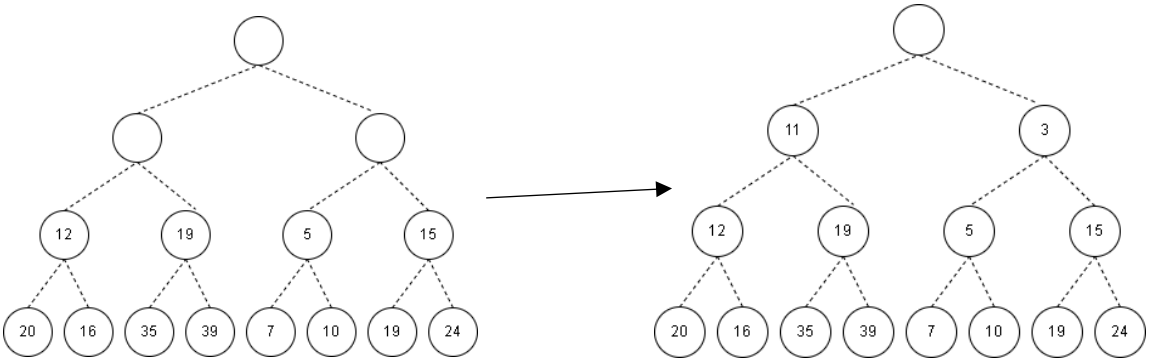
The problem with this approach will be much more time demanding because we traverse the entire hash table after removing an entry to check if there exists another entry that should have been there. If there are no entries, then we will traverse the entire hash table and return nothing [Complexity O(n) assuming hash table of size n]. Additionally, if we do find an element that should have been at another place, the entry of that element will now be vacant, and we will have to search if another element should have been at that entry.

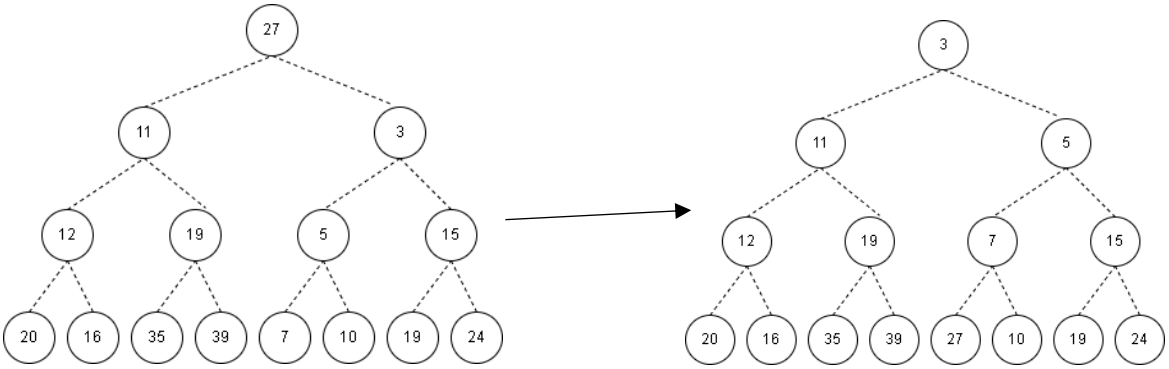
Question 3)

Part (i) Height of the tree will be log(n) +1 = log(15) +1 = 2

Step 1:

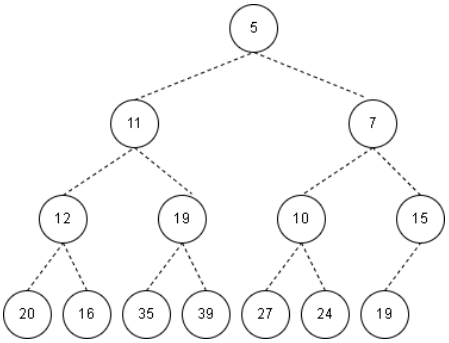




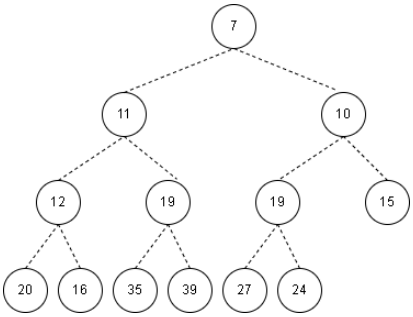


Applying RemoveMin on the final tree 6 times:

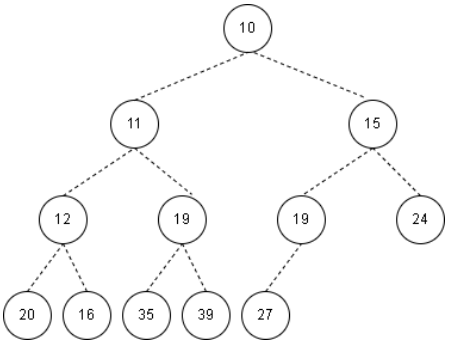
1st Removal (Remove 3):



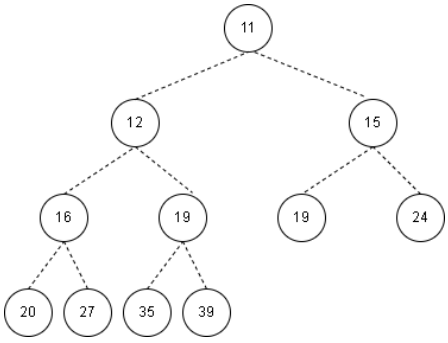
Remove 19:



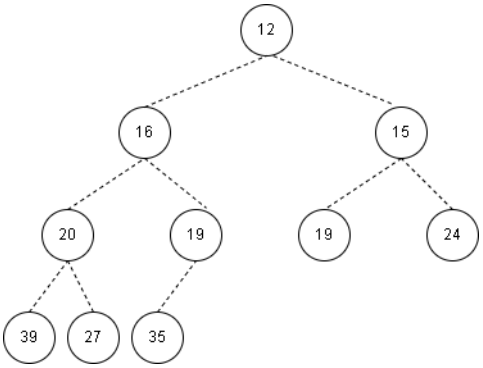
Remove 24:



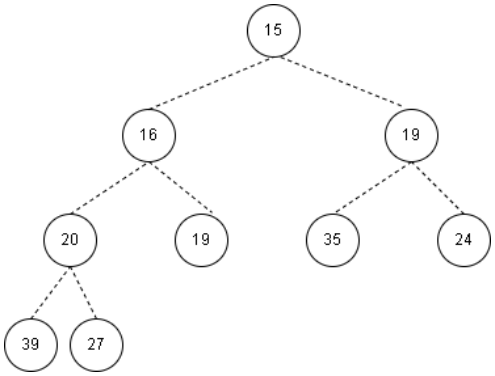
Remove 27:



Remove 39:



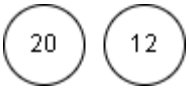
Remove 35:



Question 3 Part (ii)

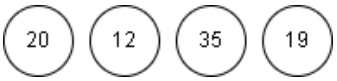
Insert(20)

Insert(12)



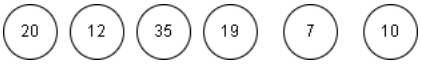
Insert(35)

Insert(19)



Insert(7)

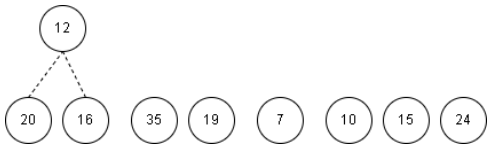
Insert(10)



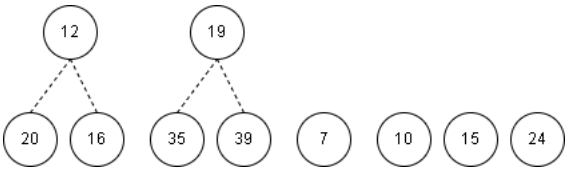
Insert(15)

Insert(24)

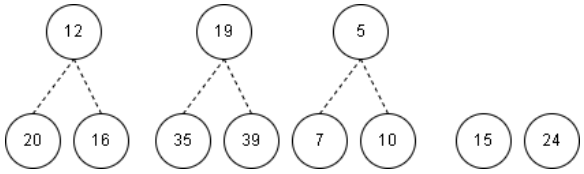
Insert(16)



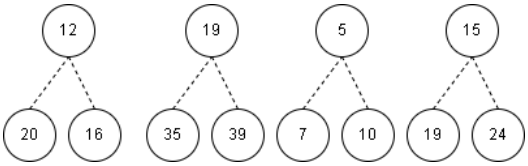
Insert(39)



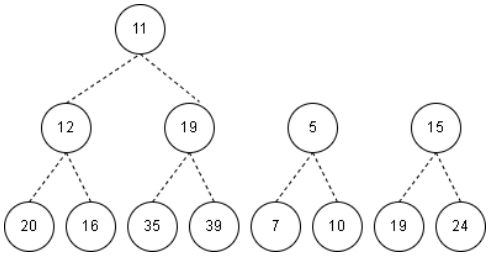
Insert(5)



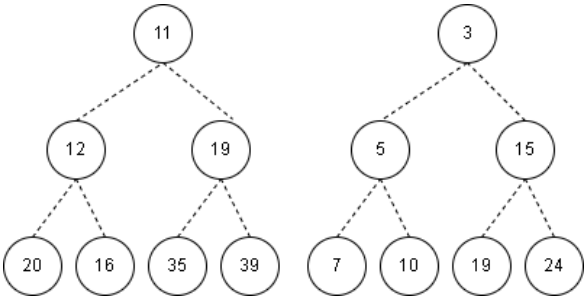
Insert(19)



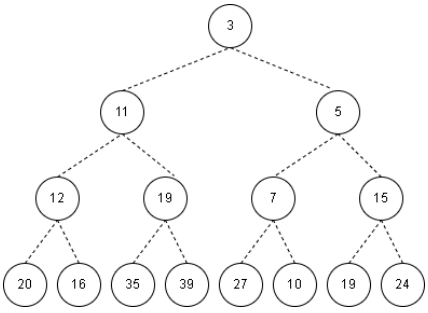
Insert(11)



Insert(3)



Insert(27)



Question 4)

a)

182, 91, 78	261		16		70	97, 58	202, 33	21	243,217	36, 166	180	51
0	1	2	3	4	5	6	7	8	9	10	11	12

b) 6 insertions.

Question 5)

(The red numbers are the ones that got swapped in their respective iterations)

The green numbers are the ones that are sorted

i))

Iteration 1: **23 32 72 22 73 40 30 20 60 16 74 28 14 76**	**Iteration 2:** **23 32 22 72 40 30 20 60 16 73 28 14 74 76**
Iteration 3: **23 22 32 40 30 20 60 16 73 28 14 72 74 76**	**Iteration 4:** **22 23 32 30 20 40 16 60 28 14 72 73 74 76**
Iteration 5: **22 23 30 20 32 16 40 28 14 60 72 73 74 76**	**Iteration 6:** **22 23 20 30 16 32 28 14 40 60 72 73 74 76**
Iteration 7: **22 20 23 16 30 28 14 32 40 60 72 73 74 76**	**Iteration 8:** **20 22 16 23 28 14 30 32 40 60 72 73 74 76**
Iteration 9: **20 16 22 23 14 28 30 32 40 60 72 73 74 76**	**Iteration 10:** **16 20 22 14 23 28 30 32 40 60 72 73 74 76**
Iteration 11: **16 20 14 22 23 28 30 32 40 60 72 73 74 76**	**Iteration 12:** **16 14 20 22 23 28 30 32 40 60 72 73 74 76**
Iteration 13: **14 16 20 22 23 28 30 32 40 60 72 73 74 76**	

ii)) I think both algorithms will have the same time complexity in the worst case, where the array is sorted in descending order. Because both algorithms behave similarly where we look for the smallest element and then swap them. Both algorithms should have O(n^2)

Question 6)

Iteration 1: 4.9.1 -- 6,9,2 -- 9,9,2 -- 4,7,2 -- 1,8,2 -- 5,4,3 -- 7,8,3 -- 2,6,4 -- 2,9,5 — 3,5,6 — 9,4,7
Iteration 2: 5,4,3 – 9,4,7 – 3,5,6 – 2,6,4 – 4,7,2 – 1,8,2 – 7,8,3 – 4,9,1 – 6,9,2 – 9,9,2 – 2,9,5,

Iteraiton 3:
1,8,2 – 2,6,4 – 2,9,5 -- 3,5,6 – 4,7,2 – 4,9,1 – 5,4,3 – 6,9,2 – 7,8,3 – 9,4,7 – 9,9,2

Question 7:) N = 19, q = 7

Hash Function 1: k mod N

Hash Function 2: q-k mod q

K	H(k)	D(k)	Probes
45	**7**	**4**	**7**
25	**6**	**3**	**6**
12	**12**	**2**	**12**
61	**4**	**2**	**4**
38	**2**	**4**	**2**
88	**12**	**3**	**12 + 3**
39	**1**	**3**	**1**
18	**18**	**3**	**18**
29	**10**	**6**	**10**
29	**10**	**6**	**10 + 6**
35	**16**	**7**	**16 + 7**

Put(45)

							45											
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(25)

						25	45											
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(12)

						25	45					12						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(61)

				61		25	45					12						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(38)

		38		61		25	45					12						
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(88)

		38		61		25	45					12			88			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Remove(12)

		38		61		25	45					X			88			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(39)

	39	38		61		25	45					X			88			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Remove(61)

	39	38		X		25	45					X			88			
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(18)

	39	38		X		25	45					X			88			18
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(29)

	39	38		X		25	45			29		X			88			18
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(29)

	39	38		X		25	45			29		X			88	29		18
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Put(35)

	39	38		39		25	45			29		X			88	29		18
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

i) Size of the longest cluster: 2

ii) 3 collisions happened when inserting the values 88, 29 and 35

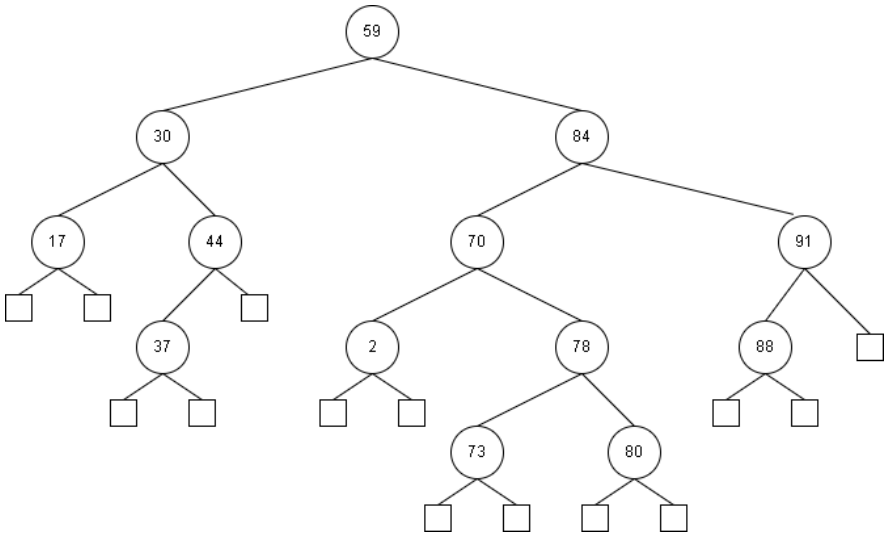
iii) The load factor is 9/19 = 0.4737

Question 8)

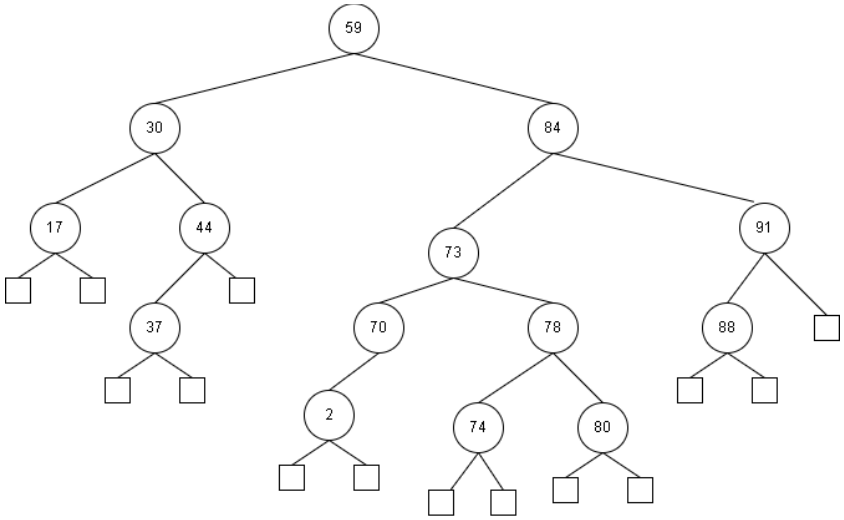
The AVL tree is unbalanced. The balance of the root node is 2 (left-heavy).

To fix this, a left-right shift is done for the subtree whose root node is 80.

Resulting balanced AVL tree:

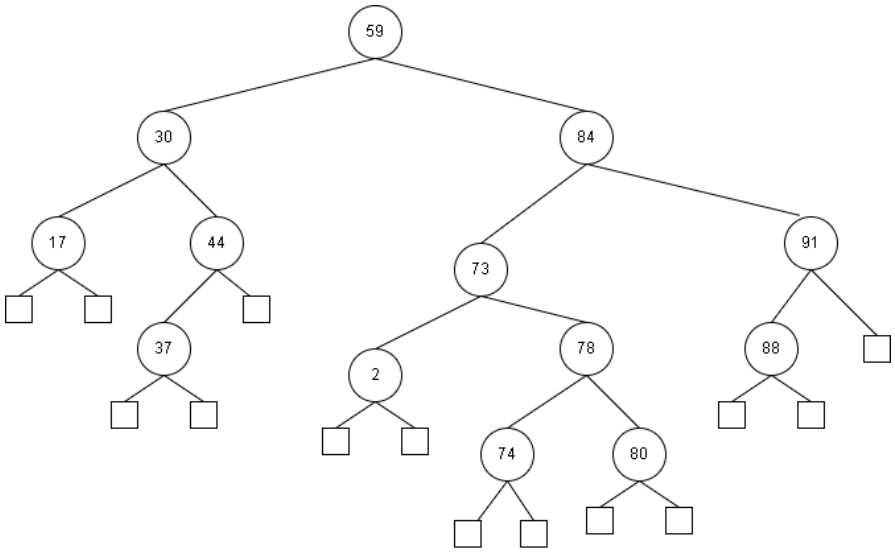


ii) Put(74)



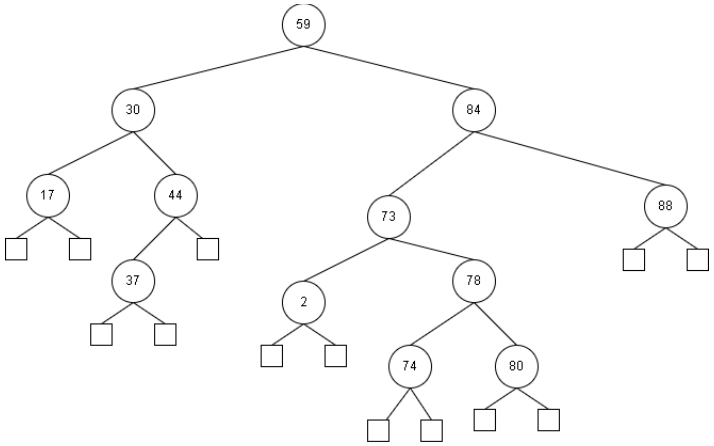
The complexity of this operation is equal to the height of the tree = O(logn)

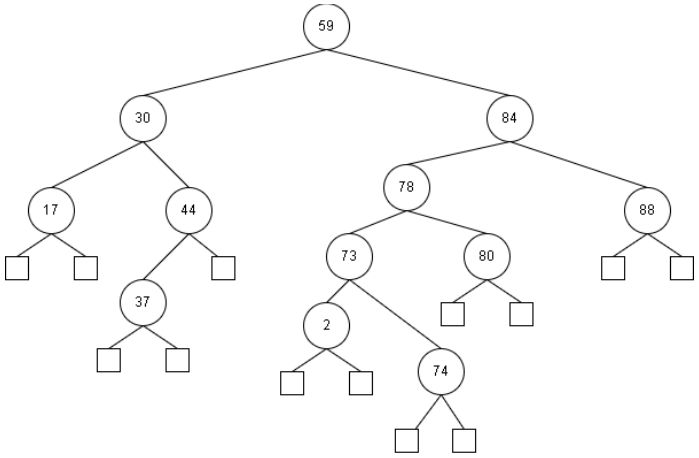
iii)

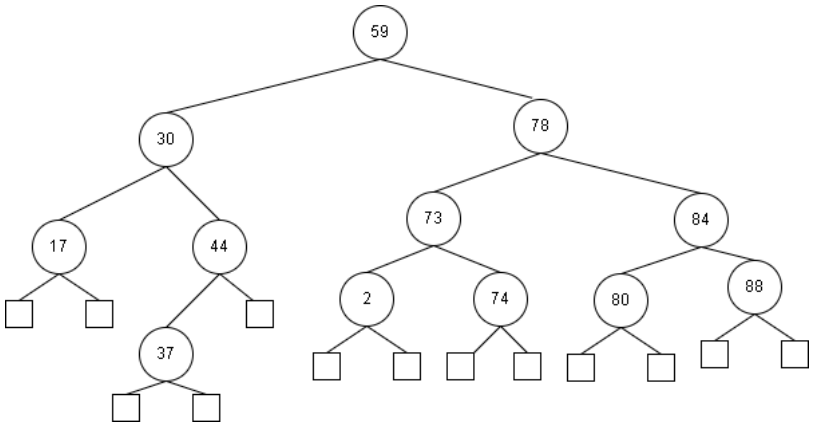


The complexity of removing 70 is O(logn)

iv)) Remove(91)







Time complexity is O(logn)

Question 9) Starting from Node N

Visited Nodes	Adjacent Nodes	Shortest Distance	Distance
[]	[A,W,D]	D	1
[N, D]	[A,W,M]	W	3
[N,D,W]	[A,M,T]	A	5
[N,D,W,A]	[M,T,E]	M	5
[N,D,W,A,M]	[T,E,K]	T	7
[N,D,W,A,M,T]	[E,K]	E	8
[N,D,W,A,M,T,E]	[K]	K	15
[N,D,W,A,M,T,E,K]			