# <u>2016 Spring COP 3502 Final Exam</u>

1) (10 pts) 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 | | | | | | | | | | | | |

| index | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| value | | | | | | | | | | | |

2) (10 pts) 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.

3) (12 pts) It was mentioned in class that a Bubble Sort could be implemented recursively, since after one iteration of a Bubble Sort on n elements, the maximum value is correctly sorted, so what remains is an array of size n-1. Fill in the function shown below so that it implements a recursive Bubble Sort. You may (and should) call the swap function that is provided below.

```
void swap(int* ptrA, int* ptrB) {
    int temp = *ptrA;
    *ptrA = *ptrB;
    *ptrB = temp;
}

void bubbleSortRec(int* array, int n);
```

4) (8 pts) Write the function below so that takes in an integer n and returns the number of bits set to 1 in the binary representation of n. In order to get full credit, please use bitwise operators.

```
int bitCount(int n);
```

5) (15 pts) A new company, iTraits, has designed a personality test so that people can identify others like them and unlike them. They use a 20 question survey of yes/no questions to characterize each person in their database. Each person's personality can be stored efficiently in a computer program as an integer, where the $i^{th}$ bit is set to 0 if the answer to question i is no, and the $i^{th}$ bit is set to 1 if the answer to question i is yes. (The questions are numbered 0 to 19.) For example, if a person's answers to questions 0 through 3 were yes and questions 4 through 19 were 0, the value $15 = 2^0 + 2^1 + 2^2 + 2^3$ would be stored for their personality.

We define someone's arch-nemesis as the person with whom they differ on the most number of traits. Complete the function below so that it takes in the value corresponding to a particular individual's personality (personA), the array of values (otherPeople) representing personalities of each other person in a database, and the length of that array (length), and returns the index into otherPeople of the arch-nemesis of personA. If there are multiple arch-nemesi, you may return the index of any one of them. You may call the bitCount function from question 4 and assume that it works properly. In order to get full credit, please use bitwise operators.

```
int getArchNemesis(int personA, int* otherPeople, int length);
```

6) (15 pts) Consider a binary search tree of integers where each node stores the number of nodes in its subtree. The struct for such a tree is as follows:

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

Write a function that takes in a pointer to a `struct bintreenode root`, and an integer `k`, and returns the $k^{th}$ smallest integer in the tree rooted at root. (Hint: make your function recursive. If you look at how many values are in the left subtree, it will tell you whether or not the $k^{th}$ smallest value is in the left subtree, the right subtree or the root.)

```
// Pre-conditions: root != NULL and 1 <= k <= root->numNodes.
int findKthSmallest(struct bintreenode* root, int k);
```

7) (12 pts) Using the iteration technique, find a Big-Oh bound for the following recurrence relation, in terms of n:

$$T(n) = 2T\left(\frac{n}{2}\right) + n^2, \text{ for } n > 1$$
$$T(1) = 1$$

8) (7 pts) An $O(n^2)$ algorithm takes half a second to run on an input of size 10,000. Roughly, how long will it take to run on an input of size 50,000, in seconds?

9) (7 pts) What is the base 10 value of 5789 equal when converted to base 4?