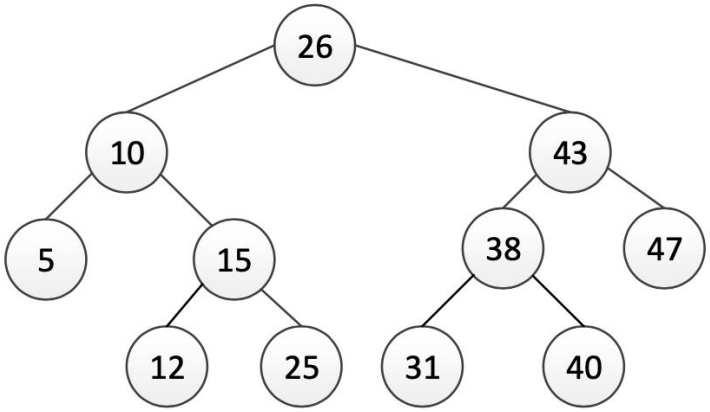
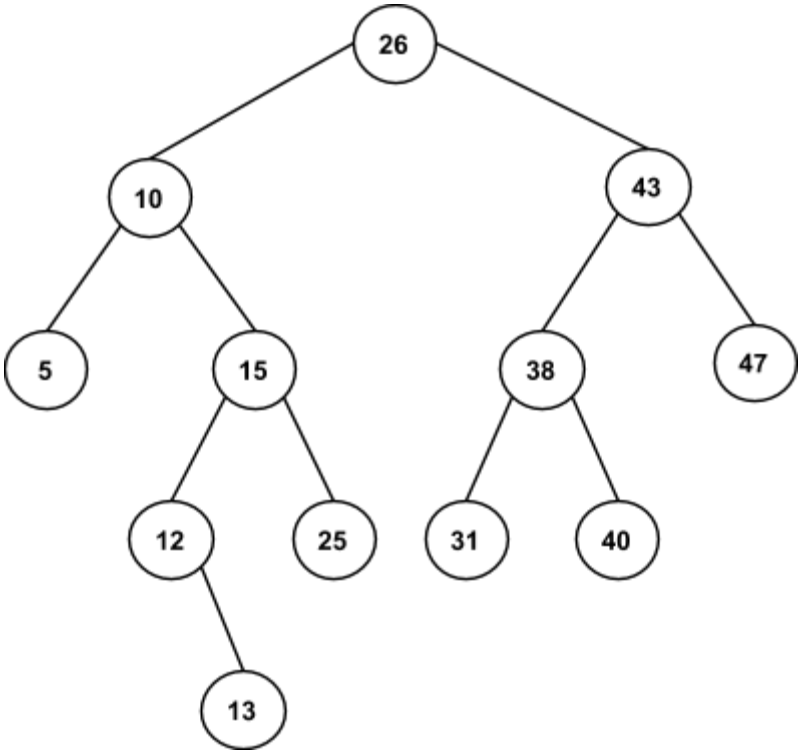
Problem 1

Consider the following binary search tree. Show the resulting tree if you add the entry with key=13 to the above tree. You need to describe, step by step, how the resulting tree is generated.



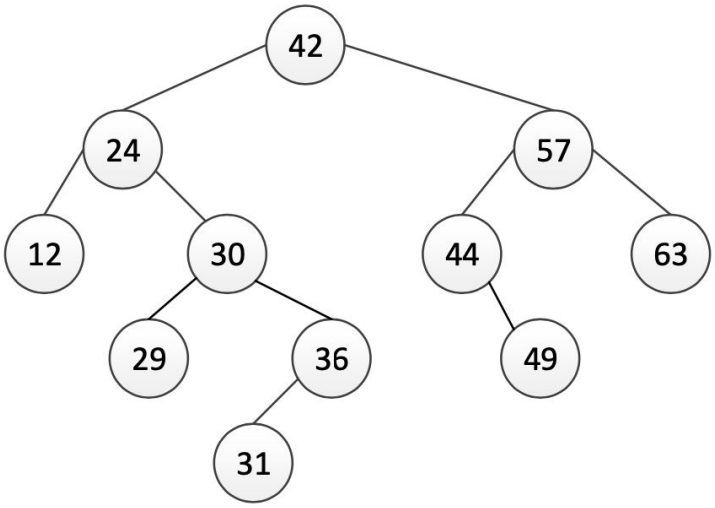
Answer:



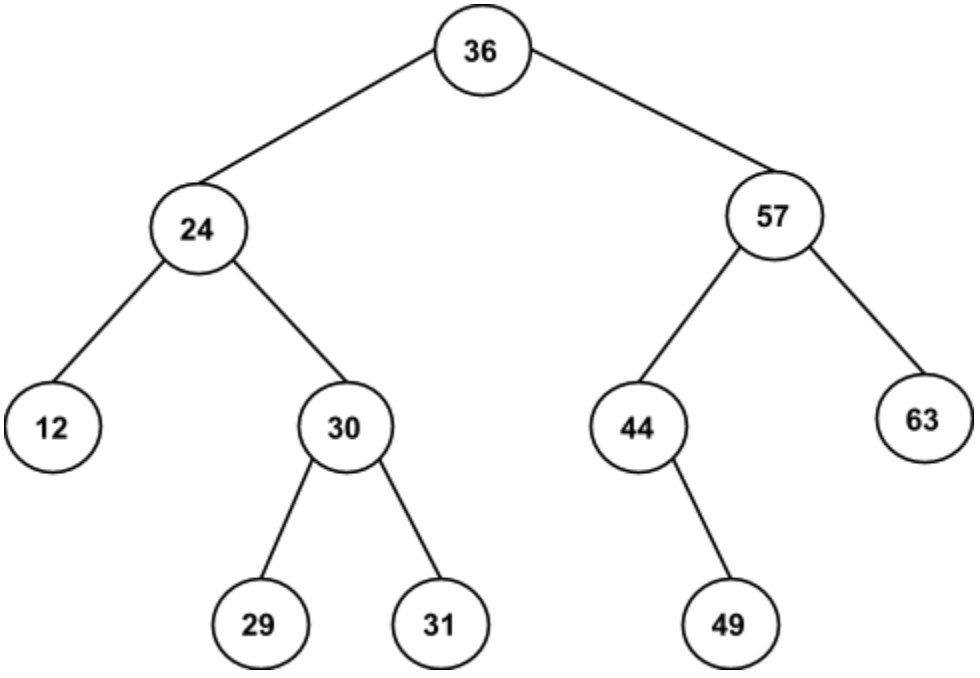
The resulting binary search tree after adding the key 13 was determined by starting at the root 26. Since 13 is smaller than 26 we went to the right child 10. 13 is larger than 10 so we go to the right child 15. 13 is smaller than 15 so we go to the right leaf node 12. The key 13 is then inserted to the right of 12 since it is larger than the parent 12: 26 → 10 → 15 → 12 → 13 .

Problem 2

Consider the following binary search tree. Show the resulting tree if you delete the entry with key=42 from the above tree. You need to describe, step by step, how the resulting tree is generated.



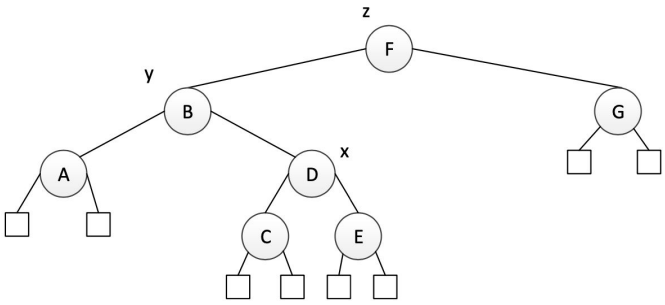
Answer:



To remove the key value 42, first we need to look for the largest leftmost key that is smaller than p (the value being removed). This key entry is 36 so we name that r. We remove 42 and then bring p=36 up to the root position. We replaced the only child of r, which was 31 in the empty spot left by r. So now the path of the tree reads: 36 → 24 → 30 → 31 .

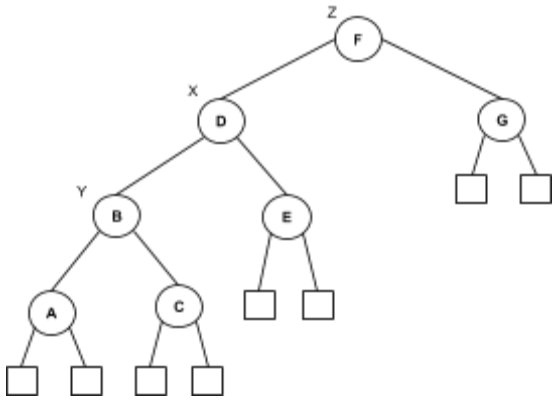
Problem 3

Consider the following AVL tree, which is unbalanced. Note that the nodes *F*, *B* and *D* are labeled *z*, *y* and *x*, respectively, following the notational convention used in the textbook. Apply a trinode restructuring on the tree and show the resulting, balanced tree.

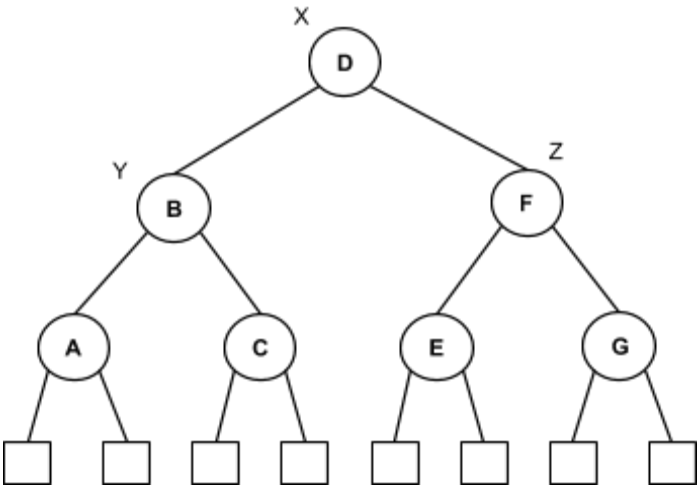


Answer:

Step1

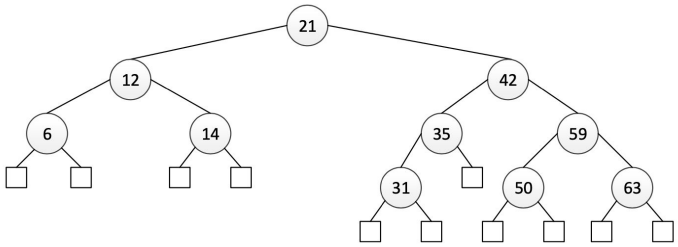


Step2



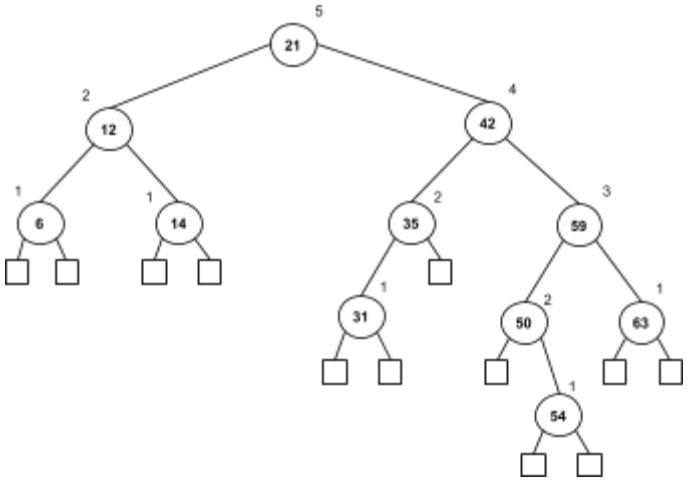
Problem 4

Consider the following AVL tree. Show the resulting, balanced tree after inserting an entry with key=54 to the above tree. You must describe, step by step, how the resulting tree is obtained.

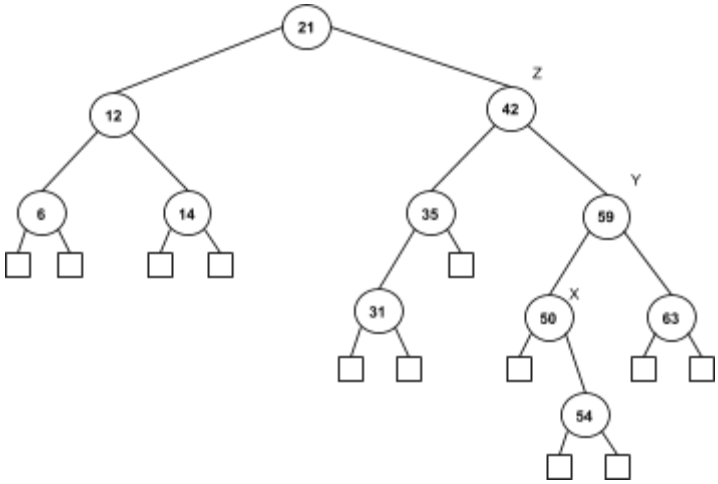


Answer:

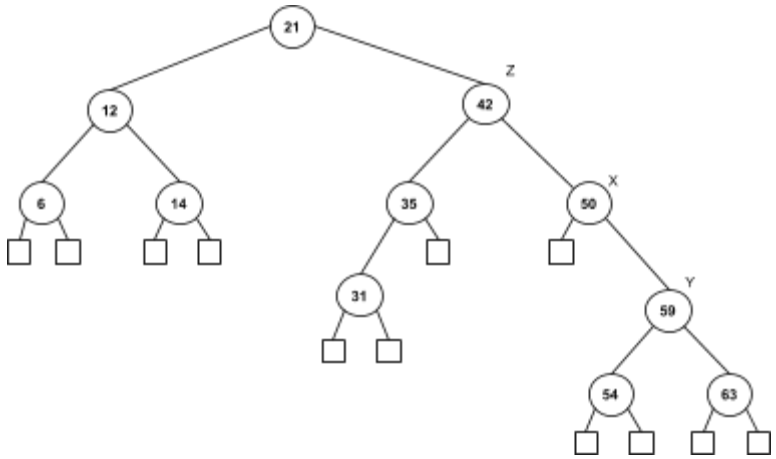
Step1



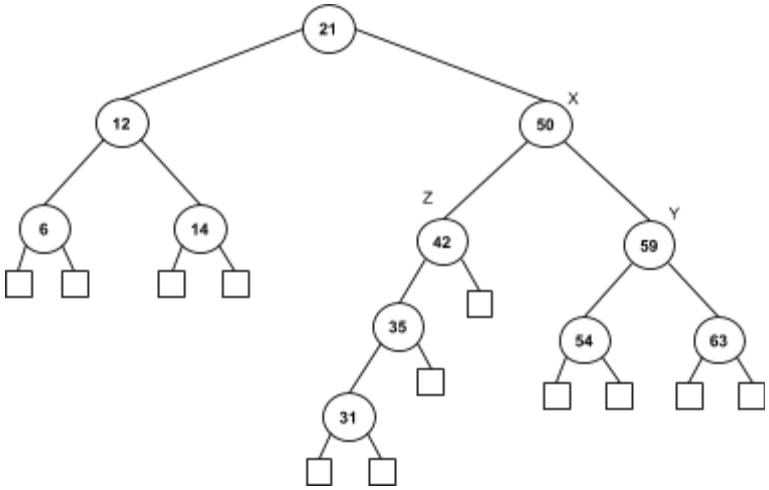
Step2



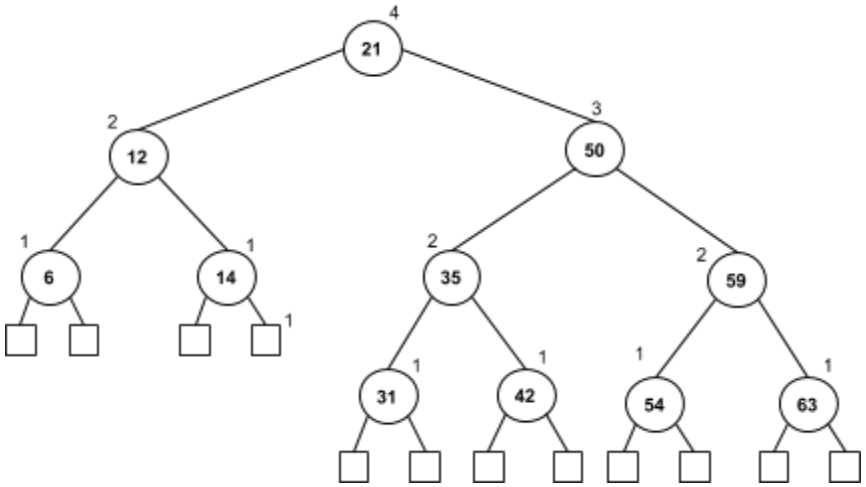
Step3



Step4



Step5

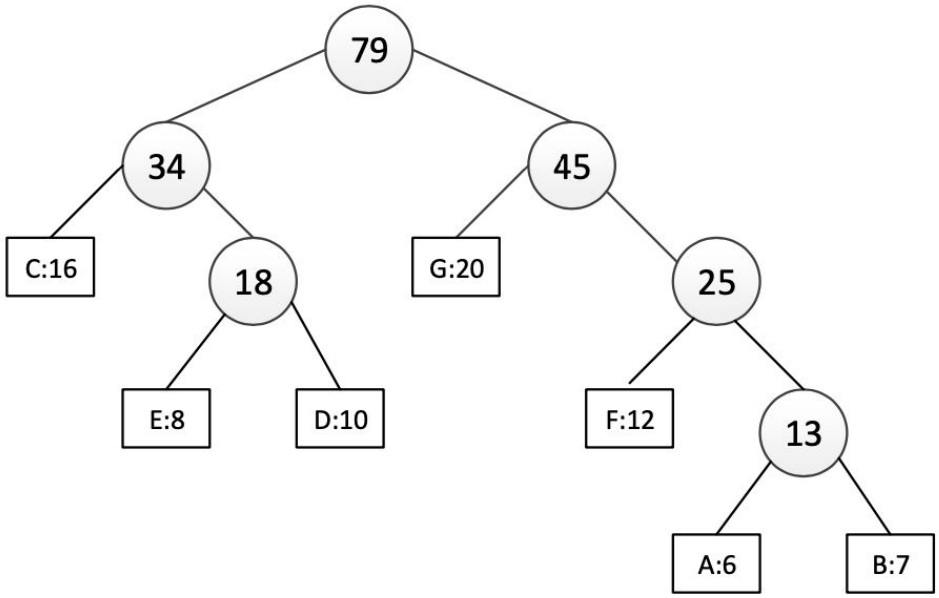


Initially this tree is balanced, and then after the insertion of key 54 it changes the height of the tree to those listed in step1. Moving up the tree to the root using the path with the greatest height: 54 → 50 → 59 → 42 → 21. The first node we get to where the difference of the height of the children is greater than 1 is node with key 42, so we label that node Z. The child of node 42 with the greater height is node 59, that is labelled Y. For node 59 the child node 50 has a greater height so we label 50 as X. This gives us the configuration to start the trinode rebalancing.

The trinode balance is performed in steps 3 and 4. At which point the tree is still not balanced so we perform a single rotation in step 5 to get the final balanced version of the tree. Now the tree is in balanced AVL form again: meaning that the difference between the height of each of the children on the same level is no greater than 1.

Problem 5

Consider the following Huffman tree:



Answer:

1. Encode the string “BDEGC” to a bit pattern using the Huffman tree

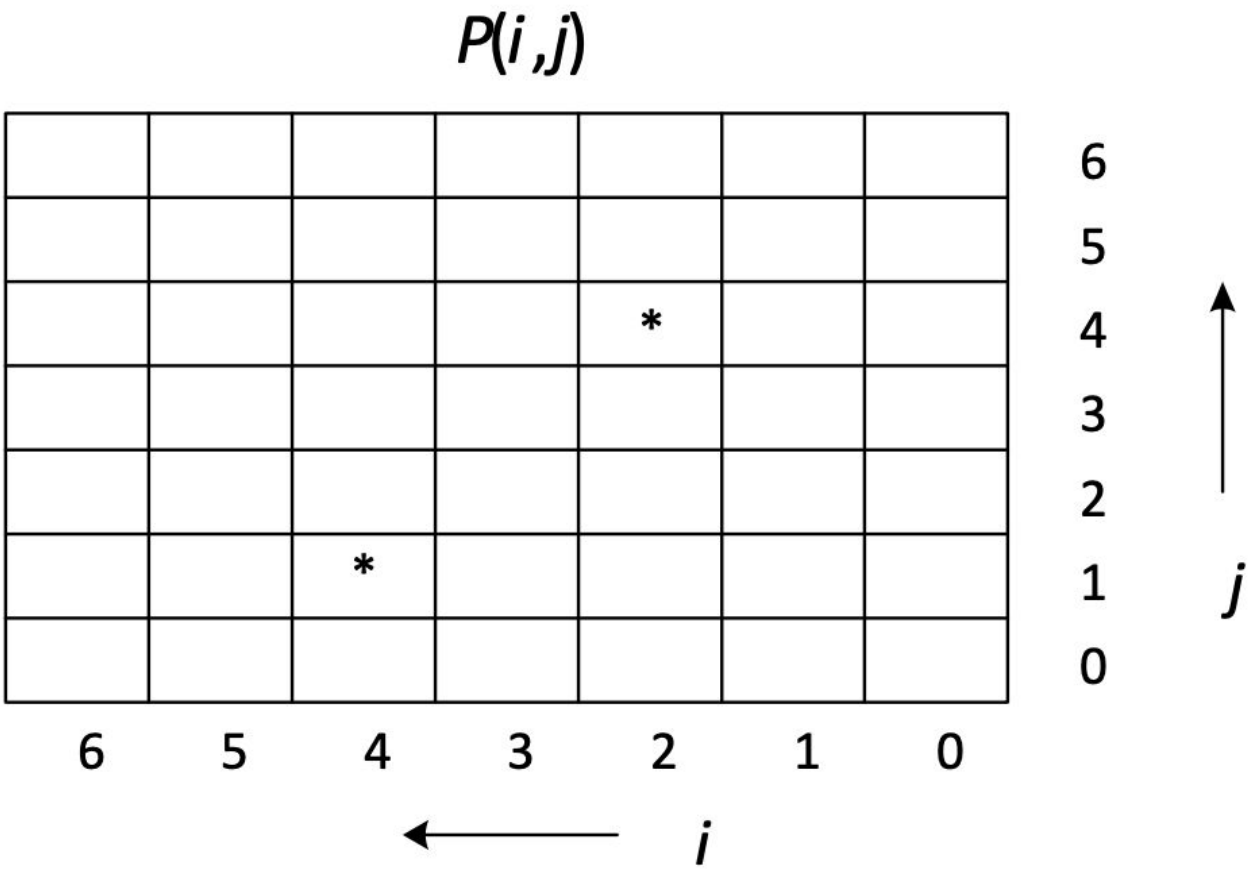
   11110110101000

2. Decode the bit pattern “010111010011”

   EAGD

Problem 6

This question is about the *World Series* problem that we discussed in the class. The following is the probability matrix for the problem. Calculate the probabilities of *P*(4, 1) and *P*(2, 4), which are marked with * .



Answer:

1. *P*(4, 1) = (P(3, 1) + P(4, 0)) / 2 = (⅛ + 0)/2 = 1/16

2. *P*(2, 4) = (P(1, 4) + P(2, 3)) / 2 = (15/16 + 11/16)/2 = 26/32

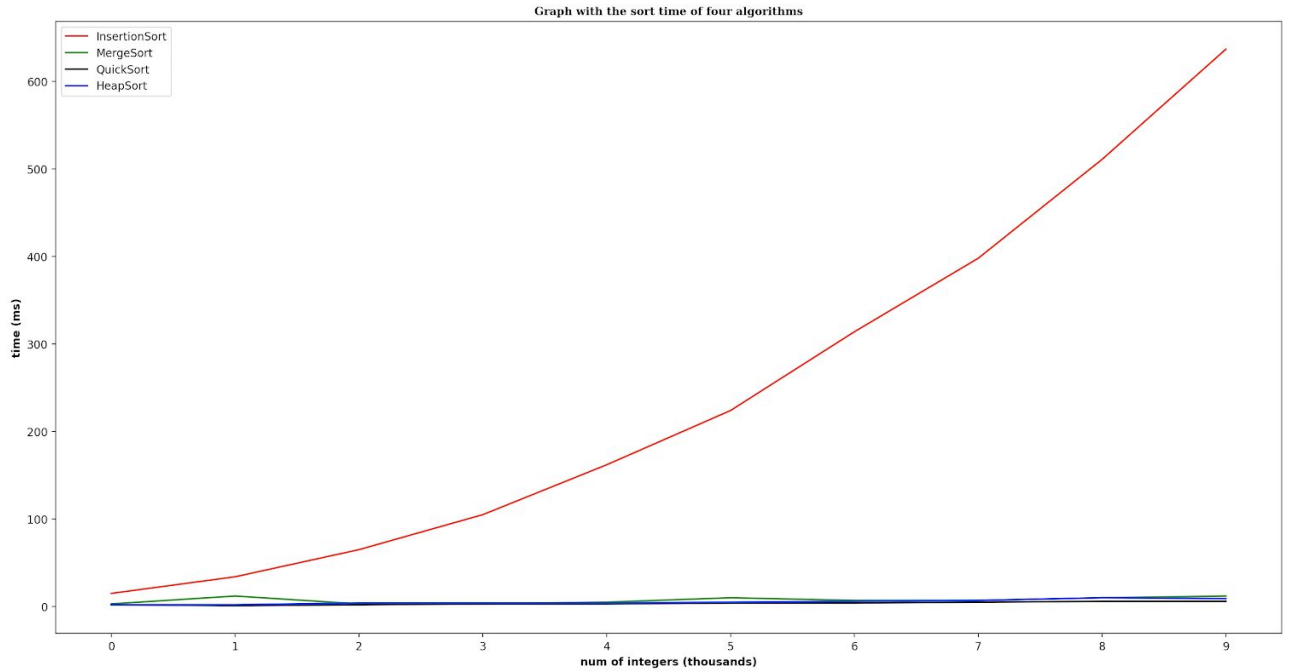
   P(1, 4) = (P(0, 4) + P(1, 3)) /2 = (1 + ⅞)/2 = 15/16

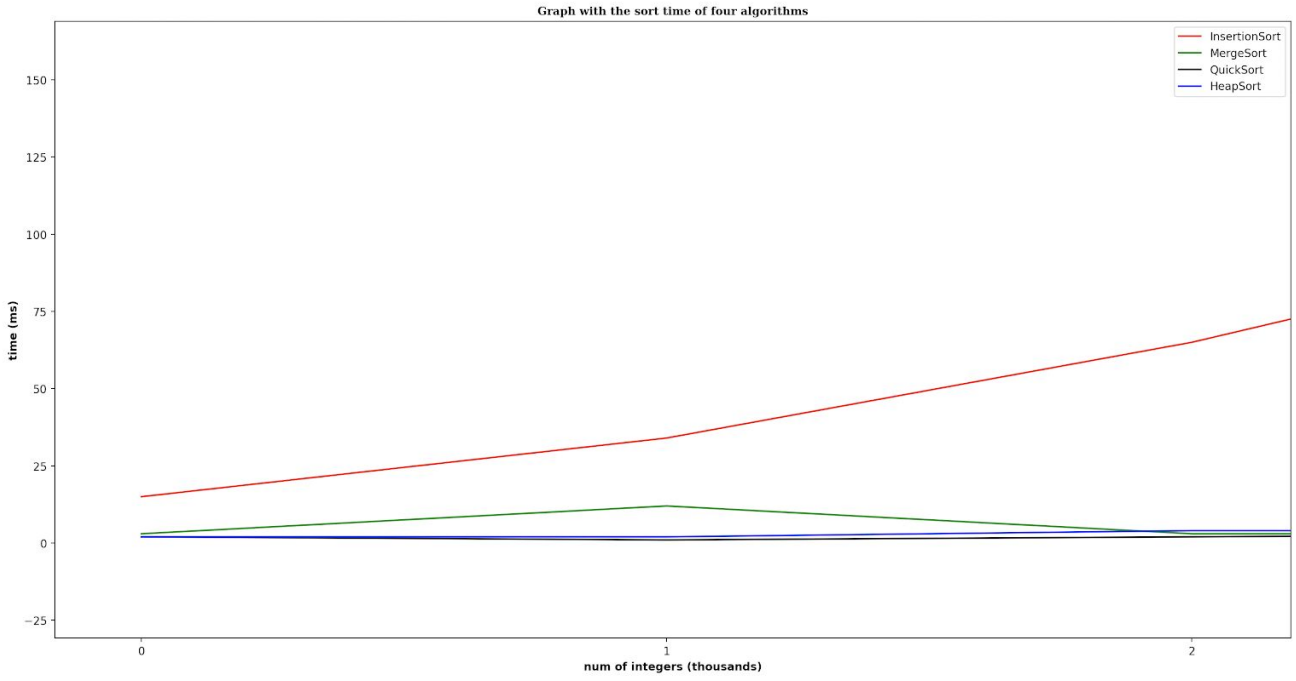
   P(2, 3) = (P(1,3) + P(2, 2)) /2 = (⅞ + ½)/2 = 11/16

Problem 7

Discussion/ observation of Problem 7: This part must include what you observed and learned from this experiment and it must be “substantive”.

Answer:





	Number of integers sorted									
Algorithm	10,000	20,000	30,000	40,000	50,000	60,000	70,000	80,000	90,000	100,000
insertion	15	34	65	105	162	224	314	398	511	637
merge	3	12	3	3	5	10	7	7	10	12
quick	2	1	2	3	3	4	4	5	6	6
heapsort	2	2	4	4	4	5	6	7	10	9

I thought heapsort would have been the faster algorithm by far, however the elapsed sorting time was comparable to quicksort and was not much faster than mergesort. Mergesort and quicksort were implemented as recursive methods. In comparison insertion sort which was not a recursive method took the longest time to sort though the integers and the elapsed time increased exponentially as the size of the array also increased.

As mentioned by the instructions the running time for all the algorithm does not increase or decrease linearly, excluding insertionsort. There was an increase in sorting time for mergesort moving from 10k integers to 20k integers but then another decrease in time to sort 30k integers. Not sure why this occurred, but there was another spike in sorting time for 60k integers; rounding out with 90k and 100k being the largest sorting time. For quicksort generally the sorting time increased slightly as the array size increased, but the increase in size was nominal. Lastly, heapsort had a very similar trend in sorting time to quicksort, with lower times for the smaller arrays with the time slightly increasing towards the larger arrays.