CPSC 331 Assignment 1

Name: Egor Ushakov 30009589, Donald McEachern 10139108

October 6, 2019

1. Give a reasonably short proof that the function $f(n) = n$ is a bound function for this recursive algorithm.

   Proof. To prove the function $f(n) = n$ is the bound function for the SmacG algorithm, we must show that

   (a) $f(n) = n$ is an integer valued function [1]

   (b) When the SmacG is applied recursively, the value of the function, $f(n) = n$, has been decreased by at least one [1].

   (c) If $f(n) \leq 0$ for some $n \geq 0$ then the SmacG algorithm does not call itself recursively during this execution [1].

   To show (a), we start by assuming that $f(n) = n$, where n is a non-negative integer. Notice $f : \mathbf{N} \to f(\mathbf{N})$, where $f(\mathbf{N})$ is the range of f, and $\mathbf{N}$ is the set of natural numbers. By the definition, $f(\mathbf{N}) = \mathbf{N} \subset \mathbf{Z}$, where $\mathbf{Z}$ is the set of all integers. Therefore, the function $f(n) = n$ maps the set of all natural numbers to itself, which are integers. This establishes that f is an integer-valued function.

   To prove (b), we want to show that the value of f is decreased by at least once when the algorithm is applied recursively. One can easily see by inspecting the code that the algorithm is applied recursively only if $n \geq 4$. If this is the case, the algorithm is applied recursively at line 9 with input values $n-1$, $n-3$, and $n-4$. Therefore,

   $$f(n-4) = n-4 < f(n-3) = n-3 < f(n-1) = n-1 < f(n) = n$$

   This shows the value of the function has been decreased by at least once when the algorithm is applied recursively, which proves (b)

   Lastly, we show that (c) holds by contradiction. For the sake of contradiction, suppose that the SmacG algorithm calls itself recursively for input $n \geq 0$. It is only possible for $f(n) \leq 0$ if and only if $f(n) = 0$. It follows that when $f(n) = 0$, the value of $n = f^{-1}(0) = 0$, and the algorithm calls itself recursively by the assumption. However, when $n = 0$ the test on line 1 of the smacG algorithm is executed and passes, which continues the execution to line 2, where the smacG number, $\mathcal{M}_0 = 0$, is returned as output. This is a contradiction, since an integer, $\mathcal{M}_{\prime} = 0$, is returned as output instead of the algorithm itself. Therefore, the smacG algorithm doesn't call itself recursively for input $f(n) = 0$ and it is the only value for which $f(n) \leq 0$, as $f(n) = n \geq 0$ by the precondition. This establishes (c).

The function $f(n) = n$, therefore, is a bound function of the recursive smacG algorithm because it satisfies parts (a), (b), and (c) □

2. Prove that the smacG algorithm correctly solves the ”MacGonagall Mystery Computation” problem.

   Proof. In order to prove the smacG algorithm correctly solves the ”MacGonagall Mystery Computation” problem, we must show that the execution of this algorithm eventually ends and the *nth* MacGonagall number, $\mathcal{M}_n$, is returned as output when this happens, with the assumption that the precondition is satisfied [1]. Before we do, certain assumptions have to be made. The first assumption that needs to be made is that integer arithmetic is correct and exact when included in the execution of an algorithm. The second assumption that needs to be made is that equality tests for integers are performed correctly during the execution of an algorithm.

   Since what we desire to prove is to be proved by induction in the strong form, we must consider the choice of base cases. The base cases selected here are $n = 0$, $n = 1$, $n = 2$, and $n = 3$. These cases are considered in the basis.

   Basis: Suppose that $n = 0$. Then, during the execution of the smacG algorithm on input n, the test at line 1 is checked and passed. This causes the execution to continue with line 2, where it terminates, and the MacGonagall number, $\mathcal{M}_0 = 1$, is returned as output when this happens. In this case, the post-condition is satisfied with no undocumented side effects.

   Suppose, next, that $n = 1$. Then, during the execution of the smacG algorithm on input n, the test on line 1 fails, since $n \neq 0$. This causes the execution to continue to line 3, where the test is checked and passed. As a result of this, line 4 is executed which terminates the execution of the algorithm, and the output value of $\mathcal{M}_1 = 0$ is returned when this happens. Therefore, the post-condition is satisfied with no undocumented side effects.

   Suppose, instead, that $n = 2$. During the execution of the smacG algorithm on input n, the test at line 1 is checked and failed. This causes the execution of the algorithm to continue with line 3, where the test also checked and failed. As a consequence, the execution continues with line

5. On this line, the test is checked and passed, so the execution continues with line 6, where the execution terminates, and the MacGonagall number, $\mathcal{M}_2 = 5$, is returned as output. Hence, the post-condition is satisfied with no undocumented side effects.

Finally, suppose that $n = 3$. Since this is the case, the tests at lines 1, 3, and 5 are checked and failed. As a result, the test on line 7 is passed and checked, which results in the the execution of the algorithm to continue to line 8. On this line, the execution of the algorithm terminates, and the MacGonagall number, $\mathcal{M}_3 = 8$, is returned as output when this happens. Therefore, the post-condition is satisfied with no undocumented side effects in this case.

Inductive Step: Let k be an arbitrary integer such that $k \geq 3$. It is enough to use the inductive hypothesis to prove the following inductive claim:

Inductive Hypothesis: Suppose that n is a nonnegative integer such that $0 \leq n \leq k$. If the algorithm smacG is executed given n as input, then this execution of the algorithm eventually ends, and the *nth* MacGonagall number is returned as output when this happens.

Inductive Claim: If the smacG algorithm is executed on $n = k+1$ input, then this algorithm eventually terminates, and the $k + 1^{st}$ MacGonagall number is returned as output.

To show that the inductive claim holds, first suppose that $n = k+1$ is given as input. Since $k \geq 3$ by the inductive step, it follows that $n = k + 1 \geq 4$. Considering that $n \geq 4$, the tests at lines 1, 3, 5, and 7 are all checked and failed. This causes the execution of the algorithm to continue with line 9.

Notice that

$$n = k + 1 \geq 4 \implies k = n - 1 \geq 3 \implies 3 \leq n - 1 = k \leq k \tag{2.1}$$

$$n = k + 1 \geq 4 \implies k - 2 = n - 3 \geq 1 \implies 1 \leq n - 3 = k - 2 \leq k \tag{2.2}$$

$$n = k + 1 \geq 4 \implies k - 4 = n - 4 \geq 0 \implies 0 \leq n - 4 = k - 3 \leq k \tag{2.3}$$

By (2.1) and the inductive hypothesis, it follows that the recursive execution of the algorithm for the $k - 1^{st}$ input eventually ends on line 9 when executed, and $\mathcal{M}_k = \mathcal{M}_{n-1}$ is returned as output.

The same reasoning is applied for the $k - 2^{nd}$ input. By (2.2) and the inductive hypothesis, the recursive execution of the algorithm also ends on line 9, and $\mathcal{M}_{k-2} = \mathcal{M}_{n-3}$ is returned as output when this happens.

Finally, by (2.3) and the inductive hypothesis, the recursive execution of the algorithm for the $k-3^{rd}$ input eventually ends as well on line 9, and the $\mathcal{M}_{k-3} = \mathcal{M}_{n-4}$ is returned as output.

By the examination of line 9, one can see the execution of the algorithm eventually ends. Furthermore, since $k+1 \geq 4$, it follows that by the definition of $\mathcal{M}_{k+1}$ that the value returned as output is

$$\mathcal{M}_{k+1} = 2\mathcal{M}_k - 2\mathcal{M}_{k-2} + \mathcal{M}_{k-3} = 2\mathcal{M}_{n-1} - 2\mathcal{M}_{n-3} + \mathcal{M}_{n-4}$$

Therefore, the inductive proof is established.

By the basis, the inductive hypothesis, and the inductive claim, it follows that the execution of this algorithm for a non-negative n input ends with the post-condition satisfied. Furthermore, this algorithm has no undocumented side effects. Since the algorithm's post-condition is satisfied when it terminates and it has no undocumented side effects, then this algorithm is correct. Thus, the desired result is proved. □

3. This is a programming question. Please java code for this part of the assignment.

4. Let $T_{smacG}(n)$ be the number of steps included in the execution of the algorithm, smacG, shown in Figure 1 on input n, for a non-negative integer n— assuming that the uniform cost criterion is used to define this and then only steps counted are the numbered steps shown in Figure 1. Give a recurrence for $T_{smacG}(n)$. Note: If your answer is correct then you should be able to use your recurrence to show that $T_{smacG}(0) = 2$, $T_{smacG}(1) = 2$ $T_{smacG}(2) = 4$, $T_{smacG}(3) = 5$, $T_{smacG}(4) = 15$, and $T_{smacG}(5) = 27$.

   SOLUTION: Let $T_{smacG}(n)$ be the number of steps included in the execution of the smacG algorithm. To derive a recurrence for $T_{smacG}(n)$, we break the smacG algorithm down into five cases for when $n = 0$, $n = 1$, $n = 2$, $n = 3$, and $n \geq 4$, and count the total number of steps executed in each case using the uniform cost criterion [3].

   When $n = 0$, the test on line 1 in the *smacG* algorithm is checked and passed when executed. When this happens, the execution of this algorithm continues with line 2, where the execution is terminated, and the MacGonagall number, $M_0 = 1$, is returned as output. Since each step executed follows the uniform cost criterion, the total number of steps included in this case is 2. Therefore, we have $T_{smaG}(0) = 2$

   Next, we consider the case when $n = 1$. Since $n \neq 0$, the test at step 1 in the algorithm is checked and failed when executed. This causes the execution of the algorithm to continue with line 3, where the test on this line is checked and passed. As a result, line 4 is then executed where the algorithm terminates, and the MacGonagall number, $M_1 = 0$, is returned as output when this happens. Notice the number of steps executed in this case is 3, so $T_{smaG}(1) = 3$

Suppose, next, that $n = 2$. It can be seen that the tests at lines 1 and 3 are checked and failed when executed. With this being said, the execution of the algorithm continues with line 7. On this line, the test is checked and passed, so line 8 in the algorithm is executed, which causes the algorithm to halt, and the MacGonagall number, $M_2 = 5$, is returned as output when this happens. In this case, the total number of steps executed is 4, so $T_{smaG}(2) = 4$.

When $n = 3$, the tests at lines 1, 3, and 5 are checked and failed. When this happens, the execution of the algorithm continues with line 7, where the test on this line is checked and passed. As a consequence, the execution of the algorithm ends and the output MacGonagall number, $M_3 = 8$, is returned as output when this occurs. The total number of steps executed when $n = 3$ is clearly 5. Therefore, we have $T_{smaG}(3) = 5$ as desired.

Lastly, consider the case when $n \geq 4$. One can see right away just by inspecting the code that the tests at lines 1, 3, 5, and 7 are checked and failed. As a result, the execution of the algorithm continues on with line 9. On this line, the algorithm calls itself recursively with integer inputs $n-1$, $n-3$, and $n-4$. Since 5 steps are executed in the current iteration of the algorithm along with the three recursive calls, the total number of steps executed when using the uniform cost criterion is $T_{smaG}(n) = 5 + T_{smaG}(n-1) + T_{smaG}(n-3) + T_{smaG}(n-4)$
Therefore, the recurrence is given by

$$T_{smacG}(n) = \begin{cases} 2 & \text{n} = 0 \\ 3 & \text{n} = 1 \\ 4 & \text{n} = 2 \\ 5 & \text{n} = 3 \\ 5+\text{T}_{smacG}(n-1) + T_{smacG}(n-3) + T_{smacG}(n-4) & \text{n} \geq 4 \end{cases}$$

5. Use your recurrence to prove that $T_{smacG}(n) \geq (3/2)^n$ for every non-negative integer n. Thus the number of steps executed by the smacG algorithm is at least exponential in its input

   Proof. We use strong form of mathematical induction to prove that the inequality, $T_{smacG}(n) \geq (3/2)^n$, for all $n \geq 0$. By how $T_{smacG}(n)$ is defined, we must consider the natural numbers $n = 0$, $n = 1$, $n = 2$, and $n = 3$ as base cases. These base cases are considered in the basis.

Basis: Suppose that $n = 0$. Then

$$T_{smacG}(0) = 2 = \frac{6}{3} \geq (3/2)^0 = 1 = \frac{3}{3}$$

since $6 > 3$. This shows the inequality above holds when $n = 0$.

Similarly, if $n = 1$ then by the definition,

$$T_{smacG}(1) = 3 = \frac{6}{2} \geq (3/2)^1 = \frac{3}{2}$$

since $6 > 3$. As a result, the inequality holds for this case as well.

Suppose, now, that $n = 2$. It follows by the definition that

$$T_{smacG}(2) = 4 = \frac{16}{4} \geq (3/2)^2 = \frac{9}{4}$$

since $16 > 9$. Therefore, the inequality in this case is satisfied.

Suppose that $n = 3$. By the definition,

$$T_{smacG}(3) = 5 = \frac{40}{8} \geq (3/2)^3 = 27/8$$

since $40 > 27$. Therefore, the inequality for this case holds.

Inductive Step: Suppose that k is an arbitrary integer such that $k \geq 3$. It is sufficient to use the inductive hypothesis to prove the following inductive claim.

Inductive Hypothesis: $T_{smacG}(s) \geq (3/2)^s$ for every integer s such that $0 \leq s \leq k$

Inductive Claim: If $s = k + 1$ then $T_{smacG}(k + 1) \geq (3/2)^{k+1}$

Firstly, notice that $s = k + 1 \geq 4$ by the inductive step. By the definition of $T_{smacG}(n)$ and this inequality, it follows that

$$T_{smacG}(k + 1) = 5 + T_{smacG}(k) + T_{smacG}(k - 2) + T_{smacG}(k - 3)$$

.

Notice that

$$k + 1 \geq 4 \implies k = s - 1 \geq 3 \implies 3 \leq s - 1 \leq k \qquad (5.1)$$

$$k + 1 \geq 4 \implies k - 2 = s - 3 \geq 1 \implies 1 \leq s - 3 \leq k \qquad (5.2)$$

$$k + 1 \geq 4 \implies k - 3 = s - 4 \geq 0 \implies 0 \leq s - 4 \leq k \qquad (5.3)$$

By (5.1), (5.2), (5.3), and the inductive hypothesis, we have

$$T_{smacG}(s-1) \geq (3/2)^{s-1} \qquad (5.4)$$

$$T_{smacG}(s-3) \geq (3/2)^{s-3} \qquad (5.5)$$

$$T_{smacG}(s-4) \geq (3/2)^{s-4} \qquad (5.6)$$

It follows by (5.4), (5.5), and (5.6) that

$$\begin{aligned}
&5 + T_{smacG}(k) + T_{smacG}(k-2) + T_{smacG}(k-3) \\
&= 5 + T_{smacG}(s-1) + T_{smacG}(s-3) + T_{smacG}(s-4) \\
&\geq 5 + (3/2)^{s-1} + (3/2)^{s-3} + (3/2)^{s-4} \\
&\geq (3/2)^{s-1} + (3/2)^{s-3} + (3/2)^{s-4} \\
&= (1 + \frac{3}{2} + (\frac{3}{2})^3)(3/2)^{s-4} \\
&= (1 + \frac{3}{2} + \frac{27}{8})(3/2)^{s-4} \\
&= \frac{47}{8}(3/2)^{s-4} = \frac{94}{16}(3/2)^{s-4} \\
&\geq \frac{81}{16}(3/2)^{s-4} = (3/2)^4(3/2)^{s-4} = (3/2)^s = (3/2)^{k+1}
\end{aligned}$$

This establishes that $T_{smacG}(k+1) \geq (3/2)^{k+1}$, as required in the inductive claim.

By the basis, inductive hypothesis, and inductive claim, it follows that that $T_{smacG}(n) \geq (3/2)^n$ for $n \geq 0$. $\square$

6. State the **loop invariant** for the while loop at lines 15-17

   The following assertion, $\mathcal{A}$, is a loop invariant for the while loop in the fmacG algorithm.

   (a) n is an input value such that $n \geq 4$
   (b) M is a(variable) integer array with length $n+1$
   (c) i is an integer variable such that $4 \leq i \leq n+1$
   (d) $M[j] = \mathcal{M}_j$ for every j such that $0 \leq j \leq i-1$

7. Prove that your answer for the previous question really is a loop invariant for the while loop in this algorithm.

Proof. We want to prove the assertion, A, is indeed a loop invariant for the while loop in the fmacG algorithm. Since the loop test has no side effects, we use **Loop Theorem 1** to this. In order to be able to apply this theorem, we must first show that the remaining hypotheses of the theorem hold:

- The assertion, $\mathcal{A}$, is satisfied whenever the while loop is reached, with the assumption that the problem's precondition is satisfied [2]
- If the assertion, A, is satisfied at the beginning of any execution of the loops body then it is also satisfied when this execution of the loop body ends, assuming that the problem's precondition is satisfied [2].

Consider an execution of the fmacG algorithm that begins with its precondition satisfied. That is, n is an integer input such that $n \geq 4$.

Define $\mathbf{N}$ to be the set of non-negative integers. By this definition, this set is just the set of natural numbers. We proceed by defining two more sets, $S_1 := \{0, 1, 2, 3\}$ and $S_2 := \{4, 5, 6, ...\}$, that give the option of writing $\mathbf{N} = S_1 \cup S_2$. Since $n \geq 0$ by the precondition, it follows that n is either in S_1 or S_2, but not in both sets, since S_1 and S_2 are mutually exclusive by construction. In the case that the input value $n \in S_1$, then there is nothing to prove here, since the while loop is never reached.

On the other hand, suppose that $n \in S_2$. Then the tests on lines 1, 3, 5, and 7 are checked and failed, which cause the execution of the algorithm to continue with line 9. Notice the value of n remains unchanged when the while loop is reached. Therefore, $n \geq 4$ when the while loop, which establishes the first condition of A as required.

When line 9 is executed, M is declared to be an integer array of length $n+1$. Since n is unchanged when the loop is reached, then so is the length of M. The data type of the array is also not changed at this. Hence, the second condition of the assertion, A, holds when the while loop is reached.

From the execution of lines 10 to 13, the integer array M is declared to have values $M[0] = 1 = M_0$, $M[1] = 0 = M_1$, $M[2] = 5 = M_2$, and $M[3] = 8 = M_3$. Just before the while loop is reached on line 15, the variable i is declared to be an integer whose value is 4 when line 14 is executed. Since the values of $M[j]$ are not changed when the loop is reached, $M[j] = \mathcal{M}_j$ for every j such that $0 \leq j \leq i - 1 = 3$ when the loop is reached. Furthermore, i and n are not changed as well, so it is true that $4 \leq i \leq n + 1$ before the execution of the loop. Therefore, the fourth and third conditions are satisfied for the assertion A when the loop is reached.

Since all of the conditions in assertion A hold when the while loop is reached, the second hypothesis of the loop invariant is established.

All that remains to be shown is the third hypothesis of the theorem. That is, if the assertion A, is satisfied at the beginning of any execution of the loops body then it is also satisfied when this execution of the loop body ends.

First, we show the first condition of the assertion, $\mathcal{A}$, holds at the end of the execution of the body of the loop. We begin with the assumption that at the beginning of the execution of the body of the loop that n is an integer input such that $n \geq 4$. Then, lines 15 and 16 of the loop body do not change n when executed. Since these are all the steps executed in the body of the while loop, this implies that $n \geq 4$ at the end of the execution of the body of the loop. Thus, the first condition of the assertion $\mathcal{A}$ holds at the end of the execution of the body of the while loop.

Second, we show the second condition of the assertion, $\mathcal{A}$, holds at the end of the execution of the body of the loop. To do this, first suppose that M is an integer array with length $n + 1$ at the beginning of the execution of the body of the loop. Notice lines 15 and 16 of the body of the loop do not change either the type or the length of the array when executed. This establishes that M is not affected at the end of the execution of the body of the while loop. Therefore, the second condition of the assertion at the end of the execution of the body of the loop holds.

Third, we want to show that the third condition of the assertion, A, holds at the end of the execution of the body of the loop. To do this, we suppose the third condition of the assertion is satisfied at the beginning of the execution of the body of the while loop. Furthermore, the loop test, $i \leq n$, is checked and passed before this execution commences, which means that $4 \leq i \leq n$. In the body of the while loop, line 16 doesn't change the value of i when executed. On the other hand, step 17 of the body does. When this step is executed, the declared variables, i, increases by one. As a consequence of this, $5 \leq i + 1 = i \leq n + 1$. Since n remains unchanged at the beginning of the execution of the body of the while loop to the end, then $5 \leq i \leq n + 1$ at the end of the execution of the body of the loop. Therefore, the third condition of the assertion holds at the end of the execution of the body of the loop.

Fourth, we want to show that the fourth condition of the assertion, A, holds at the end of the execution of the body of the loop. In order to do this, suppose $M[j] = M_j$ for every integer j between 4 and $i - 1$ at the beginning of the execution of the body of the while loop. Again, the loop test on step 15 must be checked and passed before the execution continues on with step 16. This implies that i is an integer such that $4 \leq i \leq n$ at the beginning of the execution of the loop body. After line 16 is executed, $M[i] = 2M[i-1] - 2M[i-3] + M[i-4] = 2\mathcal{M}_{i-1} - 2\mathcal{M}_{i-3} + M_{i-4} = M_i$, and the execution continues with line 17. On this line, i is increased by one, which means $5 \leq i = i+1 \leq n+1$ and $M[j] = \mathcal{M}_j$ for every integer such that $0 \leq j \leq i-1$ at the end of execution of the the body of the loop. Therefore, part 4 of the assertion, A, at the end of the execution of the body of the loop.

Since all the four conditions of the loop invariant hold at the beginning of the execution of the body of the while loop to the end of the execution of the loop body and the problem's precondition is satisfied by the first condition of the while loop, the third hypothesis of the theorem is established. Now, that all three hypotheses of the **Loop Theorem 1** are satisfied, we use this theorem to establish that the assertion, A, is indeed loop invariant. $\square$

8. Use this to prove that this algorithm is partially correct.

   Proof. To show the $fmacG$ algorithm is partially correct, we must establish that

   (a) this execution of the algorithm eventually ends with the problem's post-condition satisfied with no undocumented inputs or global data accessed, and no undocumented data modified [2]
   or

   (b) this execution of the algorithm never halts [2]

   Consider an execution of the $fmacG$ algorithm that begins with the precondition for the MacGonagall Mystery problem satisfied. This means that input n is an integer such that $n \geq 0$, so either $n = 0$, $n = 1$, $n = 2$, and $n = 3$, or $n \geq 4$.

   Suppose that $n = 0$ to start off. If this is the case then the test on line 1 passes, so the execution continues with line 2. When this step is executed, the execution of the algorithm terminates, and the MacGonagall number, $\mathcal{M}_0 = 1$, is returned as output. As a result, the problem's post-condition is satisfied with no side effects.

   Suppose, next, that $n = 1$. Since $n \neq 0$, the execution of the test of the algorithm on step 1 fails and continues on to step 3. On this step, the test passes and this causes the execution of the algorithm to eventually end, and the MacGonagall number, $M_1 = 0$, is returned as output. The problem's post condition is satisfied in this case as well.

   Suppose, now, that $n = 2$. By tracing the execution of the algorithm, it can be easily seen that the test on line 1 fails, as $n \neq 0$. This causes the execution to continue with line 3. On this line, the test fails as well, and the execution of the algorithm continues on with line 5. When this line of code is executed, the execution of the algorithm continues to line 6, where the algorithm terminates, and the MacGonagall number, $M_2 = 5$, is returned as output. In this case, the problem's post-condition is satisfied,

If it is the case that $n = 4$, then the tests at lines 1, 3, and 5 are checked and failed, which causes the execution of the algorithm to continue on with line 7. On this line, the test is checked and passed, and line 8 is executed. This line causes the algorithm to halt, and return the MacGonagall number, $M_3 = 8$, as output. As a result, the problem's post-condition is satisfied when this happens.

Finally, suppose that $n \geq 4$. Then the tests at lines 1, 3, 5, and 7 are checked and failed. The execution of the algorithm then continues on with line 9. After the execution of lines $9 - 14$, the while loop is reached and executed. There are two cases to consider when the while loop is reached: the first is (a) and the other one is (b). We first consider (a) and then (b).

In order for the execution of the algorithm to continue with the body of the while loop, the loop test on line 15 must have been checked and passed for all the integers i such that $4 \leq i \leq n$. It follows by the second condition of the loop invariant that $4 \leq i \leq n + 1$ before the beginning of the execution of the while loop, at the beginning of the execution of the body of loop and the end, and after the execution of the while loop. On the other hand, the loop test must have been checked and failed for $i > n$. By the first condition of the loop invariant, the integer input value n is such that $n \geq 4$. Since n is not changed and i is also an integer such that $4 \leq i \leq n + 1$, it follows that $i = n + 1$, so that $M[j] = \mathcal{M}_j$ for all integers j such that $0 \leq j \leq i - 1$. This is indeed true since the second condition of the loop invariant states that M is an integer array whose length is $n + 1$. Recognizing that the statement, $M[j] = \mathcal{M}_j$ for all integers j such that $0 \leq j \leq i - 1$, is the fourth condition of the loop invariant, it follows that there is no undocumented side effects when the execution of the algorithm terminates on line 18. Simultaneously, the *nth* MacGonagall number is returned as output, which satisfies the problem's post-condition. Therefore, part (a) is established.

All that remains to be shown is case (b). If the execution of the while loop does not halt then the execution of the fmacG algorithm does not either. There is nothing to prove here if this is the case and we are done.

Since parts (a) and (b) hold, the fmacG algorithm is partially correct. $\square$

9. State a bound function for the while loop in this algorithm and prove that your answer is correct.

   $f(n, i) = (n + 1) - i$ is the bound function for the loop in the algorithm fmacG.

Proof. In order to show that f is a bound function for the fmacG algorithm, we must show that

(a) f is a integer valued function and does not affect any global data or variables [2]

(b) when the body of the loop is run that the value of the function is decreasing by at least 1 [2].

(c) in the case that $f \leq 0$ that the failure of the condition to run the loop ends the loop [2].

We begin by proving that the function f is indeed an integer-valued function. Firstly, we know that for the algorithm to commence the precondition that is given indicates that the value passed in n needs to be a non-negative integer. If this is the case then n is a non-negative integer. All that remains to be shown for (a) is that i is also a non-negative integer. In order for i to be instantiated as an integer in the algorithm, line 14 needs to be executed. This is only possible if the value of n is greater than or equal to 4. Otherwise, this line in the algorithm is never reached at all. With that being stated, i is instantiated as an integer whose value is 4 on line 14 if $n \geq 4$. Since we know that n and i are both integers, we need to show that within the body of the loop that the two values do not change any data outside the algorithm or the loop. For n, this can be clearly seen because n is never in any of the lines passed a new value either in the loop or in the algorithm in general, only used a conditional comparison in line 15 and a index return from the array $M[]$ in line 18. In the case of i, we can see that it does not affect any values or data else where since it is instantiated in line 14 as a integer value, and increased by one in line 17. When the loop test on line 15 is executed, i is compared to n in the loop test in line 15. As a result, it is not reassigning any values including n. Therefore, we know it to be bound only to the algorithm. Therefore, f to be a integer-valued function that does not affect any global data or variables

Next, we show that when the body of the loop is run, the function f is indeed decreasing by at least 1. To show this, we need line 14 in the algorithm to be executed, where the variable, i, is declared to be an integer whose value is 4. This can only happen when $n \geq 4$. Otherwise, line 14 is never reached at all. In the case when $n \geq 4$, i is declared be an integer whose value is 4 when line 14 is executed. After this line is executed, the execution of the algorithm continues on with line 15. On this line, the loop test is checked and passed, since $n \geq 4 = i$, so the execution of the algorithm continues on with the body of the loop. When line 17 in the body of the loop is executed, the value of i is increased by one. By the first condition of the loop invariant, n is unchanged at the beginning of the execution of the loop and at the end of the execution of the body of the loop. Since n is unchanged and i is incremented by one in the loop body, the function, $f(n,i) = (n+1) - i$, will decrease by at least one, as required in (b).

Finally, we need to show that when the value of f is less than or equal to zero, the while loop will indeed end. In order for the execution of the fmacG algorithm to reach the while loop, n needs to be an integer such that $n \geq 4$. Otherwise, the loop will never be reached at all. In the case when $n \geq 4$, the while loop is reached on line 15. Notice that the body of the loop is only executed when $4 \leq i \leq n$ by line 15. On the other hand, the body of the loop is not executed when $i > n$ on line 15. By the second condition of the loop invariant, $4 \leq i \leq n+1$ at the beginning of the execution of the while loop to the end. This implies that the body is not executed only when $i = n+1$. Therefore,

$$f(i,n) = n + 1 - i \leq 0 \implies n + 1 \leq i \implies i > n \qquad (9.1)$$

It now follows by (9.1) that the loop test will fail when $f \leq 0$ on line 15 and the while loop will terminate as a result. Therefore, (c) is established.

It is from these three conditions thus being met; f being a integer valued function, f decreasing by at least 1 with every iteration of the loop, and from the loop terminating when $f \leq 0$ that we know $f(n,i) = (n+1) - i$ to be the bound function for the loop.

□

10. Prove that if this algorithm is executed when the precondition for the "MacGonagall Mystery Computation" problem is satisfied, and the while loop is reached and executed, then the execution of the loop eventually ends.

Proof. Since the loop test on line 15 does not change the values of inputs, local variables, or global data, we will use **Loop Theorem 2** from class to show that the execution of the while loop in the fmacG algorithm eventually ends [2]. In order to be able to apply this theorem, we must show that

(a) a bound function for the while loop in the algorithm exists [2]

(b) if the "MacGonagall" Mystery Computation" problem's is satisfied when an execution of the algorithm including this loop begins, then every execution of the loop body ends [2].

Since $f(i, n) = n + 1 - i$ is proved to be a bound function for the fmacG algorithm above, there is nothing to prove here. Therefore, (a) is established immediately.

All that remains to be shown is (b). That is, every execution of the body of the while loop in the fmacG algorithm ends.

Consider an execution of the fmacG algorithm when the precondition is met. That is, n is an integer input such that $n \geq 0$.

Suppose that the value of $n = 0$. In this case, the test on line 1 is checked and passed, so the execution of the algorithm continues on with line 2, where the algorithm terminates, and the MacGonagall number, $M_0 = 1$, is returned as output.

Suppose, next, that $n = 1$. The test at line 1 is checked and failed, so the execution of the algorithm continues with line 3. On this line, the test is checked and passed, which causes line 4 to be executed, where the algorithm terminates, and the MacGonagall number, $M_1 = 0$, is returned as output when this happens.

Suppose, next, that $n = 2$. The tests at lines 1 and 3 are checked and failed, which causes the execution of the algorithm to continue on with line 5. On this line, the test is checked and passed, so the execution continues on with line 6, where the algorithm terminates, and the MacGonagall number, $M_2 = 5$, is returned as output when this happens.

Suppose, next, that $n = 3$. The tests at lines 1, 3, and 5 are checked and failed, so the execution of the algorithm continues with line 7. On this line, the test is checked and passed, so the execution of the algorithm continues on with line 8, where the algorithm terminates, and the MacGonagall number, $M_3 = 8$, is returned as output when this happens.

Finally, suppose that $n \geq 4$. Then the tests at lines 1, 3, 5, and 7 are checked and failed, which causes the execution of the algorithm to continue with line 9. When lines $9 - 13$ are executed, the variable, M, is declared to be an integer array with length $n + 1$ such that $M[0] = 1$, $M[1] = 0$, $M[2] = 5$, and $M[3] = 8$. Just before the while loop is reached on line 15, the variable, i, is declared to be an integer whose value is 4 when line 14 is executed. When the while loop is eventually reached on line 15, the loop test is checked and passed on this line, since $n \geq 4 = i$, which causes the execution of the algorithm to continue with the body of the loop. The algorithm will then repeat the steps on lines 16 and 17 until the loop condition is broken when $i > n$, which in the case given is after just one execution. When the loop test is checked and passed, the execution of the while loop ends and continues on with line 18, where the algorithm terminates, and the *nth* MacGonagall number is returned as output when this happens. We can also see that from the steps that the only two variables being changed are i and M, which are created within the algorithm. Thus, the algorithm has no affect on the external variables and global data that might be surrounding it. Therefore, (c) is established.

Since all of the hypotheses of **Loop Theorem 2** are satisfied, it now follows by this theorem that every execution of this while loop during an execution of the fmacG algorithm, starting with an integer input n such that $n \geq 0$, will eventually end as desired. □

11. Using what has been proved so far, complete a proof that the fmacG algorithm correctly solves the "MacGonagall Mystery Computation" problem.

    To prove that the algorithm correctly solves the problem, we need to make sure that the algorithm satisfies two properties to meet the definition of correct.

    (a) That the algorithm is partially correct

    (b) That the algorithm terminates

    Proof. From question 8 above, we do know that the algorithm is partially correct. That is, when the precondition of the fmacG algorithm is satisfied, it either ends with the algorithm's post-condition satisfied and no other global variables or data changed or, it never ends at all. From question 10, we see that the algorithm indeed does terminate without affecting any of the global data and variables that might be present. Therefore, the properties, (a) and (b), are satisfied. Since these two properties are satisfied, the algorithm meets the definition of correctness, that is to say that; it is seen to return the correct post-condition when the precondition is met and that without changing any variables or global data terminates in a finite number of steps. Thus, we see fmacG is indeed proven to be correct. □

12. Using techniques introduce in this course, give an upper bound for $T_{fmacG}(n)$, as a function of n, that is as precise as you can. Your final answer should be "in closed form", so that it does not include any summations or recurrences. SOLUTION:

    In order to derive an upper bound for $T_{fmacG}(n)$, as a function of n, we break the fmacG algorithm into five parts for when $n = 0$, $n = 1$, $n = 2$, $n = 3$, and $n \geq 4$, and count the total number of steps executed in each case using the uniform cost criterion.

    Suppose that $n = 0$. Then the test on line 1 is executed and passed, so the execution of the algorithm continues with line 2, where the algorithm terminates, and the MacGonagall number, $\mathcal{M}_0 = 1$, is returned as output. Since each line in the algorithm contains one step, the total number of steps executed in this case using uniform cost criterion is 2. Therefore, $T_{fmacG}(0) = 2$.

Suppose, next, that $n = 1$. The test at line 1 is checked and failed, so the execution of the algorithm continues with line 3. On this line, the test is checked and passed, so the execution of the algorithm continues to line 4, where the algorithm terminates, and the MacGonagall number, $\mathcal{M}_1 = 0$, is returned as output. Since each line in the algorithm contains one step, the total number of steps executed using uniform cost criterion is 3. As a result, we have that $T_{fmacG}(1) = 3$ for this case.

Suppose, next, that $n = 2$. Then the tests at lines 1 and 3 are checked and failed, so the execution of the algorithm continue with line 5, where the test is checked and passed. As a result, the execution of the algorithm continues on with line 6, where the algorithm ends, and the MacGonagall number, $M_2 = 5$, is returned as output when this happens. Since each line in the algorithm contains one step, the total number of steps executed using uniform cost criterion is 4. Thus, we have $T_{fmacG}(2) = 4$.

Suppose, next, that $n = 3$. The tests at lines 1, 3, and 5 are checked and failed, which causes the execution of the algorithm to continue with line 7. On this line, the test is checked and passed, so the execution of the algorithm continues with line 8, where the algorithm halts, and the MacGonagall number, $\mathcal{M}_3 = 8$, is returned as output when this happens. Since each line in the algorithm contains one step, the total number of steps executed is 5 in this case. This means that $T_{fmacG}(3) = 5$.

Finally, suppose that $n \geq 4$. Then the tests at lines 1, 3, 5, and 7 are checked and failed, so the execution of the algorithm continues with line 9. Just before the execution of the while loop continues on with line 15, lines $9 - 14$ are executed. Since each line in the algorithm contains one step, the total number of steps executed before the while loop is reached on line 15 is 10.

As noted above, the bound function for this algorithm is $f(n, i) = n+1-i$, and the initial value of this bound function will calculate the total number of executions of the loop body. By the first condition of the loop invariant, n is unchanged before the execution of the while loop all the way to after the execution of the while loop. This means that n is fixed. Furthermore, the second condition of the loop invariant states that i is an integer variable such that $1 \leq i \leq n + 1$ before the execution of the while loop all the way after the execution of the while loop. Since n is unchanged and $i = 4$ is the smallest value that i can take on, the initial value of the bound function is $f(n, 4) = n + 1 - 4 = n - 3$, which means the total number of executions of the body of the loop is $n - 3$. Furthermore, this initial value of the bound function implies that the loop test is executed $n-2$ times. Considering that there are two steps in the body of the loop(at lines 16 and 17), the total number of steps executed is

$$\sum_{i=1}^{n-3} 2 + \sum_{i=1}^{n-2} 1 = (2n-6) + (n-2) = (3n-8)$$

The last step executed in the fmacG algorithm is on line 18, where the algorithm halts, and the *nth* MacGonagall number, M_n, is returned as output when this happens. As a result, the total number of steps executed after the while loop is 1.

Adding all of the steps executed before the while loop, during the while loop, and after the while loop gives $10 + (3n-8) + 1 = 3n + 3$. Therefore, it follows that $3(n+1)$ is an upper bound for $T_{fmacG}(n)$ when $n \geq 4$, since $T_{fmacG}(n) \leq 3(n+1)$. In summary,

$$T_{fmacG}(n) = \begin{cases} 2 & \text{n = 0} \\ 3 & \text{n = 1} \\ 4 & \text{n = 2} \\ 5 & \text{n = 3} \\ \text{3n + 3} & \text{n} \geq 4 \end{cases}$$

for all $n \geq 0$.

13. This question is a programming question, so it can be ignored in this document

14. Find an expression for the *nth* MacGonagall number, as a function of n, in closed form— so that it does not include a summation or recurrence — and prove that your answer is correct.

    SOLUTION: To find an expression for the *nth* MacGonagall number, as a function of n, in closed form, we use the following claim.

    Claim: Suppose a, b, x are functions such that $a, b, x : \mathbf{N} \rightarrow \mathbf{R}$, where $\mathbf{N}$ and $\mathbf{R}$ are the set of natural numbers and real numbers, respectively. Define $a(n) = n^2$, $b(n) = (-1)^n$, and $x(n) = \mathcal{M}_n$. Then $a(n) + b(n) = x(n)$

    Remark: To simplify notation, we write $x_n = x(n)$, $a_n = a(n)$, and $b_n = b(n)$

    Proof. This claim will be proved using a strong form of mathematical induction. The way the function x is defined requires one to use natural numbers $n = 0$, $n = 1$, $n = 2$, and $n = 3$ as base cases. These cases are considered in the basis.

Basis: Suppose that $n = 0$ to begin with. Then by the definition,

$$a_0 + b_0 = 0^2 + (-1)^0 = 1 = M_0 = x_0$$

This establishes that $a_0 + b_0 = x_0$

Suppose, next, that $n = 1$. Again, by the definition,

$$a_1 + b_1 = 1^2 + (-1)^1 = 1 - 1 = M_1 = 0$$

Therefore, $a_1 + b_1 = x_1$.

Suppose, now, that $n = 2$. Yet again, by the definition,

$$a_2 + b_2 = 2^2 + (-1)^2 = 4 + 1 = 5 = M_2 = 5$$

which establishes that $a_2 + b_2 = x_2$ is established

Lastly, suppose that n = 3. By the definition,

$$a_3 + b_3 = 3^2 + (-1)^3 = 9 - 1 = 8 = \mathcal{M}_3 = x_3$$

Thus, it is established that $a_3 + b_3 = x_3$.

Inductive Step: Let k be an arbitrary integer such that $k \geq 3$. It is sufficient to use the inductive hypothesis to prove the following inductive claim.

Inductive Hypothesis: Suppose that s is an integer such that $0 \leq s \leq k$, so that $a_s + b_s = x_s$.

Inductive Claim: If $s = k + 1$ then $a_{k+1} + b_{k+1} = x_{k+1}$.

We want to show that the inductive claim holds. By the inductive step, it follows that $s = k + 1 \geq 5$. By this inequality and the definition of x,

$$x_{k+1} = \mathcal{M}_{k+1} = 2\mathcal{M}_k - 2\mathcal{M}_{k-2} + \mathcal{M}_{k-3}$$

Notice that

$$k + 1 \geq 4 \implies s - 1 = k \geq 3 \implies 3 \leq s - 1 \leq k \qquad (14.1)$$

$$k + 1 \geq 4 \implies s - 3 = k - 2 \geq 1 \implies 1 \leq s - 3 \leq k \qquad (14.2)$$

$$k + 1 \geq 4 \implies s - 4 = k - 3 \geq 0 \implies 0 \leq s - 4 \leq k \qquad (14.3)$$

By (14.1), (14.2), (14.3), and the inductive hypothesis, it follows that

$$\mathcal{M}_{s-1} = (s-1)^2 + (-1)^{s-1} \qquad (14.4)$$

$$\mathcal{M}_{s-3} = (s-3)^2 + (-1)^{s-3} \qquad (14.5)$$

$$M_{s-4} = (s-4)^2 + (-1)^{s-4} \qquad (14.6)$$

It now follows by (14.4), (14.5), and (14.6) that

$$2\mathcal{M}_k - 2\mathcal{M}_{k-2} + \mathcal{M}_{k-3} = 2\mathcal{M}_{s-1} - 2\mathcal{M}_{s-3} + \mathcal{M}_{s-4} =$$

$$2((s-1)^2 + (-1)^{s-1}) - 2((s-3)^2 + (-1)^{s-3}) + (s-4)^2 + (-1)^{s-4} =$$

$$2[s^2 - 2s + 1 + (-1)^{s-1}] - 2[s^2 - 6s + 9 + (-1)^{s-3}] + s^2 - 8s + 16 + (-1)^{s-4} =$$

$$2s^2 - 4s + 2 + 2(-1)^{s-1} - 2s^2 + 12s - 18 - 2(-1)^{s-3} + s^2 - 8s + 16 + (-1)^{s-4} =$$

$$s^2 + 2(-1)^{s-1} - 2(-1)^{s-3} + (-1)^{s-4} = s^2 + (-1)^{s-4}$$

Substituting $s = k+1$ back into the equation gives

$$(k+1)^2 + (-1)^{k-3} = (k+1)^2 + (-1)^{k-3}(-1)(-1)(-1)(-1) = (k+1)^2 + (-1)^{k+1}$$

Therefore,

$$x_{k+1} = \mathcal{M}_{k+1} = (k+1)^2 + (-1)^{k+1} = a_{k+1} + b_{k+1}$$

which establishes the inductive claim.

It follows by the basis, inductive hypothesis, and inductive claim that $a_n + b_n = x_n$ for $n \geq 0$. □

Therefore, it is established by the claim that the closed form for the *nth* MacGonagall number is $n^2 + (-1)^n$.

References

[1] Wayne Eberley (2019) *Computer Science 331 Notes Introduction to Analysis of Algorithms I: Correctness of Simple Recursive Algorithms.*

[2] Wayne Eberley (2019) *Computer Science 331 Notes Introduction to Analysis of Algorithms II: Correctness of Simple Algorithms with a while Loop.*

[3] Wayne Eberley (2019) *Computer Science 331 Notes Lecture 5: Analyzing the Running Times of Algorithms.*