



## Midterm Examination 2

ECE-242 Algorithms and Data Structures  
Spring Semester, 2003

Monday, March 24, 2003

Print your name and ID number neatly in the space provided below; print your name at the upper right corner of every page. Place a picture ID on your table for verification.

Your Name:
ID Number:
Instructor Name:

This booklet should be eight pages long including this page. If not, report this to the instructor or the TA. Do all problems in this booklet. Try not to spend too much time on any one question. Raise your hand if you have a question.

This is a closed book exam except for a double-sided half-page aid sheet. Foreign students can use a dictionary. Nothing else is allowed. Use terminology from the lecture notes. You must define other terminology before you use it. **Do all work in the space provided.** Ask the proctor if you need more paper. **Nothing on the back of the sheets will be graded.**

Question	Points	Score	Grader
Multiple Choice	18		
Short Answers	44		
Problem 1	18		
Problem 2	20		
Total	100		

**(Multiple Choice, 18 points)** Each question is worth 3 points and it has only one correct answer. Mark one answer; if we cannot understand what you marked you will receive no credit! In all questions,  $n$  is the number of keys in the data structure.

1) The worst-case performance of MERGESORT is

- (a)  $O(n^2)$                       (b)  $O(n \log n)$                       (c)  $O(n)$                       (d)  $O(n^3)$

2) The best-case performance of MERGESORT is

- (a)  $O(n^2)$                       (b)  $O(n \log n)$                       (c)  $O(n)$                       (d)  $O(n^3)$

3) The worst-case time to perform a single rotation in an AVL tree is

- (a)  $O(n)$                       (b)  $O(\log n)$                       (c)  $O(1)$                       (d)  $O(n \log n)$

4) Searching for a key in a heap takes worst-case time

- (a)  $O(n \log n)$                       (b)  $O(1)$                       (c)  $O(n)$                       (d)  $O(n^2)$

5) The best-case performance for Quicksort is

- (a)  $O(n \log n)$                       (b)  $O(1)$                       (c)  $O(n)$                       (d)  $O(n^2)$

6) The best-case to search for a key in AVL tree is

- (a)  $O(n \log n)$                       (b)  $O(1)$                       (c)  $O(n)$                       (d)  $O(\log n)$

**(Short Answer, 44 points)** Read each question carefully and write your answer clearly. Answer in the specified areas only. **Marks may be deducted if you do not follow these instructions!** Credit for each question is shown next to the question.

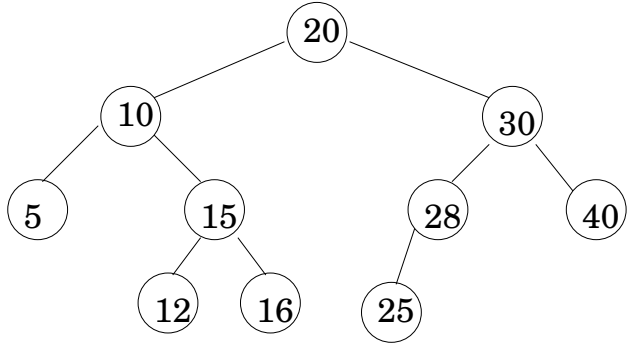
**[1. Programming, 6 points]** State what the following function does in less than 12 words.

```
void DoSomething(int *first, int *second)
{
    *first = *second - *first;
    *second = *second - *first;
    *first = *second + *first;
}
```

**[2. Binary Search Trees, 6 points]** Show the *final* Binary Search Tree if we start with an initially empty tree and we insert the seven keys below as shown from left to right:

50, 40, 45, 47, 46, 35, 41

[3. **AVL Trees, 4+4+4 points**] Draw the *final* AVL tree after applying each of the insertions in the *original* tree shown below. If you perform rotations, *indicate* they key of the node rotated and the type of the rotation (left or right) next to the tree. Write multiple rotations (if any) in the *sequence* performed.



**Insert(2) in original tree**

**Insert(11) in original tree**

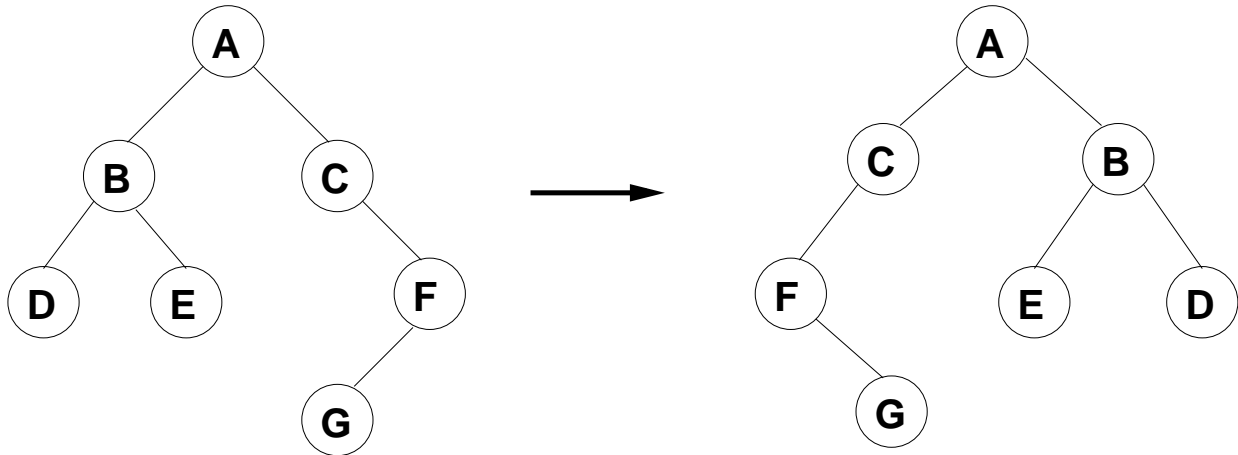
(Short Answer 3 cont.)

**Insert(22) in original tree**

[4. **Sorting, 10 points**] Perform radix sort for the following set of numbers. Fill the contents of the array with the appropriate integers at each stage of the algorithm.

<b>1470</b>				
<b>2381</b>				
<b>1072</b>				
<b>3588</b>				
<b>491</b>				
<b>2484</b>				
<b>382</b>				
<b>1233</b>				
<b>2478</b>				
<b>1531</b>				
<b>INPUT</b>	<b>STAGE 1</b>	<b>STAGE 2</b>	<b>STAGE 3</b>	<b>STAGE 4</b>

[5. Programming, 10 points] Write recursive C code for the function `ReverseTree()`. This function takes a pointer to the root of a binary tree and flips it left-to-right to generate a “mirror” image of the original, as in the example below. Do not use other functions or loops. You do not need to duplicate the tree, only modify the tree you are given.



```
typedef struct node_struct {
    int data;
    struct node_struct *left, *right;
} node_t;
```

```
void ReverseTree(node_t *root)
{
```

```
}
```

**(Problem 1, Programming, 8+10 points)**

Ackerman's function is defined recursively on nonnegative integers  $m$  and  $n$  as follows:

$$\begin{aligned} a(m, n) &= n+1 && \text{if } m == 0 \\ a(m, n) &= a(m-1, 1) && \text{if } m != 0, n == 0 \\ a(m, n) &= a(m-1, a(m, n-1)) && \text{if } m != 0, n != 0 \end{aligned}$$

(a) Calculate the value of  $a(2, 2)$ .

(b) Write a single recursive function in C that computes Ackerman's function. Do not use other functions or loops. Assume parameters  $m$  and  $n$  passed are always greater or equal to zero.

```
int MyAckerman(int m, n)
{
```

```
}
```

**(Problem 2, Heaps, 12+8 points)** Recall `BuildHeap(1..n)` that builds a heap of  $n$ -elements from an array:

```
for i = n/2 to 1 by -1 do
  bubble_down(i);
```

(a) For the array drawn below, fill the boxes for each iteration of `BuildHeap(1..10)`. Assume that the root of the heap holds the *maximum* element.

1	2	3	4	5	6	7	8	9	10
12	28	7	31	2	45	40	30	50	1

--	--	--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--	--	--

--	--	--	--	--	--	--	--	--	--

(b) Draw the final heap (binary tree) structure.