Assignment 1 CPSC 331

Juwei Wang UCID 30053278

1.To prove f(n) = n is a bound function

（Citation: Solution03 Page 6 On D2l, Tut slides 03）

Firstly, we should prove the input is integer-valued. Because n is an integer-valued input which satisfies all the definition of this recursive algorithm's input (By the precondition, n is of type integer.

Secondly, when executing the line 10 and line 11, the algorithm calls itself with the input n replaced by the input n-1, n-2, n-3 and n-4.so we can conclude that n reduces by at least 1 with every recursive call replaced by n –1, n-2, n-3 and n-4.

Thirdly, because the precondition is satisfied, we can know that if f(n) = n<= 0
As well then n = 0. Finally, the algorithm will end between the line 1-8 without the algorithm calling itself recursively.

To conclude, because all the properties above are satisfied, the f(n) = n is a bound function for the recursive algorithm.

2.By looking at the code, it is rationally to be considered that the input is fixed and there is no global data accessible or revised.so we can prove the followings:

（Citation: Solution03 Page 6-7 On D2l, Tut slides 03）

Claim: For every non-negative integer n, if the sGrin algorithm is executed with n as input then this this execution of the algorithm eventually ends, and the Grindelwald number, Gn, is returned as output.

Theorem: Suppose n is a non-negative integer and that the algorithm sGrin is executed given n as input. Then the execution ends and returns the nth Grindelwald number Gn as output.

Proof Basis: This will be proved using strong induction on n. there are 4 cases n = 0, n = 1, n = 2, n = 3 that will be considered as basis.

Basis: (n = 0). When n = 0, then the test at line 1 will pass and execution will terminate at line 2, returning the value of 1, which is the G0th number. Thus the 0th Grindelwald number 1 is returned when n = 0.

(n = 1). When n = 1, then the test at line one will fail, but the test at line 3 will pass and execution will terminate at line 4, returning the value of 2, which is the G1st Grindelwald number. Thus the 1st Grindelwald number 2 is returned when n = 1.

(n = 2). When n = 2, then the test at line 1 and at line 3 will fail, but the test at line 5 will pass and execution will terminate at line 6, returning the value of 3, which is the G2nd Grindelwald number. Thus the 2nd Grindelwald number 3 is returned when n = 2.

(n = 3). When n = 3, then the test at line 1, line 3 and line 5 will fail, but the test at line 7 will pass and execution will terminate at line 8, returning the value of 4, which is the G3rdGrindelwald number. Thus the 3rd Grindelwald number 4 is returned when n = 3.

Inductive Step: Suppose k is an integer such that k >= 3.

Inductive Hypothesis: Suppose n is a non-negative integer such that 0 <= n <= k. Then if the algorithm sGrin executes using n as input, then it eventually ends and the Grindelwald number will be returned as output.

Inductive Claim: If the algorithm sGrin is executed with n = k+1 as input, then the algorithm eventually ends and the G_{k+1} Grindelwald number is returned as output.

Proof: Suppose that the algorithm sGrin is executed with n = k + 1 as input. Since k >= 3, then k+1 >= 4 so n >= 4.

Since n >= 4, then the tests at line 1, line 3, line 5 and line 7 will all fail, so line 9 will be executed. Thus, the function will be called recursively with (n-1), (n-2), (n-3), (n-4) as arguments.

1. For the recursive call n-1, since n = k+1 >= 3, then n – 1 = k +1 -1 = k >= 3. Thus, by the inductive hypothesis it follows that the algorithm eventually terminates and $G_{n-1} = G_k$ is returned as output.

2. For the recursive call n-2, since n = k+1 >= 3, then n – 2 = k +1 -2 = k-1 >= 2. Thus by the inductive hypothesis it follows that the algorithm eventually terminates and $G_{n-2} = G_{k-1}$ is returned as output.

3. For the recursive call n-3, since n = k+1 >= 3, then n – 3 = k +1 -3 = k-2 >= 1. Thus, by the inductive hypothesis it follows that the algorithm eventually terminates and $G_{n-3} = G_{k-2}$ is returned as output.

4. For the recursive call n-4, since n = k+1 >= 3, then n – 4 = k +1 -4 = k-3 >= 0. Thus by the inductive hypothesis it follows that the algorithm eventually terminates and $G_{n-4} = G_{k-3}$ is returned as output.

Thus, one can see that by inspection of line 9-11 that the algorithm eventually ends. Furthermore, since k+1 >= 3, then by the definition of Grindelwald number.
(when k +1 is even) $Gk_{+1} = 2 * G_k – 2*G_{k-2} + G_{k-3}$
(when k +1 is odd) $G_{k+1} = G_k – 3*G_{k-1} - 5 * G_{k-2}+ 2*G_{k-3}$

Thus, the inductive claim is established.

Therefore, by the property of strong induction, if the algorithm sGrin is executed given a non-negative integer n as input, then the execution eventually ends and the nth Grindelwald number $G_n$is returned as output.

Claim: For every non-negative integer n, if the sGrin algorithm is executed with n as input then this this execution of the algorithm eventually ends, and the
nth Grindelwald number, Gn, is returned as output.

Furthermore, we can see from inspection of the code that the algorithm does not have any “undocumented side effects”. The algorithm does not access any global data, nor does it change the value of the input variable n. As proven in question number 2, we proved that the output provided is the nth Grindelwald number as output.

Thus since the algorithm ends, and the correct output is produced with no side effects, then we can conclude that the algorithm sGrin correctly solves the Grindelwald number problem.

3. See the Java file.

4. （Citation: Solution06 Page 1-3 On D2l, Tut slides 05）

In order to solve the runtime for this recursive algorithm we will use the uniform cost criterion to give a recurrence for $t_{sGrin}(n)$. we will do this by looking at the values for the input n >= 0 as stated in the precondition of the problem.

When (n = 0): Then test at line 1 passes, and line 2 is executed, thus ending the execution. So then $t_{sGrin}(0) = 2$.

When (n = 1): Then the test at line 1 fails, but the test at line 3 passes so line 4 is executed, thus ending the execution. So then $t_{sGrin}(1) = 3$.

When (n = 2): Then the tests at lines 1 and line 3 fails, but the test at line 5 passes so line 6 is executed, thus ending the execution. So Then $t_{sGrin}(2) = 4$.

When (n = 3): Then the tests at lines 1, line 3 and line 5 fail, but the test at line 7 passes so line 8 is executed, thus ending the execution. So then $t_{sGrin}(3) = 5$.

When (n >= 4 and n is even), then the tests at line 1, 3, 5 and 7 fail, so line 9 is executed with the recursive calls. Thus the $t_{sGrin}(n)$ when $n > 3$ is $T_{sGrin}(n-1) + T_{sGrin}(n-3) + T_{sGrin}(n-4) + 6$.

When (n >= 4 and n is odd), then the tests at line 1, 3, 5 ,7 and 9 fail, so line 11 is executed with the recursive calls. Thus the $t_{sGrin}(n)$ when n > 3 is $T_{sGrin}(n-1) + T_{sGrin}(n-2) + T_{sGrin}(n-3) + T_{sGrin}(n-4) + 6$.

$$T(n) \begin{cases} 2 & (n = 0) \\ 3 & (n = 1) \\ 4 & (n = 2) \\ 5 & (n = 3) \\ T_{sGrin}(n-1) + T_{sGrin}(n-3) + T_{sGrin}(n-4) + 6 & (n>=4 \text{ and } n \text{ is even}) \\ T_{sGrin}(n-1) + T_{sGrin}(n-2) + T_{sGrin}(n-3) + T_{sGrin}(n-4) + 6. & (n>=4 \text{ and } n \text{ is odd}) \end{cases}$$

5. Proof that $t_{sGrin}(n) >= (3/2)^n$ for all n >= 0. We will prove this by strong mathematical induction. （Citation: Solution06 Page 1-3 On D2l, Tut slides 05）

Basis: (n = 0), then $t_{sGrin}(n) = 2 >= 1 = (3/2)^0$

(n = 1), then $t_{sGrin}(n) = 3 >= 3/2 = (3/2)^1$

(n = 2), then $t_{sGrin}(n) = 4 >= 9/4 = (3/2)^2$

(n = 3), then $t_{sGrin}(n) = 5 >= 27/8 = (3/2)^3$

Inductive step: Suppose K is an integer >= 3.

Inductive Hypothesis: Suppose that i is an integer such that 0 <= i <= k such that $t_{sGrin}(i) >= (3/2)^i$.

Inductive Claim: The inductive claim is $t_{sGrin}(k+1) >= (3/2)^{k+1}$.

Proof: $t_{sGrin}(k+1) = T_{sGrin}(k) + T_{sGrin}(k-2) + T_{sGrin}(k-3) + 6$ and

$t_{sGrin}(k+1) = T_{sGrin}(k) + T_{sGrin}(k-1) + T_{sGrin}(k-2) + T_{sGrin}(k-3) + 6$. By the recurrence

Then by the inductive hypothesis:

(When n>=4 and n is Even)

$T_{sGrin}(k) + T_{sGrin}(k-2) + T_{sGrin}(k-3) + 6 >= (3/2)^{k} + (3/2)^{k-2} + (3/2)^{k-3} + 6 =$
$(3/2)^{k+1}((3/2)^{-1} + (3/2)^{-3} + (3/2)^{-4}) + 6 = (3/2)^{k+1}(94/81) + 6 >= (3/2)^{k+1}$

(When n>=4 and n is Odd)

$t_{sGrin}(k+1) = T_{sGrin}(k) + T_{sGrin}(k-1) + T_{sGrin}(k-2) + T_{sGrin}(k-3) + 6 >= (3/2)^{k} + (3/2)^{k-1} + (3/2)^{k-2} + (3/2)^{k-3} + 6 = (3/2)^{k+1}((3/2)^{-1} + (3/2)^{-2} + (3/2)^{-3} + (3/2)^{-4}) + 6 = (3/2)^{k+1}(130/81) + 6 >= (3/2)^{k+1}$

Thus this proves that for all integers n >= 0, $T_{sGrin}(n) >= (3/2)^{n}$

6.The loop invariant for the Grindelwald algorithm provided is Loop invariant:

（Citation: Solution04 Page 1-2 On D2l, Tut slides 04）

I. n is a non-negative integer >= 4

II. i is an integer such that 4 <= i < = n + 1

III. G is an integer type array with the sizes of n+1

IV. G[s] = Gs for every integer j so that 0 <= j <= i

Proof of loop invariant:

a. Show that the test function of the loop has no side effects:

We can see from inspection of the code that the algorithm does not have any "undocumented side effects". The algorithm does not access any global data, nor does it change the value of the input variable n.

b. Show that invariant is satisfied by the time the loop is reached and during execution of the algorithm.

For the while loop to be reached an input integer n must be at least 4. Suppose an integer n passed to the function is >= 4 then the tests on line 1,3,5,7 will fail triggering the else clause starting between line 9 and 14.

Thus n is a non-negative integer such that n >= 4.

On line 9 G is assigned as an array of G[] with size n+1

On line 14 i is assigned 4.

On line 10 G[0] is assigned 1 which is $G_0 = G_{i-4}$

On line 11 G[1] is assigned 2 which is $G_1 = G_{i-3}$

On line 12 G[2] is assigned 3 which is $G_2 = G_{i-2}$

On line 13 G[3] is assigned 4 which is $G_3 = G_{i-1}$

7.From the proof, we know that the loop invariant is satisfied at the beginning of the loops execution, thus we will prove that it is also satisfied at the end.

（Citation: Solution04 Page 1-2 On D2l, Tut slides 04）

From the paragraph above it follows that the loop invariant is valid at the beginning of the loop execution given that the precondition for the algorithm is satisfied.

c. Show That if the loop invariant is satisfied at the start of the loop, then it is also satisfied at the end:

Since n >= 4 as stated i, and the value of n never changes, then n >= 4 at the end of execution of the loop.

On the final execution of the loop, i = I + 1 because if i > n, then the loop would terminate.

At the end of the final execution of the loop on line 19, variable i is incremented by 1, so that i > n, thus 4 <= i <= n+1 at the end of the loop

On the line 17 is G[i] assigned a value of 2×G[i−1]−2×G[i−3]+G[i−4]. It follows that this equation can be rewritten as $2^* G_{i-1} - 2 * G_{i-3} + G_{i-4}$ thus assigning it a value of H_{i+1} .
Since i = I + 1 on the final iteration, then I > n, thus

$G_i = G_n$. After line 19 i is incremented by 1 so i = n, thus $G_i = G_n$. (when b is even)

On the line 18 is G[i] assigned a value of G[i−1]+3×G[i−2]−5×G[i−3]+2×G[i−4] . It follows that this equation can be rewritten as G_{i-1} + 3 * G_{i-2} - 5 * G_{i-3} + 2 * G_{i-4} thus assigning it a value of H_{i+1} . Since i = I + 1 on the final iteration, then I > n, thus

G_i = G_n. After line 19 i is incremented by 1 so i = n, thus G_i = G_n. (when n is odd)

8. Proof of partial correctness of algorithm

（Citation: Solution04 Page 3 On D2l, Tut slides 04）

To prove partial correctness, we will prove that

A. the algorithm eventually ends with the number G_n returned as output and causes no undocumented side effects, OR

B.That the execution of the algorithm never ends.

From the inspection of code it follows that the algorithm has no side effects since it does not manipulate input variables or access global data.

If n = 0, then execution ends at line 2 which returns G_0 = 1.

If n = 1, then execution ends at line 4 which returns G_1 = 2.

If n = 2, then execution ends at line 6 which returns G_2 = 3.

If n = 3, then execution ends at line 8 which returns G_3 = 4

If n >= 4, then the loop will execute on line 15 will execute. Thus this loop either terminates, or it does not:

If the loop terminates, then the execution will end with the array G[] returned on line 20. By the loop invariant proved in question 7, variable G_i = G_n on the final iteration of the loop.

Thus if the loop ends, then the G_n Grindelwald number will be returned as output satisfying the problems post-condition.

If the loop does not terminate, then it is clear that the algorithm does not terminate either thus proving.

Thus both conditions for partial correctness have been proven, so we can conclude that the algorithm is partially correct.

9. Bound function for the algorithm provided : f(n,i)= n-I +1

（Citation: Solution04 Page 3 On D2l, Tut slides 06）

Proof:

a. Prove that the bound function decreases by at least 1 for each iteration of the loop:

Before the execution of while loop begins on line 14, i is declared to be an integer with a value of 4. In order for the loop to be reached and the test on line 16 to pass, n must be an integer n >= 4.
Since the value of n is never changed, and the value of i is increases after each iteration, then the bound function i increases by 1 each loop iteration.

b. Prove that when the value of the bound function is <= 0, the loop test fails.
The loop test fails when i > n.

As stated in the proof of the loop invariant, on line 19 of the final iteration of the loop, i is incremented such that i > n. If i > n, the value of the bound function n < I. If n < i, then the loop test on line 15 fails. Thus the loop test fails when the bound function n = i. (since n – I + 1 <= 0, so we can deduce n + 1 <= i)

Thus we have proved that the function f(n,i) = n - i + 1is a bound function for the algorithm.

10.Prove that if this algorithm is executed when the precondition for the “Grindelwald Gaggle Computation” problem is satisfied, and the while loop is reached and executed, then the execution of the loop eventually ends. （Citation: Solution04 Page 4 On D2l, Tut slides 06）

Claim: For every non-negative integer n, if the sGrin algorithm is executed with n as input then the while loop is reached and executed and the execution will end.

Proof Basis: This will be proved using strong induction on n. there are 4 cases n = 0, n = 1, n = 2, n = 3 that will be considered as basis.

Basis: (n = 0). When n = 0, then the test at line 1 will pass and execution will terminate at line 2.

(n = 1). When n = 1, then the test at line one will fail, but the test at line 3 will pass and execution will terminate at line 4.

(n = 2). When n = 2, then the test at line 1 and at line 3 will fail, but the test at line 5 will pass and execution will terminate at line 6.

(n = 3). When n = 3, then the test at line 1, line 3 and line 5 will fail, but the test at line 7 will pass and execution will terminate at line 8.

Inductive proof:
(n>=4) when n >= 4, the lines 1,3,5,7 all failed to test. Then the 9 – 14 lines will be executed.

1. the correct output is produced with no side effects at lines 15
2. if n is even, the lines 16,17,19 will be reached, because lines 17 and 19 are recursive assigned statements and 16 is a test statement. The loop body will finally end whenever it is executed.
3. if n is odd, the lines 16,18,19 will be reached, because lines 18 and 19 are recursive assigned statements and 16 is a test statement. The loop body will finally end whenever it is executed.

To conclude, if the algorithm runs when all the preconditions for the Grindelwald Gaggle computation problem is satisfied and the while loop is reached and the loop eventually ends.

11. To prove full correctness, it suffices to show that the algorithm is partially correct, that a bound function exists and that there are no undocumented side effects

（Citation: Solution04 Page 5 On D2l, Tut slides 06）

By inspection of the code it is clear that there are no undocumented side effects.

In question 9, we proved that the algorithm is partially correct.

In question 10, we proved that a bound function exists.

Thus we can then conclude that the algorithm is fully correct.

12. give an upper bound for TfGrin(n)（Citation: Solution06 Page 4 On D2l, Tut slides 06）

By inspection of code it follows that the algorithm and test condition have no side effects and test function halts.

From question 9,10 it follows that the while loop will terminate and the bound function for the loop exists.

By applying 2nd loop theorem it follows that n-i is the upper bound of times the while loop will be executed.

When (n = 0) : Test at line 1 passes, and line 2 is executed, thus ending the execution. So then tSGrin (0) = 2.

When (n = 1) : Test at line 1 fails, but the test on line 3 passes, and line 4 is executed, thus ending the execution. So then tSGrin (1) = 3.

When (n = 2) : Tests at line 1 and line 3 fail, but the test on line 5 passes, and line 6 is executed, thus ending the execution. So then SGrin (2) = 4.

When (n = 3) : Tests at line 1, line 3 and line 5 fail,, but the test on line 7 passes, and line 8 is executed, thus ending the execution. So then tSGrin (3) = 5.

When (n >= 4) : Tests at line 1, line 3, line 5 and line 7 fails, so lines 9 to 14 are executed before entering the while loop. (10 steps executed so far).

From the bound function established in question 9, the loop executes (n-3) times, and the loop test is executed (n - 2) times, The method return statement is then executed once. Since the body of the loop has 3 instructions (same steps whatever n is even or odd) , TSGrin (n) can be written as:

TSGrin (n) = 3(n-3) + (n-2) + 11 = 3n - 9 + n - 2 + 11 = 4n when n >= 4.

Thus the upper bound for TSGrin (n) can be written as:

$$T(n) \begin{cases} 2 \ (n = 0) \\ 3 \ (n = 1) \\ 4 \ (n = 2) \\ 5 \ (n = 3) \\ 4n \ (n >= 4 \text{ and } n \text{ is Even}) \\ 4n \ (n >=4 \text{ and } n \text{ is odd}) \end{cases}$$

13. See the Java file.

14. close form (Citation: Solution06 Page 4 On D2l)

The relationship between G_n and n is as follows:

G_{sGrin} (n) = n+1 for any integer n >= 0

We will prove this by Strong induction:

Claim: G_{sGrin} (n) = 10-n for any integer n >= 0

Proof:

Basis (n = 0): n+1 =1 = G_{sGrin} (n)

Basis (n = 1): n+1 = 2 = G_{sGrin} (n)

Basis (n = 2): n+1 = 3 = G_{sGrin} (n)

Basis (n = 3): n+1= 4 = G_{sGrin} (n)

Inductive Step: Suppose k is an integer such that k >= 4.

Inductive Hypothesis: Suppose that x is a non-negative integer such that 0 <= x <= k, then G_{sGrin} (x) = n + 1.

Inductive Claim: G_{sGrin} (k+1) = 1 + (k+1)

Proof:

From the definition of G_{sGrin} (k+1) = $2G_k - 2G_{k-2} + G_{k-3}$ (k+1 >= 4 and k+1 is even)

From the IH: G_{sGrin} (k+1) =2 (k + 1) -2(1 + (k - 2)) + (1 + (k-3))

=2k + 2 – 2k + 2 + k - 2

=k + 2

=(k+1) + 1

From the definition of G_{sGrin} (k+1) = $G_k + 3G_{k-1} - 5G_{k-2} + 2G_{k-3}$ (k+1 >= 4 and k+1 is odd)

From the IH: G_{sGrin} (k+1) =(k + 1) + 3(1+ (k -1)) -5(1 + (k - 2)) + 2(1 + (k-3))

= k + 1 + 3k – 5k + 5 + 2k - 4

= k + 2

=(k+1) + 1

Conclusion: Thus we can conclude by the property of induction that G_{sGrin} (n) = n+1 for any integer n >= 0.