CptS 223 – Advanced Data Structures in C++

Written Homework Assignment 2: Big-O and Algorithms

I. Problem Set:

1. **(20 pts)** Given the following two functions which perform the same task:

```
int g (int n)
{
   if(n <= 0)
   {
      return 0;
   }
   return 1 + g(n - 1);
}
```

```
int f (int n)
{
   int sum = 0;
   for(int i = 0; i < n; i++)
   {
      sum += 1;
   }
   return sum;
}
```

a. (10 pts) State the runtime complexity of both f() and g().

b. (10 pts) Write another function called "int h(int n)" that does the same thing, but is significantly faster.

2. **(15 pts)** State g(n)'s runtime complexity:

```
int f (int n){
  if(n <= 1){
    return 1;
  }
  return 1 + f(n/2);
}

int g(int n){
  for(int i = 1; i < n; i *= 2){
    f(n);
  }
}
```

3. **(20 pts)** Write an algorithm to solve the following problem (10 pts)

Given a nonnegative integer n, what is the smallest value, k, such that

1n, 2n, 3n, …, kn

contains all 10 decimal numbers (0 through 9) at least once? For example, given an input of "1", our sequence would be:

$$1 * 1, 2 * 1, 3 * 1, 4 * 1, 5 * 1, 6 * 1, 7 * 1, 8 * 1, 9 * 1, 10 * 1$$

and thus k would be 10. Other examples:

Integer Value	K value
10	9
123456789	3
3141592	5

(10 pts). Can you directly formalize the worst case time complexity of this algorithm? If not, why?

4. **(20 pts)** Provide the algorithmic efficiency for the following tasks. Justify your answer, often with a small piece of pseudocode to help with your analysis.

   a. (3 pts) Determining whether a provided number is odd or even.

   b. (3 pts) Determining whether or not a number exists in a list.

   c. (3 pts) Finding the smallest number in a list.

   d. (4 pts) Determining whether or not two **unsorted** lists of the same length contain all of the same values (assume no duplicate values).

   e. (4 pts) Determining whether or not two **sorted** lists contain all of the same values (assume no duplicate values).

   f. (3 pts) Determining whether a number is in a balanced BST.

5. **(25 pts)** Write a pseudocode or C++ algorithm to determine if a string `s1` is an *anagram* of another string `s2`. If possible, the time complexity of the algorithm should be in the worst case O(n). For example, ‘abc’ - ‘cba’, ‘cat’ – ‘act’. s1 and s2 could be arbitrarily long. It only contains lowercase letters a-z. Hint: the use of histogram/*frequency* tables would be helpful!

II. Submitting Written Homework Assignments:

1. On your local file system, create a new directory called HW2. Move your HW2.pdf file into the directory. In your local Git repo, create a new branch called HW2. Add your HW2 directory to the branch, commit, and push to your private GitHub repo created in PA1.
2. Do not push new commits to the branch after you submit your link to Canvas otherwise it might be considered as late submission.
3. Submission: You must submit a URL link of the branch of your private GitHub repository to Canvas.