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().

g(): O(n)
f(): O(n)

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

```
int h(int n) { return n+1;}
```
h(): O(1)

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);
  }
}
```

f(): O(n/2) = O(n)
g(n): O(n) x O(n) = O(n^2)

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?

```
int findK(int x){
        bool arr[10] = {false, false, false , false , false , false , false , false , false , false };
        int k = 1;
        int j = x;
        int d;
        while(true) {
                while(j > 0) {
                        d = j % 10;
                        j = j / 10;
                        arr[d] = true;
                }
                if (arrAllTrue(arr)) return k; // bool arrAllTrue(bool arr[]); returns true if all bool in arr are true.
                else {
                        k++;
                        j = x * j;
                }
        }
}
```

The arrAllTrue(arr) has a time complexity of O(10). Because k is always 9 or less, the first while loop only repeats at most 9 times, so the time complexity is O(9). The time complexity of the inner loop is O(n), so the overall time complexity ii O(9 * 10 * n) which is just O(n).

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.

      This would be O(1) because it would just be: return if(x%2 ==0);

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

      This one would only be O(n), n being the length of the list, as a worst case scenario. It has to go through one at a time checking if each number is the one looked for, and ends once it is found.

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

      This would be O(n), n being the length of the list. It needs to go through the list one at a time checking if each number is smaller than the previous smallest, and it would have to check every number.

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

      This one would require a nested loop, going through the first list one at a time, checking against every value of the second list until it finds a match or goes through the whole list. Because of the nested loop it'll be O(n^2), n being the length of both lists, because if they have different lengths it is automatically false with no duplicates.

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

      This one will just be O(n) because with it being sorted the same values will be in the same position in each list if they are matching, so it only takes one loop comparing the ith value of each list to each other.

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

      This one is O(n), n being the depth of the tree. Since the tree is balanced it cuts the time quite a bit because as you traverse the tree comparing the value, going left if is smaller than the node's value, or right if its larger, once you reach the bottom of whatever branch you followed, then you know that the value is not in the tree.

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!

```
bool anagrams(string s1, string s2) {
        int s1Length = s1.length();
        if (s1Length != s2.length()) return false;
        int j = s1Length - 1;
        for (int i = 0; i < s1Length; i++) {
                if (s1[i] != s2[j]) return false;
                j--;
        }
        return true;
}
```

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.