COP 3502 Quiz #2 Version A (Recursion, Linked Lists, Stacks, Queues, SLMP) Solution

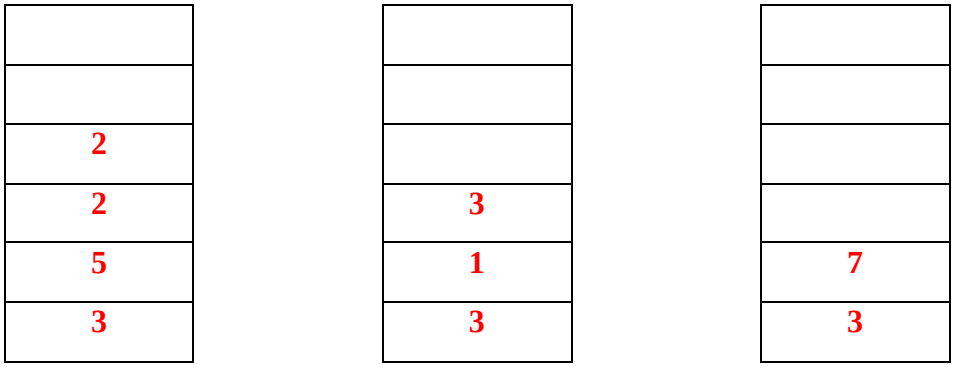
1) (5 pts) In the Towers of Hanoi code shown in class, a transcript of the moves to solve the puzzle was printed. Write a related function that adds up the sum of the disk numbers for each move for solving the puzzle for ***n*** disks. For example, if ***n = 3***, your function should return 11 because the disks that move are disk numbers 1, 2, 1, 3, 1, 2 and 1, respectively and 1 + 2 + 1 + 3 + 1 + 2 + 1 = 11. Notice that the sum to the left of the three and the right of the three are the same, so it's okay to calculate it once and multiply that by 2.

```
int sumTowerDisks(int n) {

    // Grading: 2 pts base case.
    if (n == 0) return 0;

    // Grading: 1 pt add n, 2 pts 2 times rec call on n-1.
    return n + 2*sumTowerDisks(n-1);

}
```

2) (5 pts) Evaluate the following postfix expression, showing the state of the stack at each of the indicated points.

A B C

3 5 2 2 * - 3 2 * + * 9 + 5 /



Value of the Expression: **6**

Grading: 1 pt for each stack (all or nothing), 2 pts for the final answer (all or nothing)

3) (8 pts) Consider using a linked list of digits to store an integer. Specifically, the integer 2471 would be stored as the following linked list: 1 → 7 → 4 → 2. (Thus the first node stores the units digit, the second node the ten's digit, the third node the hundred's digit and so forth.) Write a ***recursive*** function that takes in a pointer to the front of a linked list storing an integer and returns the numeric value of the list. (Hint: To solve the problem carefully think about the value generated by the list 7 → 4 → 2 in relation to the number above. Also, an empty list has a value of 0.)

```
typedef struct numNode {
    int digit;
    struct node* next;
} numNode;

int getValue(numNode* number) {

    // Grading: 2 pts base case.
    if (number == NULL) return 0;

    // Grading: 1 pt return, 1 pt n->d, 2 pts 10*, 1 pt GV,
    //          1 pt n->n
    return number->digit + 10*getValue(number->next);

}
```

4) (5 pts) What problem does the function below solve? Namely, what is the return value of the function in relation to the input parameters? Your answer should be a regular sentence or two in English and NOT a literal translation of the code. The latter will receive **no credit!**

```
// Pre-condition: array is size n and 1 <= k <= n.
int mysteryA(int* array, int n, int k) {

    int res = 0, tmp;
    for (int i=0; i<k; i++)
        res += array[i];
    tmp = res;

    for (int i=0; i<n-k; i++) {
        tmp = tmp - array[i] + array[i+k];
        if (tmp > res) res = tmp;
    }

    return res;
}
```

This function finds the maximum sum of any k consecutive values in the array.

Grading: 2 pts for mentioning adding k consecutive values from the array, 2 pts for mentioning "maximum" 1 pt for properly putting together these two ideas.

5) (5 pts) The permutation algorithm shown in class generates the permutations of {0, 1, 2, …, , n-1} in lexicographical order. Which permutation does the algorithm generate after the permutation shown below?

3, 7, 6, 2, 9, 4, 8, 5, 1, 0

3, 7, 6, 2, 9, 5, 0, 1, 4, 8

Grading: 2 pts for keeping 3, 7, 6, 2, 9 the same. 1 pt for the placement of 5, 2 pts for the correct arrangement of 0, 1, 4, and 8.

6) (7 pts) In the implementation of a queue using an array shown in class, the following four components formed the structure storing the queue:

```
elements - an array storing the items in the queue.
front - index of the array that is the front of the queue
numElements - number of elements currently in the queue.
queueSize - currently allocated space for array elements.
```

Assume that the queueSize is initially set to 10 and that the following code segment is run:

```
struct queue* MyQueuePtr = malloc(sizeof(struct queue));
init(MyQueuePtr);

for (int i=0; i<9; i++) {
    enqueue(MyQueuePtr, i+1);
    enqueue(MyQueuePtr, 2*(i+1));
    int tmp = dequeue(MyQueuePtr);
}
```

Show the value of each of the four components of the struct pointed to by MyQueuePtr right after this code segment is executed. **Note: for the array, ONLY fill in the slots that have meaningful data. Leave blank any spots that aren't storing something in the queue.**

front **9** numElements **9** queueSize **10**

index	0	1	2	3	4	5	6	7	8	9
elements	**6**	**12**	**7**	**14**	**8**	**16**	**9**	**18**		**10**

Grading: 1 pt for each of front, numElements and queueSize
1/3 pt for each of the other 9 entries, round down
So only give 7/7 if everything is correct. 4 of the entries in the array can be wrong to get 6/7