Student: Dakin Werneburg
Section 6381

Instructor: Susan Furtney
Date: January 15, 2019

CMSC 451 Homework 1

1. **Consider the following iterative function:**

```
int pentagonal(int n)
{
  int result = 0;
  for (int i = 1; i <= n; i++)
    result += 3 * i - 2;
  return result;
}
```

Rewrite the function `pentagonal` using recursion and add preconditions and postconditions as comments. Then prove by induction that the recursive function you wrote is correct.

Solution:

```
//precondition n >= 1

  int pentagonal(int n)
  {
    if(n==1)
        return 1;
    return  pentagonal (n-1) + 3*n-2
  }

//Postcondition Returns n(3n-1)/2
```

Proof by induction of n:

Base case: n=1
1(3*1-1)/2 = 1 True.

Inductive case: We assume that it is true for n = k-1, we must show it is true for n=k where n>1

return value = pentagonal(k-1) + 3*k-2. From program

$= \frac{(k-1)(3(k-1)-1)}{2} + 3k - 2$ By inductive hypothesis

$= (3k^2 - 7k + 4 + 6k - 4)/2$ By Algebra

$= (3k^2 - k)/2$ By Algebra

$= k(3k - 1)2$ By Algebra

Student: Dakin Werneburg
Section 6381

Instructor: Susan Furtney
Date: January 15, 2019

2. Suppose the number of steps required in the worst case for two algorithms are as follows:

- **Algorithm 1: $f(n) = 10n^2 + 6$**
- **Algorithm 2: $g(n) = 21n + 7$**

Determine at what point algorithm 2 becomes more efficient than algorithm 1.

Solution:

$$10n^2 + 6 = 21n + 7$$

$$10n^2 - 21n - 1 = 0$$ By Algebra

$$r_1, r_2 = \frac{21 \pm \sqrt{(-21)^2 - 4(10)(-1)}}{2(10)}$$ By Quadradic Equation

$$r_1, r_2 = \frac{21 \pm \sqrt{441 + 40}}{2(10)}$$

$$r_1, r_2 = \frac{21 \pm 21.93}{20}$$

$$r_1, r_2 = 2.15, -0.047$$

Therefore, when $n \geq 3$ algorithm 2 is more efficient than algorithm 1

Student: Dakin Werneburg
Section 6381

Instructor: Susan Furtney
Date: January 15, 2019

3. Given the following function that evaluates a polynomial whose coefficients are stored in an array:

```
double evaluate(double[] coefficients, double x) {

     double result = coefficients[0];
     double power = 1;
     for (int i = 1; i < coefficients.length; i++)        (n-1) add
     {
       power = power * x;                                 (n-1) mult
       result = result + coefficients[i] * power;         (n-1) add
                                                        + (n-1) mult
     }                                                    -------------
     return result;                                Total = 4n-4
   }
```

Let *n* be the length of the array. Determine the number of additions and multiplications that are performed in the worst case as a function of *n*.

Solution:

There are 2(n-1) additions and 2(n-1) multiplications

T(n) = (n-1) + (n-1) + (n-1) + (n-1) = 4n-4. Therefore, this function has a worst case of Big O(n).

Student: Dakin Werneburg
Section 6381

Instructor: Susan Furtney
Date: January 15, 2019

4. **Given the following recursive binary search algorithm for finding an element in a sorted array of integers:**

```
int recursiveBinarySearch(int[] array, int target, int left, int right)
{
   if (left > right)
      return -1;                                                           O(1)
   int middle = (left + right) / 2;
   if (array[middle] == target)
      return middle;                                                       O(1)
   if (array[middle] > target)
      return recursiveBinarySearch(array, target, left, middle - 1);   T(n/2)
   return recursiveBinarySearch(array, target, middle + 1, right);     T(n/2)
}
```

Assume *n* is the length of the array. Find the initial condition and recurrence equation that expresses the execution time for the worst case of this algorithm and then solve that recurrence.

Solution:

Since only one of the if-else statements will be executed, we will have O(1) for the initial condition and O(1) for the check to see if middle is the target. And only one of the recursive calls that uses divide-and-conquer to search either the left or right side of the tree which will result in T(n/2).

Since any constant is just O(1) time complexity, we get O(1) after adding all constants.

Initial Condition:

$T(1) = O(1) = 1$, for $n \leq 1$;

Recursive Equation**:** $T(n) = T(n/2) + O(1)$, for $n > 1$

Using the Master Theorem:

If $T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$ for constants a>0, b>1, d>= 0, then:

$$T(n) = \begin{cases} O(n^d), & if\ d > log_b a \\ O(n^d \log n), & if\ d = log_b a \\ O(n^{log_b a}), & if\ d < log_b a \end{cases}$$

$O(1) = O(n^0)$ a = 1, b = 2, d = 0 $log_2 1 = 0$

Therefore, T(n) = O (log n)