EECS360 — Data Stuctures and Algorithms
First Mid-Term Exam (Make-up)

Feb 22, 2000

You are allowed one sheet of notes, no larger than 8.5x11 inches, double-sided, hand-written in a reasonable size font. No books, notebooks, references, friends ...

Name (printed):

Signature:

1 Short Answers / Multiple Choice / True or False (25 pts)

1. (**10 points**) Some of the following statements are true, others are false. Circle the correct answer.

(a)	No operation in a splay tree takes longer than *O(log n)*	True	False
(b)	The depth of AVL trees is *O(log n)*.	True	False
(c)	$log^k N = O(N)$ for any constant k.	True	False
(d)	Two programs A and B have worst time complexity no greater than $1000N$ and $2N^2$, respectively. Program A will run faster than program B, no matter what N is.	True	False
(e)	The depth of a binary search tree cannot be greater than *log n*.	True	False
(f)	A queue can be efficiently implemented with an array.	True	False
(g)	Take any two leaves in an AVL tree. The difference in their depths cannot be greater than 1.	True	False
(h)	There are trees for which preorder, inorder, and postorder traversals return the same sequence of nodes.	True	False
(i)	In a tree with branching factor $bf = 3$, every node has either 0 or 3 children.	True	False
(j)	A binary tree and a doubly linked list with no header have the same space requirements, if the content of the nodes is the same.	True	False

2. (**5 points**) Suppose we have stored an arithmetic expression of the usual kind — such as $(b - a) * (b + c)$ — in an expression tree. If we traverse the expression tree with an inorder visit which simply prints the contents of the nodes, do we get back **exactly** the original expression? Why or why not? Circle the correct answer:

   (a) No, a preorder traversal would return the original expression.

   (b) Yes, because an inorder traversal will return operators and operands in exactly the same order as in the original expression.

   (c) No, an expression tree only stores operators, not their operands.

   (d) No, because parentheses will be missing in the expression returned by the inorder traversal.

   (e) Yes, as long as we use an auxiliary stack.

3. **(5 points)** Among the following definitions, which is the most accurate one for a *splay* tree? Note that more than one may be true: you are asked to pick the most accurate one

   (a) A tree

   (b) A binary search tree

   (c) A perfectly balanced binary tree

   (d) A binary search tree which is restructured after a *find* or a *delete*, but without maintaining any balance condition

   (e) An AVL tree where the difference in heights between the two subtrees of a node can be at most two

4. **(5 points)** Arrays are considered a bad choice to implement a generic linked list. On the other hand, they are a potentially good choice to implement a stack. Pick the correct reason.

   (a) At any given point in time, a stack cannot have more than *O(log n)* elements, where N is the number of `push` operations performed so far.

   (b) Insert / delete at the same end of the list don't require other elements of the array to be shuffled.

   (c) For a stack, `malloc` may run out of space much more easily than for a generic list.

   (d) The maximum size of a stack is known a priori.

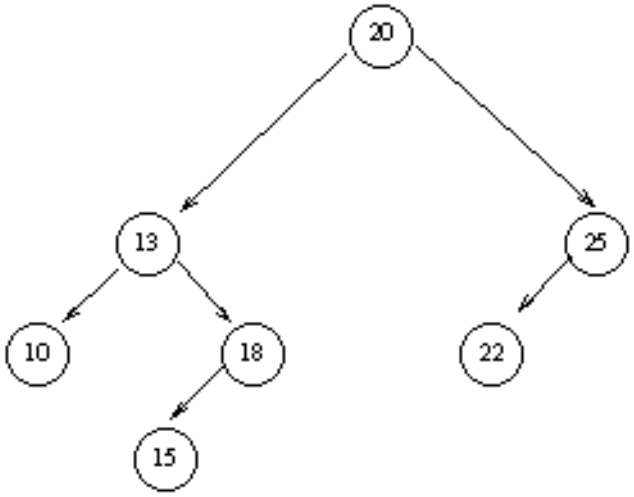
   (e) Arrays are appropriate because of fast access to any element, which is required by the stack ADT.

2 Analysis (30 points)

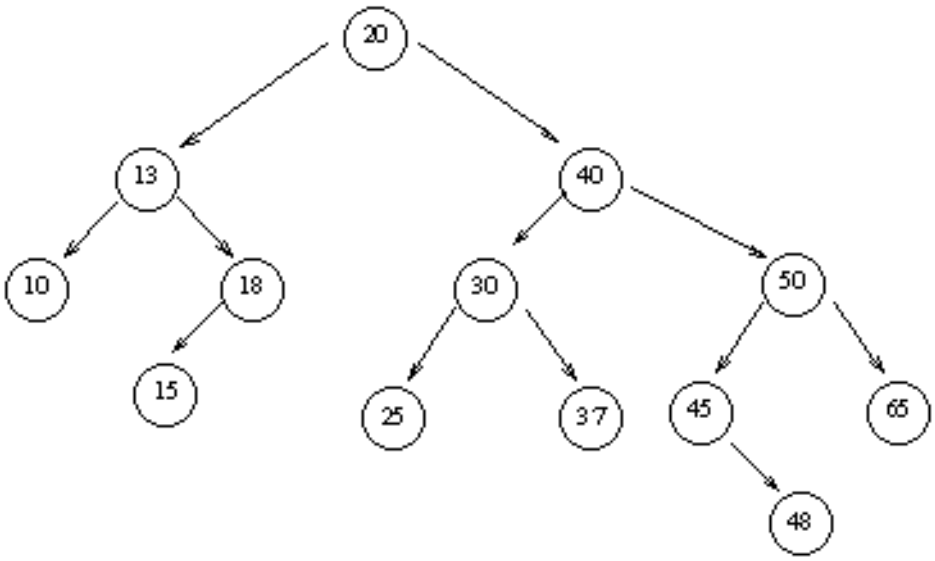
1. **(5 points)** Show (by using the usual rules) that the following program fragment is $O(N^2 logN)$.

```
for ( i=0; i < N*N; i++)
    for (j=N; j > 0; j /= 2)
       sum++;
```

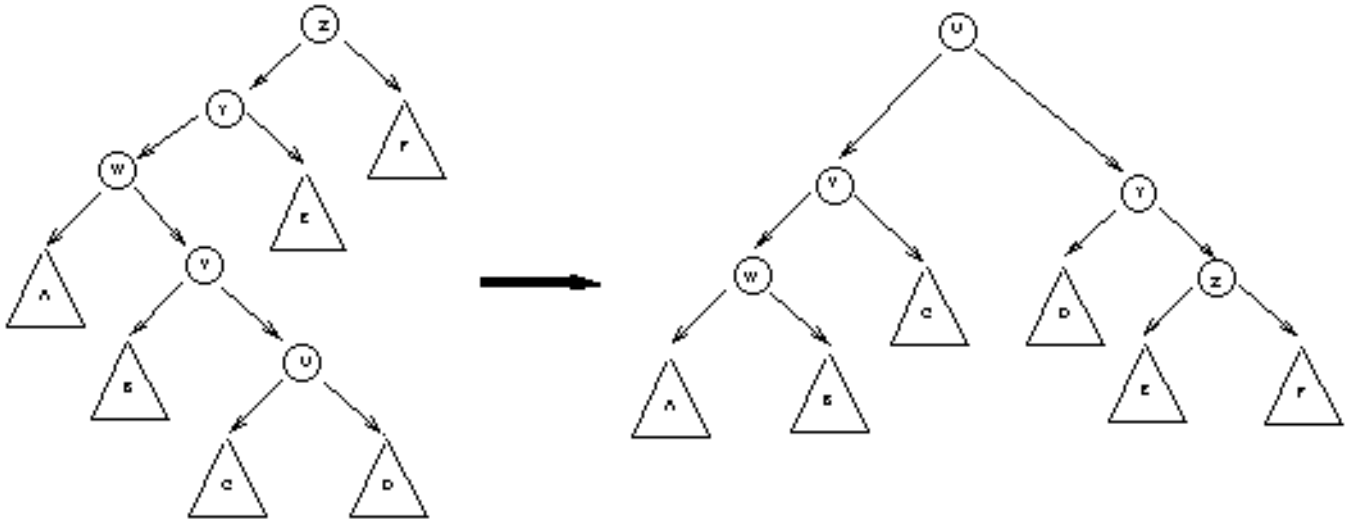
2. **(5 points)** Is the following an AVL tree? Either prove it's balanced by showing every node is balanced, or show it is not balanced by identifying all the unbalanced nodes, and showing why they're unbalanced.



3. **(5 points)** Delete 40 from the binary search tree below



4. **(15 points)** What is the sequence of transformations that transforms the tree on the left onto the tree on the right? Show the intermediate trees



3 Original code development (45 points)

3.1 Breadth-first traversal (20 points)

Write a function that traverses a binary tree breadth-first. A breadth first traversals visits a tree level by level. For example, in a breadth first traversal the tree in question 2.3 would be printed as follows (assuming the print function starts a new line every time it changes level):

```
20
13 40
10 18 30 50
15 25 37 45 65
48
```

Write a print function that prints a binary tree breadth-first. You don't need to worry about starting a new line when the function moves to the next level. You can assume the usual definition for the node of a binary tree, e.g. from Weiss:

```
typedef struct TreeNode *PtrToNode;
typedef struct PtrToNode Tree;

struct TreeNode
{
   ElementType Element;
   Tree        Left;
   Tree        Right;
};
```

3.2 A "deque" (25 points)

A *deque* is a data structure consisting of a list of items, on which the following operations are possible:

- *Push(x,D)*: Insert item X on the front end of deque D.
- *Pop(X,D)*: Remove the front item from deque D and return it.
- *Inject(X,D)*: Insert item X at the rear of deque D.
- *Eject(X,D)*: Remove the rear item from deque D and return it.

Write four functions to implement these operations. You can choose either a linked list or an array. Each function is worth 5 points; the other 5 points come from the appropriate declarations. You don't need to write `main()`.