Homework 2 Solution

SPECIAL NOTICE: for binary search tree (including AVL and B-tree) insertion and deletion operations, the following answers are only recommended. The answers may not be unique since there are many ways to finish the operations, e.g. deleting an element in a binary search tree is not unique. Different books could also describe different algorithms for these operations. However, the basic idea remains the same.

3.2 (2);

$$(((9-2)+3)*(7-1))$$

$$60$$

3.3 (2);

$$(4-((5*6)/(7+8)))$$

$$2$$

3.5;

Advantage: dynamically change the size of the stack; no need to pre-assign the memory for the stack

Disadvantage: additional space is needed to store the pointers; implementation is more complicated than the continuous array based implementation

3.9;

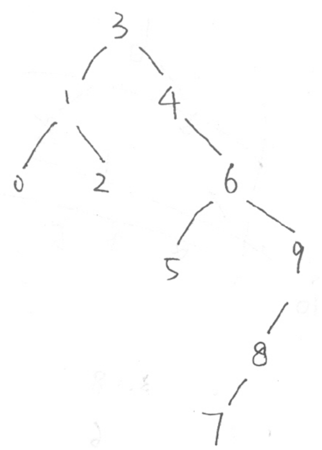
Prefix: -**ab+cde

Infix: (((a*b)*(c+d))-e)

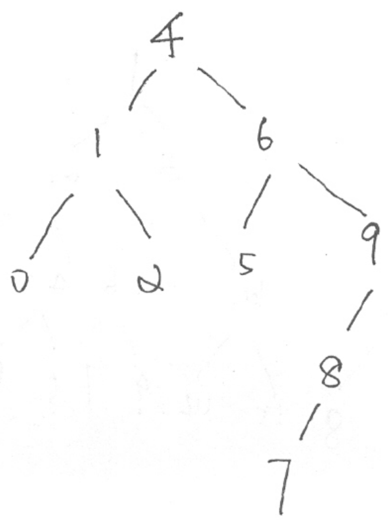
Postfix: ab*cd+*e-

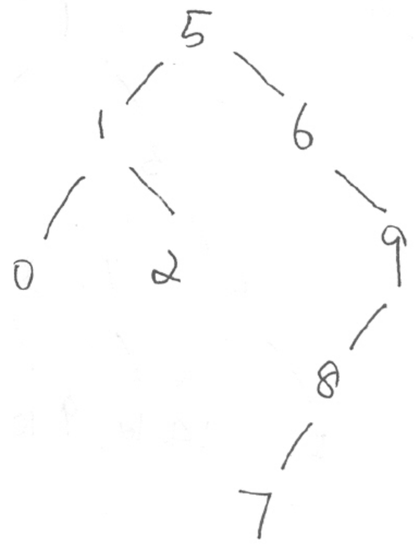
3.10;

(1) after insertion, the tree is :



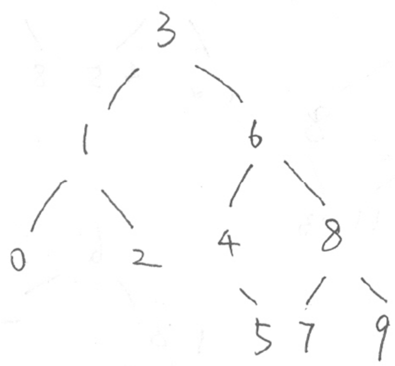
(2) after deleting the root twice, the trees are:



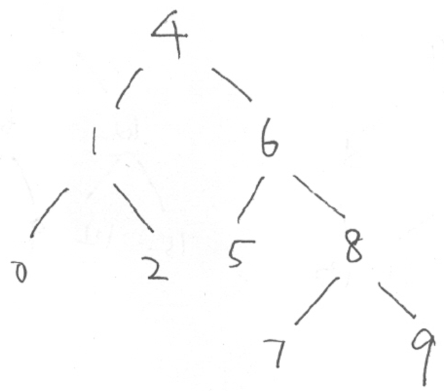


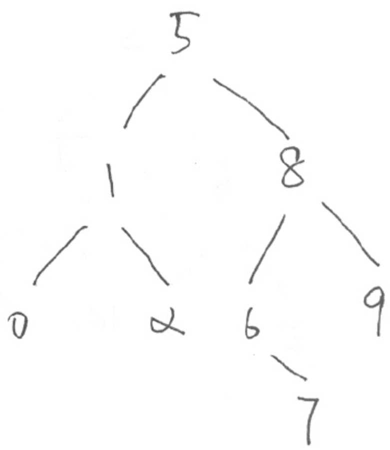
3.11;

(1) after insertion, the AVL tree is :

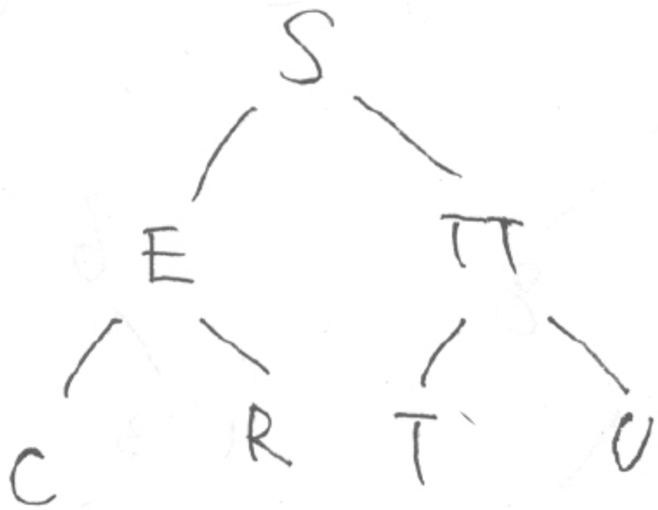


(2) after deleting the root twice, the trees are:



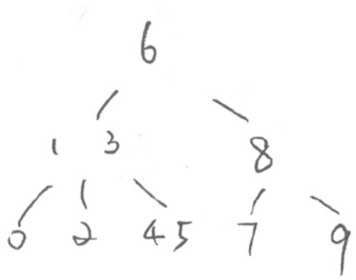


3.12 (2);

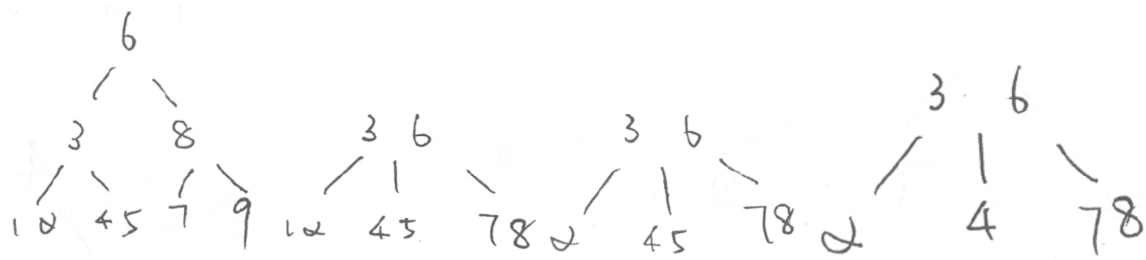


3.18;

(1) after insertion, the 2-3 tree is:



(2) after deleting 0, 9, 1 and 5, the tree are as follows:



4.1 (1) and (2);

open hash table:

0			
1	4371		
2			
3	1323	6173	
4	4344		
5			
6			
7			
8			
9	4199	9679	1989

closed hash table:

0	9679
1	4371
2	1989
3	1323
4	6173
5	4344
6	
7	
8	
9	4199

4.4;

(1) True

(2) False

(3) 3000,007 bits

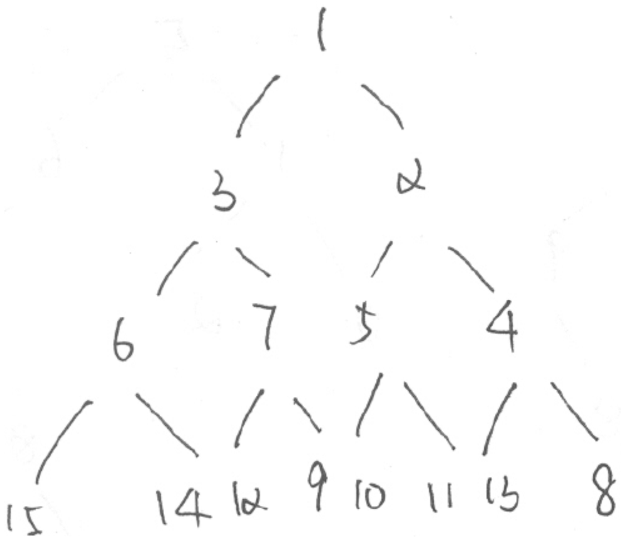
(4)(5) The problem is not clearly defined.

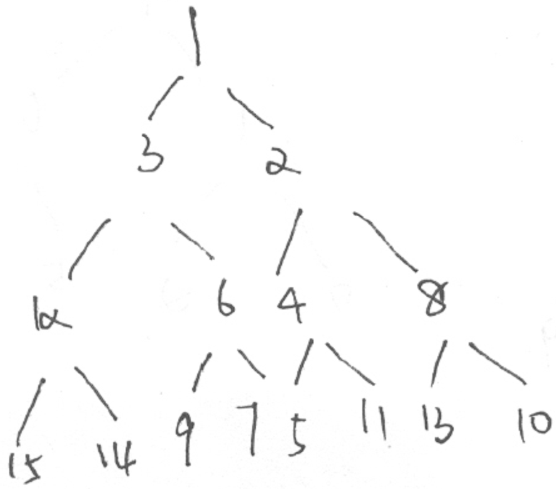
4.15;

$$1 + 2 + \cdots + N = O(N^2)$$

5.1;

The built trees are as follows, corresponding to (1) and (2):

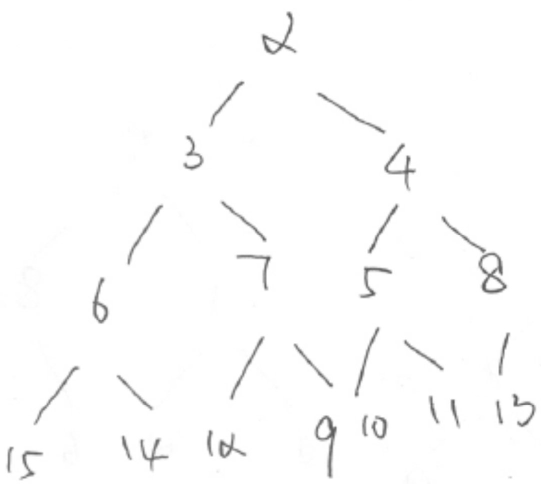


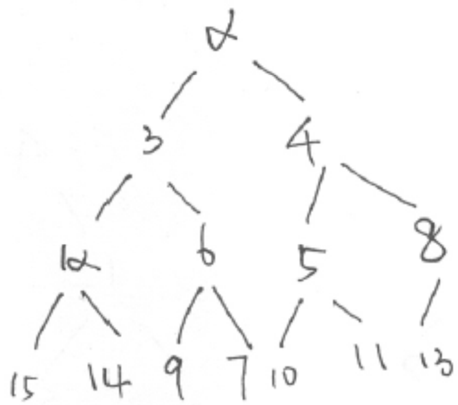


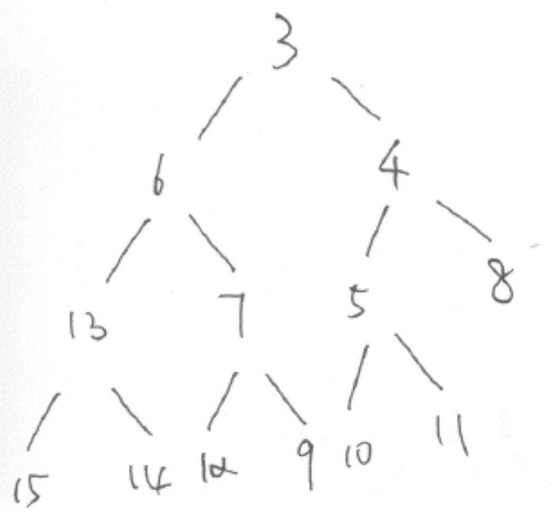
5.2;

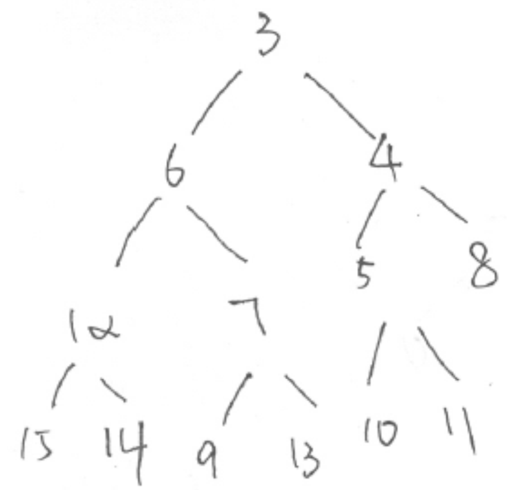
After deleting the minimal value for the above heaps three times, the heaps are listed in the following.

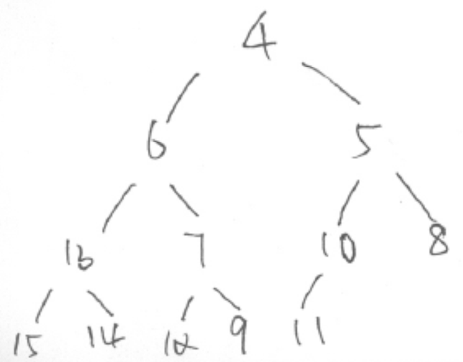
The left is for 5.1 (1); while the right is for 5.1(2).






3
6 4
13 7 5 8
15 14 12 9 10 11


3
6 4
12 7 5 8
15 14 9 13 10 11


4
6 5
13 7 10 8
15 14 12 9 11


4
6 5
12 7 10 8
15 14 9 13 11