**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 > 0

int pentagonal(int n)

{

if (n == 1)

return 1;

else

// Adds 3\*(n-2) for each Recursive Iteration

return 3 \* n – 2 + pentagonal(n – 1);

}

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

Proof by Induction:

The pentagonal function returns the values (1, 5, 12, 22, 35…)

The recurrence relation is:

1) Base Case (n = 1)

When n = 1, 1 is returned

2) Inductive Case, n > 1

Assume for n = k – 1, we must show from n = k

) Returned from function

Inductive hypothesis

Algebra

Algebra

Algebra

Algebra

Recurrence relation

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

Algorithm 1: *f*(*n*) = 10*n*2 + 6

Algorithm 2: *g*(*n*) = 21*n* + 7

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

**SOLUTION**

The point at which one algorithm becomes more efficient than another is where they intersect. Thus, we should begin by equating the two algorithms to each other.

10n2 + 6 = 21n + 7

10n2 – 21n – 1 = 0

Therefore, algorithm 2 is more efficient than algorithm 1 when n ≥ 3.

1. 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++)

{

power = power \* x;

result = result + coefficients[i] \* power;

}

return result;

}

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**

double evaluate(double[] coefficients, double x) Worst Case

{

double result = coefficients[0]; 1

double power = 1; 1

for (int i = 1; i < coefficients.length; i++) n-1

{

// FOR EACH ITERATION

power = power \*(MUL) x; n-1

result = result + (ADD) coefficients[i] \* (MUL) power; 2(n-1)

**// Multiplication occurs twice 2(n-1)**

**// Addition occurs once (n-1)**

}

return result; 1

}

Worst Case Time Complexity = 2(n - 1) + (n - 1) + (n – 1) + 3

= 2n – 2 + n - 1 + n - 1 + 3

= 4n - 1

= O(n)

1. 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;

int middle = (left + right) / 2;

if (array[middle] == target)

return middle;

if (array[middle] > target)

return recursiveBinarySearch(array, target, left, middle - 1); return recursiveBinarySearch(array, target, middle + 1, right);

}

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**

Initial Condition:

if (left > right)

return -1;

This condition is met when the recursive binary search in unable to find the element in the array of integers.

Recurrence Equation for Binary Search Algorithm:

Solving Recurrence:

Using Master Theorem:

Thus, the execution time for the worst case of this algorithm is:

**Grading Rubric**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Problem** |  |  | **Meets** |  | **Does Not Meet** |  |
|  |  |  |  | **10 points** |  | **0 points** | |
|  |  |  |  | |  |  | |
|  |  |  |  | |  |  | |
|  |  |  | Recursive version correctly written (3) | |  | Recursive version not correctly written | |
|  | **Problem 1** |  |  |  | (0) | |  |
|  |  |  |  |  |  |  |
|  |  | Precondition is correct (1) | |  | Precondition is not correct (0) | |
|  |  |  |  |
|  |  |  |  | |  |  | |
|  |  |  | Postcondition is correct (1) | |  | Postcondition is not correct (0) | |
|  |  |  |  | |  |  | |
|  |  |  | Base case is correct (1) | |  | Base case is not correct (0) | |
|  |  |  |  | |  |  | |
|  |  |  | Inductive case is correct (4) | |  | Inductive case is not correct (0) | |
|  |  |  |  |  |  |  | |
|  |  |  |  | **10 points** |  | **0 points** | |
|  |  |  |  | |  |  | |
|  |  |  |  | |  |  | |
|  | **Problem 2** |  | Determined correct value for n (5) | |  | Did not determine correct value for n | |
|  |  |  |  | (0) | |  |
|  |  |  |  |  |  |
|  |  |  |  | |  |  | |
|  |  |  | Show calculation for determining n (5) | |  | Did not show calculation for | |
|  |  |  |  |  |  | determining n (0) | |
|  |  |  |  |  |  |  | |
|  |  |  |  | **10 points** |  | **0 points** | |
|  |  |  |  | |  |  | |
|  |  |  |  | |  |  | |
|  | **Problem 3** |  | Provided correct formulas for additions | |  | Did not provide correct formulas for | |
|  | (5) | |  |  | additions (0) | |
|  |  |  |  |
|  |  |  |  | |  |  | |
|  |  |  | Provided correct formulas for | |  | Did not provide correct formulas for | |
|  |  |  | multiplications | (5) |  | multiplications (0) | |
|  |  |  |  |  |  |  | |
|  |  |  |  | **10 points** |  | **0 points** | |
|  |  |  |  | |  |  | |
|  |  |  |  | |  |  | |
|  |  |  | Provided correct initial condition (2) | |  | Did not provide correct initial condition | |
|  | **Problem 4** |  |  |  | (0) | |  |
|  |  |  |  |  |  |  |
|  |  | Provided correct recurrence equation | |  | Did not provide correct recurrence | |
|  |  |  |  |
|  |  | (4) | |  |  | equation (0) | |
|  |  |  |  | |  |  | |
|  |  |  | Provided correct solution to recurrence | |  | Did not provide correct solution to | |
|  |  |  | equation (4) |  |  | recurrence equation (0) | |