COP 3502 Study Group Sheet: Linked Lists Solutions

Directions: Work together as a group to try to solve these problems. Talk through issues and see if you can convince yourselves of the right path to move forward.

1) Suppose we have a stack implemented as a linked list. The stack is considered "full" if it has 20 nodes and empty if the head pointer is NULL. The nodes of the stack have the following structure:

```
typedef struct node {
    int data;
    struct node* next;
} node;
```

Write a function to determine if the stack is full.

```
int isFull(node *stack) {

    int cnt = 0;
    while (stack != NULL) {
        cnt++;
        if (cnt == 20) return 1;
        stack = stack->next;
    }

    return 0;
}
```

2) Write a **recursive** function that takes in the head of a linked list and frees all dynamically allocated memory associated with that list. You may assume that **all** the nodes in any linked list passed to your function (including the head node) have been dynamically allocated. It's possible that your function might receive an empty linked list (i.e., a NULL pointer), and you should handle that case appropriately.

Note that your function must be recursive in order to be eligible for credit.
The linked list node struct and the function signature are as follows:

```
typedef struct node {
     struct node *next;
     int data;
} node;

void destroy_list(node *head) {
    if (head == NULL) return;
    if (head->next != NULL) destroy_list(head->next);
    free(head);
}
```

3) Write an iterative function which takes in a pointer to a linked list and returns 1 if all the items in the list are in sorted order from smallest to largest (ties allowed) and 0 otherwise.

```
int isSorted(node *head) {

    if (head == NULL) return 1;

    while (head->next != NULL) {
        if (head->data > head->next->data) return 0;
        head = head->next;
    }

    return 1;
}
```

4) Write a function that takes in a pointer to a linked list. If the list is either size 0 or size 1, just return a pointer to the front of the list. If it's longer, take the first node in the list and move it to the back of the list, returning a pointer to the new front of the list.

```
node* frontToBack(node* head) {

    if (head == NULL || head->next == NULL) return head;

    node* newfront = head->next;
    head->next = NULL;

    node* tmp = newfront;
    while (tmp->next != NULL)
        tmp = tmp->next;

    tmp->next = head;
    return newfront;
}
```