Ashcon Abae

Design and Analysis of Algorithms

October 27, 2019

Homework 1

1.
    a. Pentagonal from iterative to recursive:
```
//Recursive method that takes n as integer value
int pentagonal(int n)
{
   //Check if n is 0, return 0 if so
   if(n == 0)
      return 0;
   //Check if n is 1, return 1 if so
   if(n == 1)
      return 1;
   //Otherwise call pentagonal with n – 1 and add (3 * n – 2)
   else
      return 3 * n – 2 + pentagonal(n – 1)
}
```
    b. Proof by Induction that function above in part (a) is correct:

    Let's look at the case for n = 1:
    Our base case, based on the iterative approach given in the question, is 3 * 1 – 2 = 1
    The function I wrote above in part (a) returns 1 because it goes through the second if clause, thus returning 1

    Now, let's look at the case for n = 2:
    Our base case, based on the iterative approach given in the question, is (3 * 1 – 2) + (3 * 2 – 2) = 1 + 4 = 5
    The function I wrote above in part (a) returns 3 * 2 – 2 + pentagonal(1) = 3 * 2 – 2 + 1 = 5

    For both cases, the results of our base case align with the result gotten from my algorithm in part (a). Now, let's consider the recursive approach to be true for n = m. So for n = m + 1, our base case would be: result = (3*1-2) + (3*2-2) + … + (3*m-2) + (3*(m+1)-2) = pentagonal(m) + 3*(m+1)-2. For the algorithm I wrote above in part (a), our result would be: result = 3*(m+1)-2 + pentagonal(m) = (3*(m+1)-2) + (3*m-2) + pentagonal(m-1) = (3*(m+1)-2) + (3*m-2) + … + 1 + 0. As we can see, both algorithms will turn into pentagonal(m) + 3*(m+1)-2 for n=m+1, thus we can say both functions are identical by induction.

2. We will be able to determine at which point Algorithm 2 is more efficient than Algorithm 1 when the number of steps in Algorithm 2 is less than the number of steps in Algorithm 1.

Steps in Algorithm 2 < Steps in Algorithm 1:

$$(21n + 7) < (10n^2 + 6)$$
$$10n^2 + 6 - 21n - 7 > 0$$
$$10n^2 - 21n - 1 > 0$$
$$n < \frac{1}{20}\left(21 - \sqrt{481}\right) \text{ OR } n > \frac{1}{20}\left(21 - \sqrt{481}\right)$$

Thus, for the above 2 values $n < \frac{1}{20}\left(21 - \sqrt{481}\right)$ OR $n > \frac{1}{20}\left(21 - \sqrt{481}\right)$, algorithm 2 becomes more efficient than algorithm 1.

3. To determine the number of additions and multiplications that are performed in the worst case, let's dissect each iteration:
   For each iteration, we have 2 multiplication operations:
   $$\text{power} = \text{power} * \text{x}$$
   $$\text{coefficients[i]} * \text{power}$$
   Also, for each iteration, we have 1 addition operation:
   $$\text{result} = \text{result} + \text{coefficients[i]} * \text{power}$$
   Therefore, worst case would be n – 1 iterations for an array of length n. So the number of addition operations in the worst case would be n – 1 and the number of multiplication operations in the worst case would be 2(n – 1). That is,
   $$O(n - 1) + O\big(2(n - 1)\big) = O(n)$$

4.
   a. Our **Initial Condition** is when $n = 1$

   b. Because at each iteration the length of the function is reduced by half, we can describe the **Recurrence Equation** that expresses the execution time for the worst case of this algorithm is as follows:
   $$T(n) = \begin{cases} 1, & \text{if n} = 1 \\ T\left(\frac{n}{2}\right) + 1, & \text{otherwise} \end{cases}$$

   c. To solve this recurrence equation, let's write:
   $$T(n) = T\left(\frac{n}{2}\right) + 1$$
   $$\rightarrow T\left(\frac{n}{2}\right) = T\left(\frac{n}{4}\right) + 1$$
   $$\rightarrow T\left(\frac{n}{4}\right) = T\left(\frac{n}{8}\right) + 1$$
   $$\dots$$
   $$\rightarrow T(4) = T(2) + 1$$
   $$\rightarrow T(2) = T(1) + 1$$
   Next, we sum up both the left and right hand sides of the equations above:
   $$T(n) + T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + \cdots + T(4) + T(2)$$
   $$= T\left(\frac{n}{2}\right) + T\left(\frac{n}{4}\right) + T\left(\frac{n}{8}\right) + \cdots + T(2) + T(1)$$
   $$+ (1 + 1 + 1 + \cdots + 1 + 1)$$

The number of 1s on the right hand side of our equation above is $\log_2 n$ because our termination condition for k iterations is:

$$\frac{n}{2^k} = 1$$
$$k = \log_2 n$$

Substituting this back in and crossing out equal terms on opposite sides of the equation, we get:

$$T(n) = T(1) + \log_2 n = 1 + \log_2 n$$

Therefore, the running time of this binary search algorithm is

$$T(n) = O(\log n)$$