HW 3

Due 23:00 November 27, 2017

CS ID:

Section (circle one):

Monday	L1J 9–11a	L1N 11a–1p	L1C 4–6p
Tuesday	L1A 11a–1p	L1F 4–6p	
Wednesday	L1B 9–11a	L1E 2–4p	L1M 6–8p
Thursday	L1D 10:30a–12:30p	L1G 4–6p	
Friday	L1K 1–3p	L1H 3–5p	

Please print out this document and write your solutions into the spaces provided. Show your work where necessary for full credit. If you require additional space, please indicate in the question space that you are writing on the last blank page, and also indicate on the blank page which question the work solves.

You must scan and upload the completed document, including this page and the last page, to GradeScope, using your 4- or 5-character CS ID.

1. [**Introductory Lucky Numbers - 3 points**].

   In this assignment, we will analyze various implementations of the Lucky numbers, to practice your skills in proofs and recurrence relations.

   The Lucky numbers are similar to the Fibonacci numbers. They are the numbers in the following integer sequence, called the Lucky sequence, and characterized by the fact that every number after the first two is the sum of the two preceding ones:

   $$2,\ 1,\ 3,\ 4,\ 7,\ 11,\ 18,\ 29,\ 47,\ \ldots$$

   For this assignment, the 0th Lucky number, denoted by L_0, is defined to be 2. The 1st Lucky number L_1 is 1, and for any other integer $k > 1$, $L_k = L_{k-1} + L_{k-2}$. Hence, the sequence above is the sequence $L_0, L_1, \ldots, L_8, \ldots$.

   (a) (2 points) By hand, calculate the 9th Lucky number L_9.

   (b) (2 points) By hand, calculate the 13th Lucky number L_{13}.

   As you can probably see, this is already too much work! How about we let the computer do the work for us? :)

2. **[Recursive Lucky Numbers - 14 points].**

   The Lucky numbers are inherently recursive. Hence, a (naïve) algorithm to calculate L_n could be the following:

```
unsigned long lucky(unsigned long n)
{
    if (n == 0)
        return 2;
    if (n == 1)
        return 1;

    return lucky(n-1) + lucky(n-2);
 }
```

   (a) (3 points) Draw a tree to illustrate the recursion structure of `lucky` when $n = 5$. The root node of your tree should correspond to `lucky(5)`, and that node should have two children, one for each of `lucky(4)`, and `lucky(3)`, etc. down to the base cases, which are leaves. Inside each node, place the value of the argument to `lucky`.

   (b) (3 points) Write a recurrence relation $T_1(n)$ to describe the running time of the function `lucky`.

(c) (4 points) Consider the following similar recurrence relation, where b, c are constants:

$$T_2(n) = \begin{cases} b & \text{if } n = 0 \\ 2T_2(n-2) + c & \text{otherwise} \end{cases}$$

Find a closed form solution for the recurrence relation $T_2(n)$.

Write the closed form here: $\boxed{}$

(d) (2 points) Compare the recurrence relations $T_1(n)$ and $T_2(n)$. Is the bound found for $T_2(n)$ an **upper-bound** or a **lower-bound** for $T_1(n)$? Circle one answer below:

Upper Bound **Lower Bound**

(e) (2 points) In 20 words or less, describe why the performance for `lucky` is not desirable.

3. **[Dynamic Lucky Numbers] - 21 points.**

   In labs, we discussed using memoization to speed up the performance of computing the Fibonacci numbers. In this assignment, we will explore another method: Dynamic Programming. The following is a dynamic programming solution which computes L_n:

```
unsigned long dpLucky(unsigned long n)
{
    unsigned long dp[n+1];
    dp[0] = 2;
    dp[1] = 1;

    for(unsigned int j = 1; j < n; j++)
    {
        dp[j+1] = dp[j] + dp[j-1];
    }

    return dp[n];
}
```

   (a) (3 points) For $n = 5$, illustrate the contents of the array `dp` before and after the iteration $j = 2$.

   For the remainder of this problem, we will evaluate the following loop invariant, and prove that `dpLucky` correctly computes L_n:

   At the start of iteration j in the for-loop of `dpLucky`, all the elements of the array `dp` from 0 to j are the Lucky numbers L_0 through L_j.

   (b) (2 points) First, consider the base case. Show that the loop invariant holds when $j = 1$.

(c) (6 points) Finish the following loop invariant proof by filling in the blanks:

Suppose, for some $k \geq 1$, that the loop invariant holds; in other words, at the start of iteration k in the for-loop, all the elements of the array `dp` from 0 to k are the Lucky numbers L_0 through L_k. We will show that the loop invariant holds at the start of iteration $k + 1$, to prove the inductive step.

By the inductive hypothesis, all the elements of the array `dp` from __________ to __________ are the Lucky numbers __________ through __________. In particular, the Lucky numbers L_k and L_{k-1} are found in `dp[_____]` and `dp[_____]` respectively. So, inside the kth iteration of the for-loop, the line `dp[k+1] = dp[k] + dp[k-1]` will set `dp[k+1]` as the Lucky number __________, by definition. Since the rest of the array `dp` has not been modified, at the start of iteration $k + 1$, all the elements of the array `dp` from __________ to __________ are the Lucky numbers __________ to __________. Thus, the inductive step has been proven.

(d) (2 points) Prove that the for-loop of `dpLucky` terminates after a finite number of iterations, completing the proof of the loop invariant.

(e) (2 points) Finally, now that you've proven the loop invariant, *use* it to prove that `dpLucky` returns the correct value of L_n at program termination.

(f) (2 points) Analyze the code of `dpLucky`. What is the running time for the function?

(g) (2 points) In 20 words or less, describe how the iterative memoized solution improves the performance of computing L_n over the recursive version.

4. **[Golden Lucky Numbers] - 25 points.**

   Define the following constant ϕ, which is also known as the Golden Ratio:

$$\phi = \frac{1+\sqrt{5}}{2}$$

   (a) (3 points) Show that ϕ and $1-\phi$ are the roots to the equation $x^2 = x + 1$.

The following formula for the Lucky numbers is claimed to be true for all natural numbers n (where the natural numbers are the numbers $0, 1, 2, \ldots$):

$$L_n = (\phi)^n + (1-\phi)^n \tag{1}$$

In the remainder of this problem, we ask you to prove formula (1) by induction.

   (b) (3 points) Prove that formula (1) holds for the base case(s).

(c) (6 points) Complete the following induction proof by filling in the blanks:

Suppose that, for some integer $k \geq 1$, formula (1) is true for all $1 \leq m \leq k$; in other words, for any m between 1 and k:

$$L_m = (\phi)^m + (1 - \phi)^m.$$

Prove that formula (1) holds for $n = k + 1$.

$F_{k+1} =$ __________________________ *by definition of Lucky Numbers*

$=$ __________________________ *by the inductive hypothesis*

$=$ __________________________ *by factoring*

$=$ __________________________ *by the fact in part a)*

$=$ __________________________ *by simplification*

Therefore, the inductive step is proven, and by the principle of mathematical induction, formula (1) is true for all natural numbers n.

(d) (2 points) What is the difference between weak induction and strong induction? Which form of induction did we use in the above inductive proof?

We have now proven a simple formula for L_n! Let's put this into code. The following code snippet uses a small variation to the function `recpow` as seen in HW2:

```
#include <math.h>

float recpow(float x, unsigned long n)
{
   if (n == 0)
      return 1;
   if (n == 1)
      return x;

   float half = recpow(x, n/2);
   if (n % 2 == 1)
      return x*half*half;

   return half*half;
}

unsigned long goldLucky(unsigned long n)
{
    float gold = (1 + sqrt(5))/2;
    return recpow(gold, n) + recpow(1 - gold, n);
}
```

(e) (2 points) In 20 words or less, describe why are we using `float` as the return type for `recpow` instead of `unsigned int` or `unsigned long`.

(f) (3 points) Write a recurrence relation to describe the running time of the function `recpow`.

(g) (4 points) Find a closed form solution for the recurrence relation found in part (f).

Write the closed form here:

(h) (2 points) What is a big-O bound for the running time for `goldLucky`? You may assume that the assignment operator, addition, subtraction, multiplication, division, and the `sqrt` function take constant time.

5. **[Running Lucky Numbers] - 7 points.**

   On the course website download the file `hw3.zip`. This file contains all the code snippets from this document. To run all versions of `lucky` presented here, run the following to compile your code, and replace `13` with a (luckier) number of your choice.

```
make
time ./lucky 13    # Use naive recursive formula
time ./lucky 13 -d # Use dynamic programming
time ./lucky 13 -g # Use Golden Ratio formula
```

   (a) (2 points) For each function, write down the **execution time**, in seconds, for that function for each value of n in the table below.

n	`lucky`	`dpLucky`	`goldLucky`	Expected Value
5				11
10				123
50				28143753123
100	Too slow!			792070839848372253127

   (b) (1 point) Do the running times of the functions above match with your big-O runtime bounds that you have found in this assignment?

   (c) (2 points) For which functions, and for which inputs n does the function **not** produce the correct value L_n? For example, if `lucky(5)` produced 10 instead of 11, include `lucky(5)` in your answer here.

   (d) (2 points) Can you describe one reason why some of the functions presented do not always produce the expected return value, even though we have proven that these functions return L_n correctly?

Blank sheet for extra work.

Blank sheet for extra work.