Assignment 4

1. (Weight: 10%) Trace the execution of the call `mystery(4)` for the following recursive function. What does this function do?

```
int mystery(int n) {
  if (n == 0)
    return 0;
  else
    return n * n + mystery(n - 1);
}
```

2. (Weight: 20%) **Programming:** Write a recursive function **`to_number`** that forms the integer sum of all digit characters in a string. For example, the result of to_number("3ac4") would be 7.

   Hint: If next is a digit character ('0' through '9'), function `is_digit(next)` in header <cctype> will return true.

3. (Weight: 10%) **Programming:** Write a recursive linear search function with a recursive step that finds the last occurrence of a target in a vector, not the first. You will need to modify the linear search function (You can find it in the slides) so that the last element of the vector is always tested, not the first.

4. (Weight: 5%) What is the big-O for the Towers of Hanoi as a function of n, where n represents the number of disks?

5. (Weight: 40%) **Programming:** Programmatically generate three type of arrays (or vectors) of size 10,000: an array that is sorted in ascending order, a reversed array (an array that is sorted in descending order), and a random array.

   - Programmatically sort the three arrays using bubble sort, selection sort, insertion sort, shell sort, merge sort, and quick sort (the regular one and the improved one) algorithms. For each sorting algorithm, calculate the number of exchanges (or shifts) and the number of comparisons for these sorting algorithms. Summarize the result in tables (see below).

   - Discuss whether your results match the big O of these algorithms.

   - **Note:** you can use the sorting algorithms on Blackboard. For quick sort, the improved one is called "middle_quick_sort". Your program will crash when you try to sort a sorted array with the regular quick sort. That's okay.

- **Submission:** Please submit the source code (.cpp files) as well as tables that summarize the results: This is what the table should look like:

Comparing the number of comparisons

	Random Array	Sorted Array	Reversed Array
Selection Sort			
Bubble Sort (improved)			
Insertion Sort			
Shell Sort			
Merge Sort			
Quick Sort			
Quick Sort (Improved)			

Comparing the number of exchanges:

	Random Array	Sorted Array	Reversed Array
Selection Sort			
Bubble Sort (improved)			
Insertion Sort (shifts)			
Shell Sort (shifts)			
Merge Sort (copies)			
Quick Sort			
Quick Sort (Improved)			

6. (Weight: 15%) **Programming:** Modify the source code for the insertion sort so that it can sort a list of integers (You can find the source code on Blackboard and the slides). Is there a difference in performance (compared to the vector-based one)?