Computer Science I (COP 3502) Exam 2 Practice Questions Solutions
Topics: Sorting, Binary Trees, AVL Trees, Heaps, Tries and Hash Tables

1) Show the contents of the following array after each iteration of Bubble Sort:

Initial Values	8	3	6	1	7	5	2
1st iteration	3	6	1	7	5	2	8
2nd iteration	3	1	6	5	2	7	8
3rd iteration	1	3	5	2	6	7	8
4th iteration	1	3	2	5	6	7	8
5th iteration	1	2	3	5	6	7	8
Last iteration	1	2	3	5	6	7	8

2) Show the result of partitioning the array below, using the leftmost element as the partition element. Please use the in-place partitioning algorithm shown in class.

Initial Values	6	9	3	12	5	2	13	18	4	1	7
After Partition	2	1	3	4	5	6	13	18	12	9	7

3) Why does Quick Sort run faster than Merge Sort, on average, in practice?

It runs faster because it does not have the overhead of using extra temporary arrays to copy values and copy back values into the original array. In short, Quick Sort runs in place, and this makes it run faster in practice even though it doesn't always split its input list into two halves for recursive calls.

4) Provide the Preorder, Inorder and Postorder traversals of the following binary tree:

```
                100
             /        \
        13               88
      /    \            /
    45     67         59
          /  \       /  \
         9    22   74    93
              /      \
            14       105
```

Pre-Order: 100, 13, 45, 67, 9, 22, 14, 88, 59, 74, 105, 93

In-Order: 45, 13, 9, 67, 14, 22, 100, 74, 105, 59, 93, 88

Post-Order: 45, 9, 14, 22, 67, 13, 105, 74, 93, 59, 88, 100

5) Show the result of deleting 47 from the binary search tree shown below. (Note: There are two possible right answers.)

```
              65
          /           \
        47             89
      /     \
    13       58
   /  \     /  \
  6    25  51  62
      /     \
    22       54
```

We can either replace 47 with the max on its right (25), or the min on its left (51). Here is the result of both (so both correct answers):

```
              65                               65
          /           \                    /           \
        25             89                51             89
      /     \                           /   \
    13       58                       13     58
   /  \     /  \                     /  \    /  \
  6    22  51  62                   6  25  54  62
            \                          /
            54                       22
```

6) In a binary search tree, consider adding all the values at an even depth from the root, and subtracting all the values at an odd depth from the root. For example, for the tree shown below:

```
            50
         /      \
       23        76
      /  \      /
    14   29    61
```

The desired sum would be 50 - 23 - 76 + 14 + 29 + 61 = 10.

Write a recursive function that calculates this adjusted sum for a binary search tree, given a pointer to its root. **(Hint: If I am a tree rooted at 50 above, then the contribution to my sum is the NEGATIVE of the corresponding sum for each of my subtrees.)** Please use the struct and function prototype given below:

```
typedef struct bintreenode {
    int data;
    struct bintreenode* left;
    struct bintreenode* right;
} bintreenode;

int getAdjustedSum(bintreenode* root) {

    if (root == NULL) return 0;

    return root->data - getAdjustedSum(root->left)
                      - getAdjustedSum(root->right);

}
```

7) (Show the result of inserting the value 60 into the AVL tree below. Put a box around your final answer.

```
            40
          /     \
        20        75
       /  \      /  \
     10   30    50   80
               /  \    \
              44  65    90
```

When we do this, we get an imbalance at 40, the rebalancing (and final answer) is on the right:

```
          40                          50
        /     \                     /     \
      20        75                40        75
     /  \      /  \              /  \      /  \
   10   30    50   80          20 44     65  80
             /  \    \        /  \       /     \
            44  65    90     10 30     60       90
                /
               60
```

8) Complete the function below so that it counts the number of words in the trie pointed to by root and returns this value. Please use the struct definition and function prototype shown below:

```
typedef struct trie {
    int isWord;
    struct trie* next[26];
} trie;

int numWords(trie* root) {

    if (root == NULL) return 0;
    int res = root->isWord;
    for (int i=0; i<26; i++)
        res += numWords(root->next[i]);
    return res;
}
```

9) Consider implementing a hash table that stores integers, using the linear probing strategy. Assume that the hash table uses the hash function, f, defined below and that size is 23. Show the contents of the table after the following insertions have been made, in the order given: 4362, 999235, 7283624, 8123456, 77, 11111111, 52, 123, 7999999 and 12345675.

```
int f(int n, int size) {
    int res = 0;
    while (n > 0) {
        res = (res + (n%10))%size;
        n = n/10;
    }
    return res;
}
```

index	0	1	2	3	4	5	6	7	8	9	10	11
value							**8123456**	**52**	**11111111**	**7283624**	**123**	**12345675**

index	12	13	14	15	16	17	18	19	20	21	22
value			**999235**	**4362**	**77**	**7999999**					

Note: The key here is to realize that the function just returns the sum of the digits mod size. If you don't realize that, this question takes 20 minutes instead of 3 and it takes away time from the rest of the exam. This was done by design, to reward the students who were able to look at the code and translate what it does into English as opposed to a literal translation line by line.

10) Consider inserting the following items into a minimum heap in the following order: 18, 8, 16, 4, 3, 12, 17, 2, 22, and 7. Show the state of the heap (drawn as a complete binary tree) after the completion of each insertion. Draw a box around each of your answers.

Solution

1st insert:

```
18
```

2nd insert:

```
   8 (perc up 1)
  /
18
```

3rd insert:

```
    8
  /   \
18     16
```

4th insert:

```
      4 (perc up twice)
    /   \
   8     16
  /
18
```

5th insert:

```
        3 (perc up twice)
      /   \
    4      16
  /   \
18     8
```

6th insert:

```
       3
     /    \
    4      12 (perc up 1)
  /   \   /
18    8 16
```

7th insert:

```
        3
      /    \
    4       12
  /   \    /   \
18     8  16   17
```

8th insert:

```
          2 (perc up 3 times)
       /      \
     3         12
   /   \     /    \
  4     8   16    17
 /
18
```

9th insert:

```
        2
      /    \
    3       12
   /  \    /  \
  4    8  16   17
 / \
18 22
```

10th insert:

```
          2
        /   \
      3       12
    /   \    /  \
  4      7  16   17
 / \    /
18 22  8
```