HW8

CH8

2. Balance the AVL tree in Figure 8-18. Show the balance factors in the result.

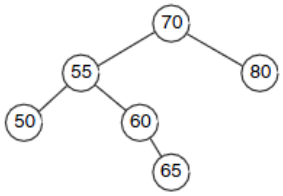

FIGURE 8-18 Figure for Exercise 2

4. Add 68 to the AVL tree in Figure 8-19. The result must be an AVL tree. Show the balance factors in the resulting tree.

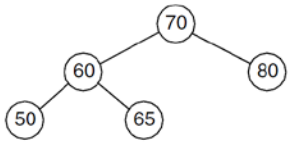

FIGURE 8-19 Figure for Exercises 3 and 4

10. Insert 44 and 50 into the tree created in Exercise 7.

Reference of EX.7:

7. Create an AVL tree using the following data entered as a sequential set. Show the balance factors in the resulting tree:

```
7 10 14 23 33 56 66 70 80
```

12. Delete the node containing 80 from the AVL tree in Figure 8-21.

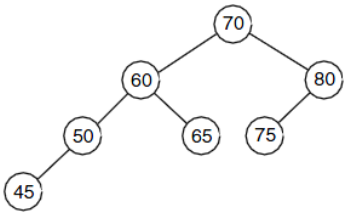

FIGURE 8-21 Figure for Exercises 12 and 13

14. Write an iterative version of Algorithm 8-1, "AVL Tree Insert."

ALGORITHM 8-1 AVL Tree Insert

```
Algorithm AVLInsert (root, newData)
Using recursion, insert a node into an AVL tree.
   Pre    root is pointer to first node in AVL tree/subtree
          newData is pointer to new node to be inserted
   Post   new node has been inserted
   Return root returned recursively up the tree
 1 if (subtree empty)
   Insert at root
   1  insert newData at root
   2  return root
 2 end if
 3 if (newData < root)
   1  AVLInsert (left subtree, newData)
   2  if (left subtree taller)
      1  leftBalance (root)
   3  end if
 4 else
   New data >= root data
   1  AVLInsert (right subtree, newPtr)
   2  if(right subtree taller)
      1  rightBalance (root)
   3  end if
 5 end if
 6 return root
end AVLInsert
```

CH9

2. Make a heap out of the following data read from the keyboard:

```
23  7  92  6  12  14  40  44  20  21
```

6. Apply the delete operation to the heap in Figure 9-21. Repair the heap after the deletion.

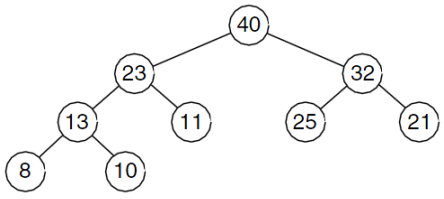

FIGURE 9-21 Heap for Exercises 5, 6, and 7

14. Show the resulting heap after 33, 22, and 8 are added to the following heap:

```
50  30  40  20  10  25  35  10  5
```

29. Our study of tree algorithmics has shown that most tree structures are quite efficient. Let's examine the efficiency of heaps. Modify the heap ADT developed in Section 9.3 to determine the complexity of building a heap. For this program measure efficiency as the number of data moves necessary to build the heap.

    To determine a pattern, run your program with arrays filled with random numbers. Use five different array sizes: 100, 200, 500, 1000, and 2000. Then analyze the heuristics developed in these runs and determine which big-O notation best applies. Prepare a short report of your findings with appropriate tables and graphs.

33. An airline company uses the formula shown below to determine the priority of passengers on the waiting list for overbooked flights.

$$\text{priority number} = A / 1000 + B - C$$

where

A is the customer's total mileage in the past year
B is the number of years in his or her frequent flier program
C is a sequence number representing the customer's arrival position when he or she booked the flight

Given a file of overbooked customers as shown in Table 9-2, write a program that reads the file and determines each customer's priority number. The program then builds a priority queue using the priority number and prints a list of waiting customers in priority sequence.

Name	Mileage	Years	Sequence
Bryan Devaux	53,000	5	1
Amanda Trapp	89,000	3	2
Baclan Nguyen	93,000	3	3
Sarah Gilley	17,000	1	4
Warren Rexroad	72,000	7	5
Jorge Gonzales	65,000	2	6
Paula Hung	34,000	3	7
Lou Mason	21,000	6	8
Steve Chu	42,000	4	9
Dave Lightfoot	63,000	3	10
Joanne Brown	33,000	2	11

TABLE 9-2 Data for Project 33