Name: Inho Choi **ID:** 1801187.

Problem 1:

a. Implement a recursive function sumOfDigits that takes a positive integer as input and returns the sum of all of the digits in the integer. For example, the sum of 55 is 10, the sum of 123 is 6, etc. Do not use any global or static variables.

```
int sumOfDigits(int num) {
        If (n==0) {
            return 0;
        }

        return n + sumOfDigits(n-1);

}
```

b. How many times is sumOfDigits called for any given number? Use induction: how many for the number 1? how many for the number 432? how many for 5555? etc.

The sumofDigits is called n+1 times.

Name: Inho Choi **ID:** 1801980.

Problem 2:

a. A palindrome is a word, phrase or sequence that reads the same backward as forwards, for example, "bob", "step on no pets". Write a recursive function isPalindrome that takes a cstring and its size as input and returns true if it is a palindrome, false otherwise. The size must be the same as the number of actual characters in the cstring. Do not use any global or static variables. Below is some example usage:

```
isPalindrome("tacocat", 7); // true
isPalindrome("abka", 4);    // false
```

You should not require any additional libraries or functions to implement this function.

```
bool isPalindrome(const char * s, int size) {
    if( size == 0 || size == 1 ) {
        return true;
    }
    if( s[0] == s[size-1] ){
        s++;
        size--;
        return isPalindrome( s, (size-1));
    }
}
```

b. How many times if isPalindrome called for any given cstring? Use induction, how many for "a"? how many for "aa"? how many for "aba"? etc.

The isPalindrom is called (size/2) times.
"a" is called 1 time, "aa" is called 2 times, and "aba" is called 2 times.

Name: Inho Choi **ID:** 1801780.

Problem 3

a. Implement a recursive function deleteList, that takes pointer to the head of a singly linked list and deletes the whole list.

```
struct Node {
        int val;
        Node* next;
};

void deleteList(Node* head) {
if ( head == nullptr) {
    return 0
}
Node *killMe = head;
 head = head->next;
   delete killMe;
 return deleteList(head);

}
```

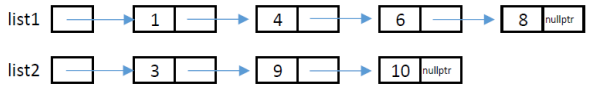
b. How many calls to deleteList would you make if you passed in a linked list with q nodes?

The deleteList is called q times.

Name: Inho Choi **ID:** 1801081

Problem 4:

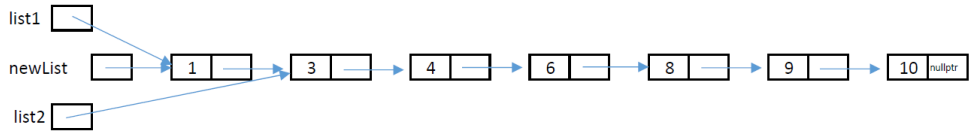
a. Given the Node definition above, implement a recursive function that merges two sorted singly linked lists into a single sorted linked list. The function should return the head of the new list. You may not create any new Nodes, this is known as an in-place merge. For example suppose we have two list:



And we call our merge function on these two lists:

```
Node* newList = inPlaceMerge(list1, list2);
```

The state of our program after that function call may look like:




```
Node* inPlaceMerge(Node* list1, Node* list2) {
if( list1 == nullptr){
    return list2;
}
if( list2 == nullptr){
    return list1;
}
if ( list1->val < list2->val){
    list1->next = inPlaceMerge( list1->next, list2);
    return list1;
}else{
    list2->next = inPlaceMerge (list1, list2->next);
    return list2;
}
}
```

b. If list1 has **n** nodes and list2 has **m** nodes, roughly how many times is inPlaceMerge called? Use induction: if list1 has 1 node and list2 has 4 nodes? If list1 has 3 and list2 has 2 nodes?

The InPlaceMerge is called n times.
If list 1 has 1 node and list 2 has 4 nodes, it calls 1 time.
If list 1 has 3 node and list 2 has 2 nodes, it calls 3 times.

Name: Inho Choi **ID:** 180178D.

Extra space for inPlaceMerge if provided space is not sufficient:

```
Node* inPlaceMerge(Node* list1, Node* list2) {

}
```