COP 3502 – Final Exam Review Notes – Spring 2004

Comprehensive Portion

- Big-Oh notation.
- Summations and closed forms.
  - Summations that begin at other than 0 or 1.
  - Know the following:

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=0}^{n} i = \frac{n(n+1)}{2} \qquad \sum_{i=1}^{n} 1 = n \qquad \sum_{i=0}^{n} 1 = n+1$$

- Run-time approximations.
  - See Exam #1 review notes plus additional sample problems for binary numbers, summations, and run-time approximations.
  - See Exam #2 review notes for additional sample problems.
- Binary numbers.
  - Conversion of binary to decimal and decimal to binary.
  - Two's complement.
- Be able to trace through code which uses recursion and/or pointers.
- Be able to write a simple recursive function.
- Dynamic arrays.
- Searching and sorting algorithms
  - Know how insertion sort, bubble sort, merge sort and quick sort work.
  - Know how linear search and binary search work.
  - Know Big-Oh complexity for the various searching and sorting algorithms.
- Stacks (LIFO) – both static and dynamic implementations.
  - Applications such as converting infix expressions to postfix expressions.
- Queues (FIFO) – both static and dynamic implementations.
- Linked lists – be able to manipulate linked lists, insert, delete, traverse, etc.

New Material

- Binary trees.
  - Know structure properties.
  - Know traversal algorithms: preorder, inorder, and postorder.
- Binary search trees.
  - Know structure and ordering properties.
  - Be able to construct one from a set of data.
  - Be able to determine if a binary tree is a binary search tree.

    - Be able to insert a new node into a BST.
    - Be able to delete a node from a BST. Find logical predecessor and logical successor nodes in a BST.
- Balancing binary search trees.
    - Know left and right rotations.
    - Know the DSW algorithm and be able to balance a BST using this algorithm.
- AVL trees
    - Know how to define one.
    - Be able to determine balance factors.
    - Be able to insert a new node into an AVL tree and properly rebalance the tree. Left rotations, right rotations, and double rotations.

Specifics

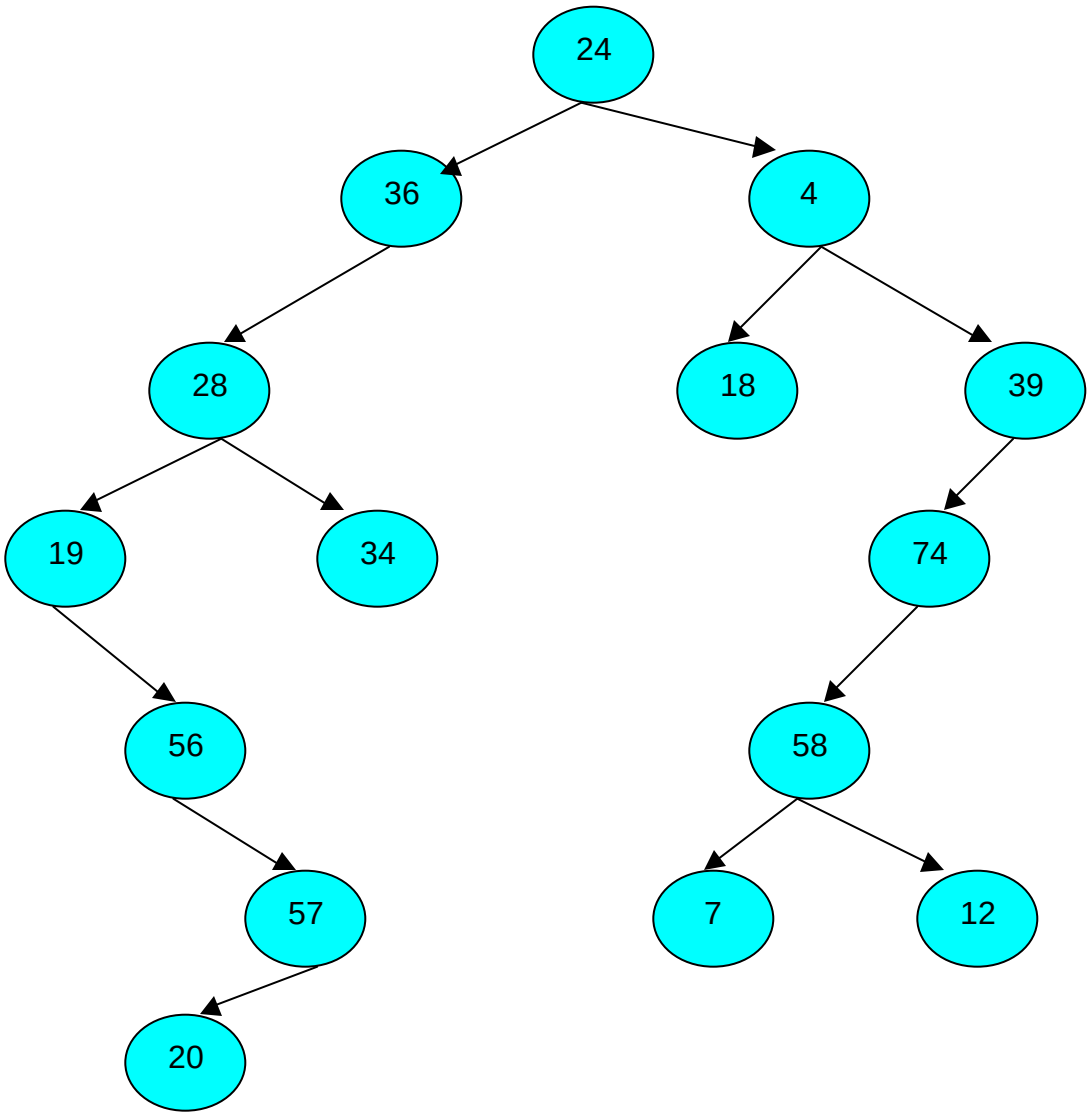
- Part of the test will be testing your ability to write functions and incorporate them into programs.
- Part of the test will be testing your ability to trace through programs or program segments, including programs with functions that use pass by value and/or pass by reference parameters.
- Material covered on the test comes from the lecture notes. No material will appear on the test which appears only in the textbook.

Review/Sample Questions

1. Given the following stack operations, show the contents of the stack after the last instruction has been executed.

    1. push(40)
    2. push(10)
    3. push(10)
    4. push(pop( ) + pop( ))
    5. push(20)
    6. push(5)
    7. push(pop( ) / pop( ))

2. For the binary tree shown below, produce preorder, inorder and postorder traversals of the tree.



3. Is the binary tree from question #2 a binary search tree?

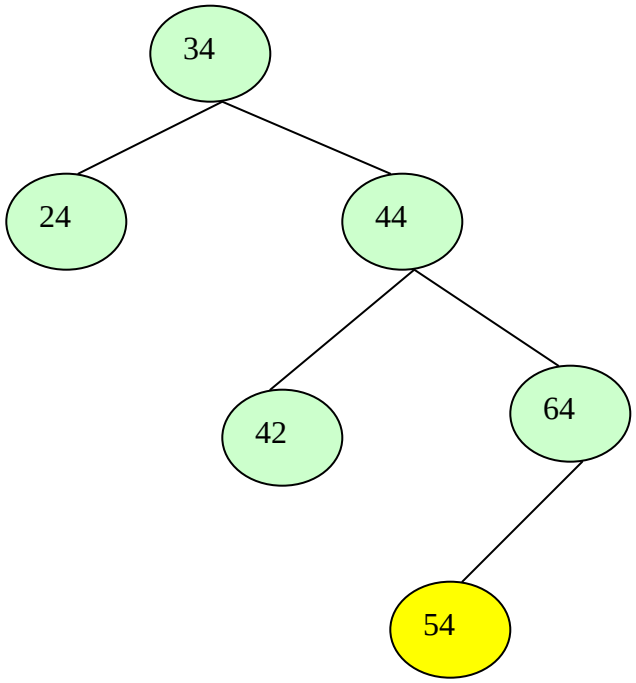
4. What is the value of $\sum_{i=14}^{26}(2i-3)$

5. Convert the following infix expression to postfix using a stack. Show the contents of the stack at the points marked A, B, C, and D in the infix expression.

$$(((7-3)+6)\times((4-2)\times 3))+((5\times 4)+(6-3))$$

A B C D

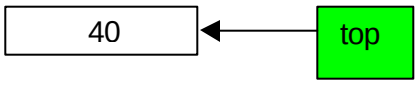
6. Consider the following tree:



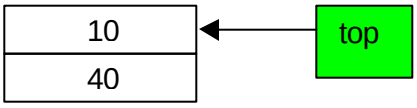
(a) Produce the balance factor for every node in the tree?
(b) Is the tree an AVL-tree according to the definition? Why or why not?
(c) Assume that the node with value 54 (the yellow node) was just inserted into the tree. Was the tree an AVL-tree before this insert occurred?
(d) Re-balance the tree that resulted after the insertion of the node with value 54 so that once again the tree is an AVL-tree.

Answers To Review Questions

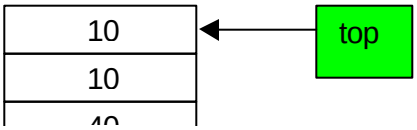
1. Stack problem.
   1. push(40)



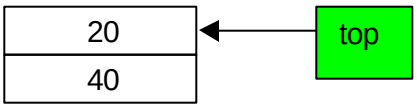
   2. push(10)



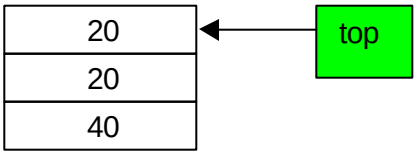
   3. push(10)



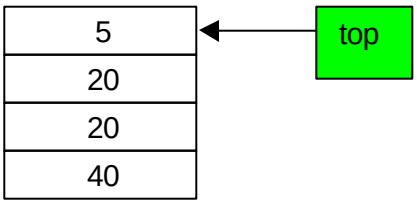
4. push(pop( ) + pop( ))



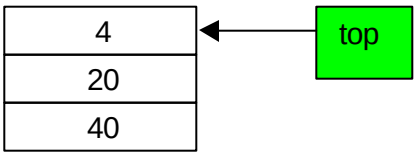
5. push(20)



6. push(5)



7. push(pop( ) / pop( ))



2. Preorder traversal:

24 36 28 19 56 57 20 4 18 39 74 58 7 12

Inorder traversal:

19 56 20 57 28 34 36 24 18 4 7 58 12 74 39

Postorder traversal:

20 57 56 19 34 28 36 18 7 12 58 74 39 4 24

3. No! For instance, 36 is in left subtree of 24 (root) and 4 is in right subtree.

4. $\sum_{i=14}^{26}(2i-3) = \sum_{i=14}^{26} 2i - \sum_{i=14}^{26} 3 = \left(2\sum_{i=1}^{16} i - 2\sum_{i=1}^{13} i\right) - \left(3\sum_{i=1}^{26} 1 - 3\sum_{i=1}^{13} 1\right)$

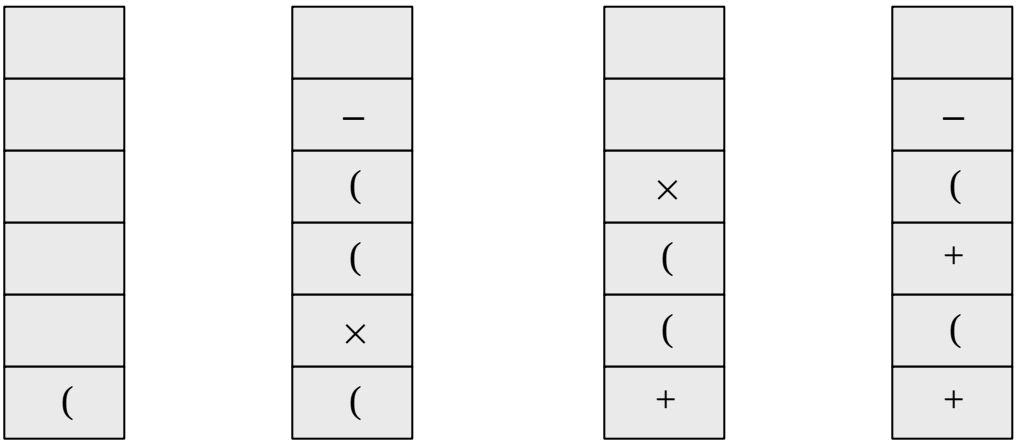
$$= 2\sum_{i=1}^{26} i - 2\sum_{i=1}^{13} i - 3\sum_{i=1}^{26} 1 + 3\sum_{i=1}^{13} 1 = \frac{2(26)(27)}{2} - \frac{2(13)(14)}{2} - 3(26) + 3(13)$$

$$= 702 - 182 - 78 + 39 \;=\; 520 - 78 + 39 \;= 481$$

5. Convert the following infix expression to postfix using a stack. Show the contents of the stack at the points marked A, B, C, and D in the infix expression.

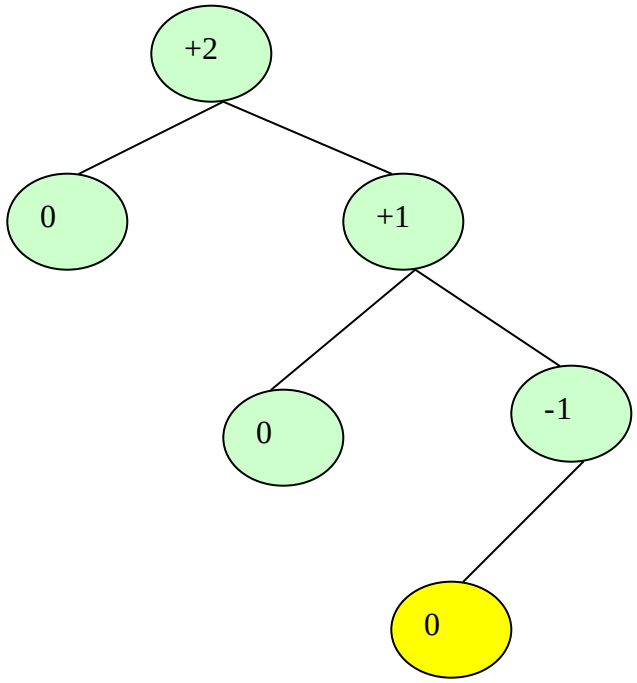
$$(((7-3)+6)\times((4-2)\times 3))+((5\times 4)+(6-3))$$

A B C D



Postfix expression is: 7 3 – 6 + 4 2 – 3 × × 5 4 × 6 3 – + +

6. (a) Same tree showing balance factors instead of data values for each node.

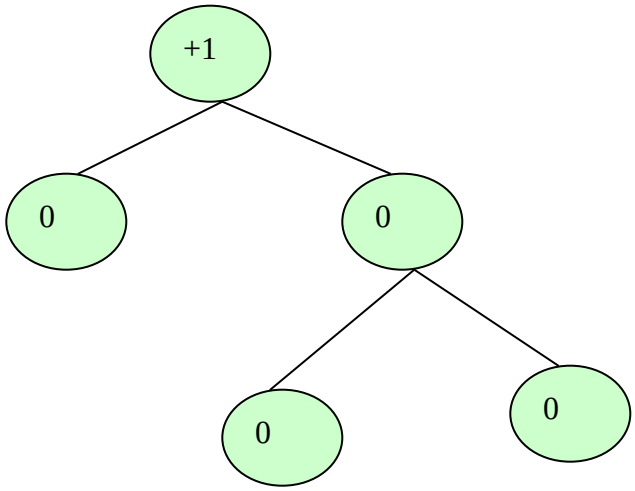


(b) Is the tree an AVL-tree according to the definition? Why or why not?

No, the root node has a balance factor of +2 and an AVL tree must have all nodes with balance factors of –1, 0, or +1.

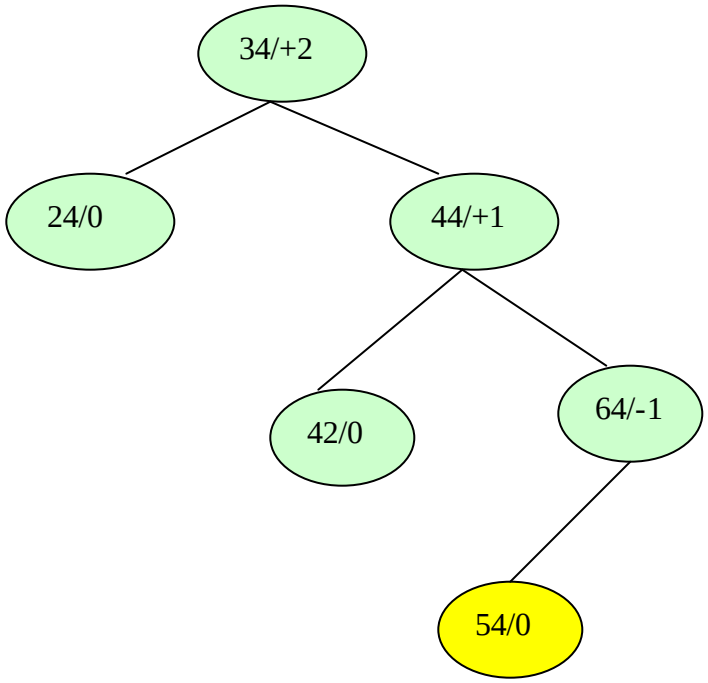
(c) Assume that the node with value 54 (the yellow node) was just inserted into the tree. Was the tree an AVL-tree before this insert occurred?

Yes, this is illustrated in the tree below which is the tree above before the insertion of the node containing value 54.



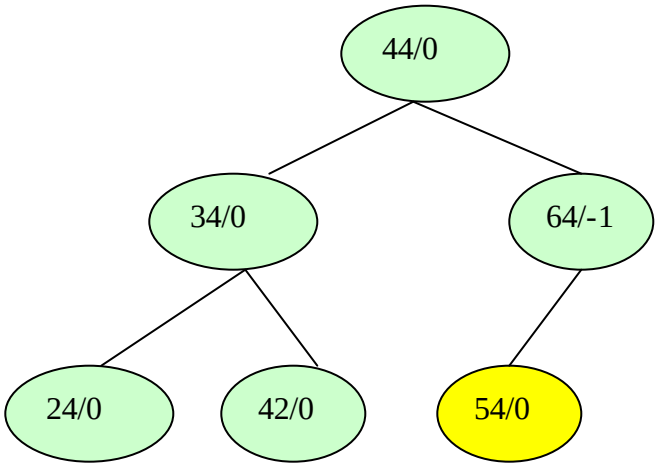
(d) Re-balance the tree that resulted after the insertion of the node with value 54 so that once again the tree is an AVL-tree.

The following trees show the re-balancing that will occur which results in the final AVL-tree. The rotation occurs about the root node of the tree since this is the node with a +2 balance factor. The insertion was in the right subtree of a right child so a single left rotation will rebalance the tree.



Tree after insertion of 54 – nodes displayed as value/balance factor

The root is the first un-balanced node and since its balance factor is positive (+2), this indicates that the problem is in its right sub-tree so a left-rotation will occur. This is shown below:



Step by step: 34 and 44 interchange (child becomes parent), but since 44 is a right child of 34, 34 will become a left child of 44 when 44 becomes the parent. 24 will remain as the left child of 24. 42 which was the left child of 44 becomes the right child of 34. 64 remains as the right child of 44.