Bedemariam Degef

Design and Analysis of Algorithms

Homework 1

* 1. 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)  
     }
  2. 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.

1. 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:  
   Thus, for the above 2 values , algorithm 2 becomes more efficient than algorithm 1.
2. 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:  
   Also, for each iteration, we have 1 addition operation:  
   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,  
   1. Our **Initial Condition** is when
   2. 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:
   3. To solve this recurrence equation, let’s write:  
      Next, we sum up both the left and right hand sides of the equations above:  
      The number of 1s on the right hand side of our equation above is because our termination condition for k iterations is:  
      Substituting this back in and crossing out equal terms on opposite sides of the equation, we get:  
      Therefore, the running time of this binary search algorithm is