Assignment 1

Haohu Shen UCID: 30063099

CPSC 331 - Data Structures, Algorithms, and Their Analysis

May 21, 2019

Question 1

CITATION

Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 42 to page 49 in L02_correctness.pdf[1] in order to providing a completed, clear and professional proof.

Proof In order to prove $f(n) = n$ is a bound function for the recursive algorithm, we show that it satisfies all 3 properties of the definition of a bound function for a recursive algorithm:

- $f(n)$ is an integer-valued function because n is an integer as input.
- We can see from the line 10 and line 11 of the algorithm described in Figure 1, when the algorithm is recursively call itself, the parameter n is reduced by at least 1, thus the value of $f(n) = n$ is reduced by at least 1.
- Since the input $n \geq 0$ as an integer to be the precondition of **Grindelwald Gaggle Computation**, when the bound function $f(n) = n \leq 0$, $n = 0$ for the precondition to be satisfied. When $n = 0$, the test at step 1 passed and execution continues with step 2 and then the execution ends, thus the algorithm is not recursively called itself.

Since $f(n) = n$ satisfies all the properties included in the definition of a bound function for a recursive algorithm, it is a bound function of this recursive algorithm *sGrin*.

Question 2

CITATION

Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 9 to page 13 in L02_correctness.pdf[1] in order to providing a completed, clear and professional proof.

Claim For every non-negative integer n, if the algorithm *sGrin* is executed with n as input, then the execution of the algorithm eventually ends, and the n^{th} Grindelwald Gaggle number, $\mathcal{G}_n$, is returned as output.

Proof This will be proved by the strong form of mathematical induction on n. Cases that $n = 0$, $n = 1$, $n = 2$,

$n = 3$ will be considered in the basis.

Basis (n=0) When $n = 0$, the test at step 1 passed, so the execution continued with step 2, which caused the execution of the algorithm eventually ends. Therefore, the value $\mathcal{G}_0 = 1$ is returned as output, as required.

Basis (n=1) When $n = 1$, the test at step 1 failed, thus execution continued with step 3. The test succeeded, thus the execution continued at step 4, which caused the execution of the algorithm eventually ends. Therefore, the value $\mathcal{G}_1 = 2$ is returned as output, as required.

Basis (n=2) When $n = 2$, the tests at step 1 and step 3 all failed, thus execution continued with step 5. The test succeeded, thus the execution continued at step 6, which caused the execution of the algorithm eventually ends. Therefore, the value $\mathcal{G}_2 = 3$ is returned as output, as required.

Basis (n=3) When $n = 3$, the tests at step 1, step 3 and step 5 all failed, thus execution continued with step 7. The test succeeded, thus the execution continued with step 8, which caused the execution of the algorithm eventually ends. Therefore, the value $\mathcal{G}_3 = 4$ is returned as output, as required.

Inductive Step: Let $k \geq 3$ be an integer. It is necessary and sufficient to use

Inductive Hypothesis: Suppose n is a non-negative integer. Suppose for all integers n such that $0 \leq n \leq k$, if the algorithm *sGrin* is executed given n as input, then the execution of the algorithm ends eventually and the Grindelwald Gaggle number $\mathcal{G}_n$ is returned as output.

to prove

Inductive Claim: If the algorithm *sGrin* is executed given $n = k + 1$ as input, then the execution of the algorithm ends eventually and the Grindelwald Gaggle number $\mathcal{G}_{k+1}$ is returned as output.

Suppose that the algorithm *sGrin* is executed given $n = k + 1$ as input. Since $k \geq 3$, n is integer such that $n = k + 1 \geq 4$. According to the parity of n, we can classify the value of n into 2 cases.

Case 1 (n is even): Since $n \geq 4$, the tests at step 1, step 3 and step 5 all failed, thus the execution of the algorithm continued with the execution of the test at step 9. Since n is an even integer, the test succeeded and the execution continued with step 10, which includes a recursive execution of the algorithm with $n - 1$, $n - 3$ and $n - 4$ as input.

- Since $n = k + 1 \geq 4, 0 \leq n - 1 = (k + 1) - 1 = k \leq k$. Thus, for the recursive call $sGrin(n - 1)$, by the ***inductive hypothesis***, the recursive execution of the algorithm eventually ends and

  $$\mathcal{G}_{n-1} = \mathcal{G}_k$$

  is returned as output.

- Since $n = k + 1 \geq 4, 0 \leq n - 3 = (k + 1) - 3 = k - 2 \leq k$. Thus, for the recursive call $sGrin(n - 3)$, by the ***inductive hypothesis***, the recursive execution of the algorithm eventually ends and

$$\mathcal{G}_{n-3} = \mathcal{G}_{k-2}$$

is returned as output.

- Since $n = k+1 \geq 4, 0 \leq n-4 = (k+1)-4 = k-3 \leq k$. Thus, for the recursive call $sGrin(n-4)$, by the ***inductive hypothesis***, the recursive execution of the algorithm eventually ends and

  $$\mathcal{G}_{n-4} = \mathcal{G}_{k-3}$$

  is returned as output.

- The execution of the algorithm eventually ends after step 10. Since $k+1 \geq 4$, by the definition of recurrence we have

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

  when n is even and $n \geq 4$, as required.

Case 2 (n is odd): Since $n \geq 4$, the tests at step 1, step 3, step 5 and step 9 all failed, thus the execution of the algorithm continued with step 11, which includes a recursive execution of the algorithm with $n-1$, $n-2$, $n-3$ and $n-4$ as input.

- Since $n = k+1 \geq 4, 0 \leq n-1 = (k+1)-1 = k \leq k$. Thus, for the recursive call $sGrin(n-1)$, by the ***inductive hypothesis***, the recursive execution of the algorithm eventually ends and

  $$\mathcal{G}_{n-1} = \mathcal{G}_k$$

  is returned as output.

- Since $n = k+1 \geq 4, 0 \leq n-2 = (k+1)-2 = k-1 \leq k$. Thus, for the recursive call $sGrin(n-2)$, by the ***inductive hypothesis***, the recursive execution of the algorithm eventually ends and

  $$\mathcal{G}_{n-2} = \mathcal{G}_{k-1}$$

  is returned as output.

- Since $n = k+1 \geq 4, 0 \leq n-3 = (k+1)-3 = k-2 \leq k$. Thus, for the recursive call $sGrin(n-3)$, by the ***inductive hypothesis***, the recursive execution of the algorithm eventually ends and

  $$\mathcal{G}_{n-3} = \mathcal{G}_{k-2}$$

  is returned as output.

- Since $n = k+1 \geq 4, 0 \leq n-4 = (k+1)-4 = k-3 \leq k$. Thus, for the recursive call $sGrin(n-4)$

, by the ***inductive hypothesis***, the recursive execution of the algorithm eventually ends and

$$\mathcal{G}_{n-4} = \mathcal{G}_{k-3}$$

is returned as output.

- The execution of the algorithm eventually ends after step 10. Since $k + 1 \geq 4$, by the definition of recurrence we have

$$\mathcal{G}_n = \mathcal{G}_{n-1} + 3\mathcal{G}_{n-2} - 5\mathcal{G}_{n-3} + 2\mathcal{G}_{n-4} = \mathcal{G}_k + 3\mathcal{G}_{k-1} - 5\mathcal{G}_{k-2} + 2\mathcal{G}_{k-3} = \mathcal{G}_{k+1}$$

  when n is odd and $n \geq 4$, as required.

In both cases, the value returned as output follows by the definition of $\mathcal{G}_{k+1}$, thus the ***inductive claim*** is established.

Conclusion: Therefore, by strong form of mathematical induction, we can conclude that, for every non-negative integer n, if the algorithm *sGrin* is executed with n as input, then the execution of the algorithm eventually ends, and the n^{th} Grindelwald Gaggle number, $\mathcal{G}_n$, is returned as output.

Question 3

NOTE The file *SGrindelwald.java* has been uploaded to D2L dropbox.

Question 4

CITATION
Please note before grading, the format of the solution acts in a similar manner to the solution provided in page 26 in L05_efficiency.pdf[3] in order to providing a completed, clear and professional solution.

Solution In order to give a recurrence for $T_{sGrin}(n)$, we suppose the precondition is satisfied and the recursive algorithm is executed, thus n is an integer such that $n \geq 0$.

- If $n = 0$, during the execution of the algorithm *sGrin* on input n, the test at line 1 succeeded and the execution continued with the step at line 2, which caused the execution of the algorithm to terminate, thus the algorithm totally executes 2 steps, therefore,

$$T_{sGrin}(0) = 2$$

- If $n = 1$, during the execution of the algorithm *sGrin* on input n, the test at line 1 failed and the execution continued with the step at line 3. The test succeeded, thus the execution continued with the step at line 4, which caused the execution of the algorithm to terminate. Thus the algorithm totally executes 3 steps, therefore,

$$T_{sGrin}(1) = 3$$

- If $n = 2$, during the execution of the algorithm *sGrin* on input n, the tests at line 1 and line 3 all failed, thus execution continued with line 5. The test succeeded, thus the execution continued with the step at line 6, which caused the execution of the algorithm to terminate. Thus the algorithm totally executes 4 steps, therefore,

$$T_{sGrin}(2) = 4$$

- If $n = 3$, during the execution of the algorithm *sGrin* on input n, the tests at line 1, line 3 and line 5 all failed, thus execution continued with line 7. The test succeeded, thus the execution continued with the step at line 8, which caused the execution of the algorithm to terminate. Thus the algorithm totally executes 5 steps, therefore,

$$T_{sGrin}(3) = 5$$

- If $n \geq 4$ and n is even, during the execution of the algorithm *sGrin* on input n, the tests at line 1, line 3, line 5 and line 7 all failed, thus execution continued with line 9. The test succeeded, thus the execution continued with the step at line 10, which totally took 6 steps, and then called itself recursively, with input $n-1$, $n-3$, $n-4$. Therefore, when $n \geq 4$ and n is even,

$$T_{sGrin}(n) = T_{sGrin}(n-1) + T_{sGrin}(n-3) + T_{sGrin}(n-4) + 6$$

- If $n \geq 4$ and n is odd, during the execution of the algorithm *sGrin* on input n, the tests at line 1, line 3, line 5, line 7 and line 9 all failed, thus execution continued with line 11, which totally took 6 steps, and then called itself recursively, with input $n-1$, $n-2$, $n-3$, $n-4$. Therefore, when $n \geq 4$ and n is odd,

$$T_{sGrin}(n) = T_{sGrin}(n-1) + T_{sGrin}(n-2) + T_{sGrin}(n-3) + T_{sGrin}(n-4) + 6$$

To conclude, when executed with a non-negative integer n as input, this recursive algorithm carries out $T_{sGrin}(n)$ steps, where

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

Question 5

CITATION

Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 48 and page 49 in L01_intro_and_math_review.pdf[2] in order to providing a completed, clear and professional proof.

Proof We prove that $T_{sGrin}(n) \geq (\frac{3}{2})^n$ for every non-negative integer n by induction on n. The strong form of mathematical induction will be used, and the cases that $n = 0$, $n = 1$, $n = 2$ and $n = 3$ will each be considered in the basis.

Basis (n=0) When $n = 0$, $T_{sGrin}(n) = T_{sGrin}(0) = 2 \geq 1 = (\frac{3}{2})^0 = (\frac{3}{2})^n$, as required.

Basis (n=1) When $n = 1$, $T_{sGrin}(n) = T_{sGrin}(1) = 3 = \frac{6}{2} \geq \frac{3}{2} = (\frac{3}{2})^1 = (\frac{3}{2})^n$, as required.

Basis (n=2) When $n = 2$, $T_{sGrin}(n) = T_{sGrin}(2) = 4 = \frac{16}{4} \geq \frac{9}{4} = (\frac{3}{2})^2 = (\frac{3}{2})^n$, as required.

Basis (n=3) When $n = 3$, $T_{sGrin}(n) = T_{sGrin}(3) = 5 = \frac{40}{8} \geq \frac{27}{8} = (\frac{3}{2})^3 = (\frac{3}{2})^n$, as required.

Inductive Step: Let $k \geq 3$ be an integer. It is necessary and sufficient to use

Inductive Hypothesis: Suppose that m is a non-negative integer. Suppose for all integers m such that $0 \leq m \leq k$,

$$T_{sGrin}(m) \geq (\frac{3}{2})^m$$

to prove

Inductive Claim:

$$T_{sGrin}(k+1) \geq (\frac{3}{2})^{k+1}$$

Since $k \geq 3$, $k+1 \geq 4$. According to the parity of $k+1$, we can classify the value of $k+1$ into 2 cases.

Case 1 (k + 1 is even):

Since $k+1 \geq 4$ and $k+1$ is even, by the definition of recurrence,

$$T_{sGrin}(k+1) = T_{sGrin}((k+1)-1) + T_{sGrin}((k+1)-3) + T_{sGrin}((k+1)-4) + 6$$

That is,

$$T_{sGrin}(k+1) = T_{sGrin}(k) + T_{sGrin}(k-2) + T_{sGrin}(k-3) + 6$$

Since $k \geq 3$, we have $0 \leq k, k-2, k-3 \leq k$, thus by the **inductive hypothesis**,

$$\begin{aligned}
T_{sGrin}(k+1) \geq (\frac{3}{2})^k + (\frac{3}{2})^{k-2} + (\frac{3}{2})^{k-3} + 6 &= (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{2}{3})^2 + (\frac{3}{2})^k(\frac{2}{3})^3 + 6 \\
&= (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{4}{9} + \frac{8}{27}) + 6 \\
&= (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{20}{27}) + 6 \\
&\geq (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{20}{27}) \\
&\geq (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{1}{2}) \\
&= (\frac{3}{2})^k(1 + \frac{1}{2}) \\
&= (\frac{3}{2})^k(\frac{3}{2}) \\
&= (\frac{3}{2})^{k+1}
\end{aligned}$$

as required in this case.

Case 2 (k + 1 is odd):

Since $k+1 \geq 4$ and $k+1$ is odd, by the definition of recurrence,

$$T_{sGrin}(k+1) = T_{sGrin}((k+1)-1) + T_{sGrin}((k+1)-2) + T_{sGrin}((k+1)-3) + T_{sGrin}((k+1)-4) + 6$$

That is,

$$T_{sGrin}(k+1) = T_{sGrin}(k) + T_{sGrin}(k-1) + T_{sGrin}(k-2) + T_{sGrin}(k-3) + 6$$

Since $k \geq 3$, we have $0 \leq k, k-1, k-2, k-3 \leq k$, thus by the **inductive hypothesis**,

$$\begin{aligned}
T_{sGrin}(k+1) \geq (\frac{3}{2})^k + (\frac{3}{2})^{k-1} + (\frac{3}{2})^{k-2} + (\frac{3}{2})^{k-3} + 6 &= (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{2}{3}) + (\frac{3}{2})^k(\frac{2}{3})^2 + (\frac{3}{2})^k(\frac{2}{3})^3 + 6 \\
&= (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{2}{3} + \frac{4}{9} + \frac{8}{27}) + 6 \\
&= (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{38}{27}) + 6 \\
&\geq (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{38}{27}) \\
&\geq (\frac{3}{2})^k + (\frac{3}{2})^k(\frac{1}{2}) \\
&= (\frac{3}{2})^k(1 + \frac{1}{2}) \\
&= (\frac{3}{2})^k(\frac{3}{2}) \\
&= (\frac{3}{2})^{k+1}
\end{aligned}$$

as required in this case.

In both cases, $T_{sGrin}(k+1) \geq (\frac{3}{2})^{k+1}$ when $k+1 \geq 4$, thus the ***inductive claim*** is established.

Conclusion: Therefore, by strong form of mathematical induction, we can conclude that, for every non-negative integer n, $T_{sGrin}(n) \geq (\frac{3}{2})^n$, that is, the number of steps executed by the *sGrin* algorithm is at least ***exponential*** in its input.

Question 6

CITATION
Please note before grading, the format of the loop invariant is in a way similar to the format provided in page 7 in L03_iterative_correctness.pdf[4] in order to providing a clear and professional loop invariant.

Claim The following is a loop invariant for *while* loop at lines 15-19 of the algorithm provided in Figure 2:

Loop Invariant:

1. n is an input integer such that $n \geq 4$
2. G is a variable integer array and its length is $n+1$
3. i is an integer variable such that $4 \leq i \leq n+1$
4. $G[j] = \mathcal{G}_j$ for every integer j such that $0 \leq j \leq i-1$

Question 7

CITATION
Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 9, 10, 14, 15 in L03_iterative_correctness.pdf[4] in order to providing a completed, clear and professional proof.

Proof

- We prove that the loop test of *while* has no side-effects:
  Note that in line 15 of the algorithm, where the loop test of *while* located, we can see that the test does not change the value of any variables, inputs or global data. Thus the execution of the loop test has no side-effects.

- We prove that the assertion stated in **Question 6** holds when the loop is reached during an execution of the algorithm and the computational problem's precondition is satisfied. Since the computational problem's precondition is satisfied, n is an input integer such that $n \geq 0$:

  - When $n = 0$, the execution of the algorithm ends after the execution of steps 1 and 2, so that the *while* loop is never reached.

  - When $n = 1$, the execution of the algorithm ends after the execution of steps 1, 3 and 4, so that the *while* loop is never reached.

    - When $n = 2$, the execution of the algorithm ends after the execution of steps 1, 3, 5 and 6, so that the *while* loop is never reached.
    - When $n = 3$, the execution of the algorithm ends after the execution of steps 1, 3, 5, 7 and 8, so that the *while* loop is never reached.
    - When $n \geq 4$, tests in steps 1, 3, 5, 7 all fail, so that the execution of the algorithm continues with line 9.
        - Since $n \geq 4$, n is an input integer such that $n \geq 4$, thus the first statement in the assertion stated in **Question 6** holds.
        - At step 9, a bounded array whose length is $n + 1$ with integer as its base type is created and assigned to G, since the type of the array and length of array are not changed from step 10 to step 14, G is a variable integer array and its length is $n + 1$, thus the second statement in the assertion stated in **Question 6** holds.
        - At step 14, an integer variable i is assigned 4, since $n \geq 4$, $n + 1 \geq 5 \geq 4$, so i is an integer variable such that $4 \leq i \leq n + 1$, thus the third statement in the assertion stated in **Question 6** holds
        - After the execution of steps 10, 11, 12 and 13, $G[0] = 1$, $G[1] = 2$, $G[2] = 3$, $G[3] = 4$, G is not changed at step 14, thus $i = 4$, that is $i - 1 = 3$. Therefore, $G[j] = \mathcal{G}_j$ for every integer j such that $0 \leq j \leq i - 1$, thus the fourth statement in the assertion stated in **Question 6** holds.
- We prove that if that the assertion stated in **Question 6** holds at the beginning of any execution of the loop body, then it also satisfied when the execution of the loop body ends:
    - At the beginning of the execution of the body of the loop, $n \geq 4$, thus n is an input integer such that $n \geq 4$. We can see that no statements in the loop body of *while* change the value of n, thus n is still an integer such that $n \geq 4$ at the end of the execution of the body of the loop, thus the first statement in the assertion stated in **Question 6** holds.
    - At the beginning of the execution of the body of the loop, G is an integer array whose length is $n + 1$. We can see that no statements in the loop body of *while* change the type or the length of G, thus G is still a variable integer array and its length is $n + 1$ at the end of the execution of the body of the loop, thus the second statement in the assertion stated in **Question 6** holds.
    - Since i is assigned 4 and is not changed at step 14 and 15, i is an integer such that $i \geq 4$ at the beginning of the execution of the body of *while* loop. Since the loop test at step 15 must have been checked and passed because it is immediately before this, $i \leq n$. Thus i, n are integers such that $4 \leq i \leq n$ at the beginning of the execution of the body of *while* loop. This holds after the execution of step 16, 17, 18 since i or n is unchanged in these steps.
      Since i is incremented by 1 at step 19 and n is not changed, we can see at the end of step 19, that is, at the end of the execution of the loop body, $5 \leq i \leq n + 1$. Therefore, i is still an integer variable such that

$4 \leq i \leq n+1$. Thus the third statement in the assertion stated in **Question 6** holds.

  - Since $G[0]=1$, $G[1]=2$, $G[2]=3$, $G[3]=4$ and $4 \leq i \leq n$ at the beginning of the execution of the body of *while* loop, we can see that $G[j]=\mathcal{G}_j$ for every integer j such that $0 \leq j \leq i-1$ at the beginning of the execution of the loop body. According to the parity of i, we can classify the value of i into 2 cases.

    - *Case 1 (i is even):* If i is even, the test at step 16 will pass and thus $G[i]$ is set such that

$$\begin{aligned} G[i] &= 2G[i-1] - 2G[i-3] + G[i-4] \\ &= 2\mathcal{G}_{i-1} - 2\mathcal{G}_{i-3} + \mathcal{G}_{i-4} \\ &= \mathcal{G}_i \end{aligned}$$

      Thus, $G[j]=\mathcal{G}_j$ for every integer j such that $0 \leq j \leq i$ after the step 17 is executed. Since the value of i is increased by 1 after step 19 is executed, so that $G[j]=\mathcal{G}_j$ for every integer j such that $0 \leq j \leq i-1$ since i is updated.

    - *Case 2 (i is odd):* If i is odd, the test at step 16 fails and the execution continues with step 18, thus $G[i]$ is set such that

$$\begin{aligned} G[i] &= G[i-1] + 3G[i-2] - 5G[i-3] + 2G[i-4] \\ &= \mathcal{G}_{i-1} + 3\mathcal{G}_{i-2} - 5\mathcal{G}_{i-3} + 2\mathcal{G}_{i-4} \\ &= \mathcal{G}_i \end{aligned}$$

      Thus, $G[j]=\mathcal{G}_j$ for every integer j such that $0 \leq j \leq i$ after the step 18 is executed. Since the value of i is increased by 1 after step 19 is executed, so that $G[j]=\mathcal{G}_j$ for every integer j such that $0 \leq j \leq i-1$ since i is updated.

    Since in both cases, $G[j]=\mathcal{G}_j$ for every integer j such that $0 \leq j \leq i-1$, we can conclude that $G[j]=\mathcal{G}_j$ for every integer j such that $0 \leq j \leq i-1$ at the end of the execution of the loop body. Thus the fourth statement in the assertion stated in **Question 6** holds.

  - Therefore, all four statements of the assertion stated in **Question 6** holds at the beginning of any execution of the loop body, and all four statements of the assertion also satisfied when the execution of the loop body ends.

- Thus the assertion stated in **Question 6** is a loop invariant for the *while* loop at line 15-19 of the algorithm provided in Figure 2 by ***Loop Theorem #1*** since all preconditions of the theorem are satisfied.

Question 8

CITATION

Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 29 to page 33 in L03_iterative_correctness.pdf[4] in order to providing a completed, clear and professional proof.

Claim The algorithm shown in Figure 2 for the **Grindelwald Gaggle Computation** problem is partially correct.

Proof Suppose the algorithm is executed with the precondition being satisfied, thus a non-negative integer n will be given as input. We show that

(1) The execution of the algorithm ends eventually with the n^{th} Grindelwald number, $\mathcal{G}_n$, is returned as output and no undocumented inputs or global data accessed or modified.

$$or$$

(2) The execution of the algorithm never ends.

- When $n = 0$, the execution of algorithm eventually ends after the execution of steps 1 and 2, and

  $$\mathcal{G}_n = \mathcal{G}_0 = 1$$

  is returned as output, thus the problem's postcondition is satisfied. Also, we can see that at steps 1 and 2 no undocumented inputs or global data are accessed or modified by the algorithm. Thus the part **(1)** is satisfied.

- When $n = 1$, the execution of algorithm eventually ends after the execution of steps 1, 3 and 4, and

  $$\mathcal{G}_n = \mathcal{G}_1 = 2$$

  is returned as output, thus the problem's postcondition is satisfied. Also, we can see that at steps 1, 3 and 4 no undocumented inputs or global data are accessed or modified by the algorithm. Thus the part **(1)** is satisfied.

- When $n = 2$, the execution of algorithm eventually ends after the execution of steps 1, 3, 5 and 6, and

  $$\mathcal{G}_n = \mathcal{G}_2 = 3$$

  is returned as output, thus the problem's postcondition is satisfied. Also, we can see that at steps 1, 3, 5 and 6 no undocumented inputs or global data are accessed or modified by the algorithm. Thus the part **(1)** is satisfied.

- When $n = 3$, the execution of algorithm eventually ends after the execution of steps 1, 3, 5, 7 and 8, and

  $$\mathcal{G}_n = \mathcal{G}_3 = 4$$

  is returned as output, thus the problem's postcondition is satisfied. Also, we can see that at steps 1, 3, 5, 7 and 8 no undocumented inputs or global data are accessed or modified by the algorithm. Thus the part **(1)** is satisfied.

- When $n \geq 4$, the steps on 1, 3, 5, 7 and the steps from 9 to 14 are executed before reaching the *while* loop.
  - If the execution of the loop terminates, then it follows by the loop invariant stated in **Question 6** which is proved to be true in **Question 7**, thus i is an integer variable such that $4 \leq i \leq n + 1$ and $G[j] = \mathcal{G}_j$ for every integer j such that $0 \leq j \leq i - 1$.

Since the execution of the loop terminates, the loop test at step 15 has been checked and failed, thus $i > n$, that is $n < i \leq n+1$, thus $i = n+1$, so $G[j] = \mathcal{G}_j$ for every integer j such that $0 \leq j \leq (n+1)-1$, that is, $0 \leq j \leq n$.
In this case, the execution of the algorithm eventually ends at step 10, where

$$G[n] = \mathcal{G}_n$$

is returned as output. Also, we can see that no undocumented inputs or global data are accessed or modified by the algorithm in this case. Thus the part **(1)** is satisfied.

- If the execution of the loop does not terminate, then the execution of the algorithm certainly does not end. Thus the part **(2)** is satisfied.

Since all cases of n that satisfied the problem's precondition discussed above satisfies part **(1)** or part **(2)** in the definition of **partial correctness**, we can conclude that the claim holds and the algorithm is partially correct.

Question 9

CITATION
Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 38 to page 39 in L03_iterative_correctness.pdf[4] in order to providing a completed, clear and professional proof.

Claim A bound function for the *while* loop in this algorithm is

$$f(n,i) = n - i + 1$$

Proof

- From the problem's precondition, we can see that n is an integer input, since i is also an integer variable from step 14, $n - i + 1$ is an integer, thus $f(n,i)$ is an integer-valued function.
- Since the value of i is increased by 1 at step 19 after the execution of the loop body of *while* and n is not changed, the value of $f(n,i)$ is decreased by 1.
- If the value of $f(n,i) \leq 0$, we have $n - i + 1 \leq 0$, that is $i \geq n+1$, in this case the loop test at step 15 will fail when check.

Therefore, since all properties of a definition of a **bound function** for a *while* loop are satisfied, $f(n,i) = n - i + 1$ is a bound function for the *while* loop in this algorithm.

Question 10

CITATION
Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 42 to

page 43 in L03_iterative_correctness.pdf[4] in order to providing a completed, clear and professional proof.

Claim If the 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.

Proof Since the precondition of the problem is satisfied, a non-negative integer n is given as input.

- When $n = 0$, the test at step 1 is checked and passed, and the execution of algorithm eventually ends after the execution of step 2, as required.
- When $n = 1$, the test at step 1 failed, and the execution continues with the test at step 3, which is checked and passed, and the execution of algorithm eventually ends after the execution of step 4, as required.
- When $n = 2$, the tests at step 1 and 3 failed, and the execution continues with the test at step 5, which is checked and passed, and the execution of algorithm eventually ends after the execution of step 6, as required.
- When $n = 3$, the tests at step 1, 3 and 5 failed, and the execution continues with the test at step 7, which is checked and passed, and the execution of algorithm eventually ends after the execution of step 8, as required.
- When $n \geq 4$, the tests at step 1, 3, 5, 7 all failed, so that steps 9, 10, 11, 12, 13, 14 are reached and executed.
    - The loop test at step 15 has no side-effects because it is a boolean expression to compare if the value of i is less than or equal to the value of n without changing values of inputs, variables or global data.
    - If i is even, only steps 16, 17 and 19 in the loop body will be executed. Since step 16 is a simple boolean expression to test if i can be divisible by 2, step 17 is a recursive an assignment statement, step 19 is a simple assignment statement, thus the execution of the loop body will terminate after the step 19 is executed.
    - If i is odd, only steps 16, 18 and 19 in the loop body will be executed. Since step 16 is a simple boolean expression to test if i can be divisible by 2, step 18 is a recursive an assignment statement, step 19 is a simple assignment statement, thus the execution of the loop body will terminate after the step 19 is executed.
    - The claim in **Question 9** states that the *while* loop of this algorithm has a bound function and is proved to be true.

Therefore, by **Loop Theorem #2**, we can conclude that if the 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.

Question 11

CITATION

Please note before grading, the reason why it is sufficient to prove that the algorithm is correct by showing the algorithm is both partially correct and terminates whenever it is executed when the problem's precondition initially satisfied is provided at page 44 in L03_iterative_correctness.pdf[4].

Claim The *fGrin* algorithm correctly solves the **Grindelwald Gaggle Computation** problem.

Proof In order to prove that the correctness of the algorithm, it is sufficient to show the algorithm is both partially correct and terminates whenever it is executed when the problem's precondition initially satisfied.

- By inspection the 'Assertion' comment of the algorithm in Figure 2, we can see the problem's precondition is satisfied.
- The algorithm is proved to be partially correct in **Question 8**.
- From the claim stated in **Question 10** which is proved to be true, we can see the execution of *while* loop eventually ends when the precondition of the problem is satisfied, thus the execution of the algorithm will also end.

Therefore, all 3 points above are sufficient to establish the claim, thus the *fGrin* algorithm correctly solves the **Grindelwald Gaggle Computation** problem.

Question 12

CITATION
Please note before grading, the format of the solution acts in a similar manner to the solution provided from page 7 to page 10 in L05_efficiency.pdf[3] in order to providing a completed, clear and professional solution.

Solution Since a natural number n is given as input and **Uniform Cost Criterion** is used to define steps of the execution, we have

- If $n = 0$, the test at step 1 is checked and passed. The execution continues and ends with the *return* statement at step 2, since there are totally 2 steps,

$$T_{fGrin}(n) = T_{fGrin}(0) = 2$$

- If $n = 1$, the test at step 1 failed and the test at step 3 is checked and passed. The execution continues and ends with the *return* statement at step 4, since there are totally 3 steps,

$$T_{fGrin}(n) = T_{fGrin}(1) = 3$$

- If $n = 2$, the tests at step 1, 3 failed and the test at step 5 is checked and passed. The execution continues and ends with the *return* statement at step 6, since there are totally 4 steps,

$$T_{fGrin}(n) = T_{fGrin}(2) = 4$$

- If $n = 3$, the tests at step 1, 3, 5 failed and the test at step 7 is checked and passed. The execution continues and ends with the *return* statement at step 8, since there are totally 5 steps,

$$T_{fGrin}(n) = T_{fGrin}(3) = 5$$

- If $n \geq 4$, the tests at step 1, 3, 5, 7 all failed and the execution continues with the steps from line 9 to line 14, so 10 steps have been executed before the *while* loop.

  Since $f(n, i) = n - i + 1$ is proved to be a bound function of the loop in **Question 9** for this algorithm and its initial value is $f(n, 4) = n - 4 + 1 = n - 3$, the loop body is executed at most $n - 3$ times. and the loop test is executed at most $n - 2$ times. According to the parity of n, we can classify the value of n into 2 cases.

  - *Case 1 (n is even):* If n is even, three steps(at line 16, 17, 19) in the loop body will be executed, so the total number of steps used in the execution of the loop is at most

$$\begin{aligned}\sum_{i=1}^{n-3} 3 + \sum_{i=1}^{n-2} 1 &= 3(n-3) + (n-2) \\ &= 3n - 9 + n - 2 \\ &= 4n - 11\end{aligned}$$

    if $n \geq 4$ and n is even.

  - *Case 2 (n is odd):* If n is odd, three steps(at line 16, 18, 19) in the loop body will be executed, so the total number of steps used in the execution of the loop is at most

$$\begin{aligned}\sum_{i=1}^{n-3} 3 + \sum_{i=1}^{n-2} 1 &= 3(n-3) + (n-2) \\ &= 3n - 9 + n - 2 \\ &= 4n - 11\end{aligned}$$

    if $n \geq 4$ and n is odd.

  Therefore, if $n \geq 4$, no matter n is even or odd, the total number of steps used in the execution of the loop is at most

$$4n - 11$$

  Since a *return* statement at line 20 is executed after line 19. Thus, when $n \geq 4$, the number of steps executed is at most

$$10 + (4n - 11) + 1 = 4n$$

Thus, when a natural number n is given as input and Uniform Cost Criterion is used to define steps of the execution, an upper bound $T_{fGrin}(n)$, can be expressed as

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

Question 13

NOTE The file *FGrindelwald.java* has been uploaded to D2L dropbox.

Question 14

CITATION
Please note before grading, the format of the proof acts in a similar manner to the proof provided from page 48 and page 49 in L01_intro_and_math_review.pdf[2] in order to providing a completed, clear and professional proof.

Claim For every non-negative integer n, the expression for the n^{th} Grindelwald number is

$$\mathcal{G}_n = n + 1$$

Proof We prove the claim by the strong form of mathematical induction on n. Cases that $n = 0$, $n = 1$, $n = 2$, $n = 3$ will be used in the basis.

Basis (n=0) When $n = 0$, $\mathcal{G}_n = \mathcal{G}_0 = 1 = 0 + 1 = n + 1$, as required.

Basis (n=1) When $n = 1$, $\mathcal{G}_n = \mathcal{G}_1 = 2 = 1 + 1 = n + 1$, as required.

Basis (n=2) When $n = 2$, $\mathcal{G}_n = \mathcal{G}_2 = 3 = 2 + 1 = n + 1$, as required.

Basis (n=3) When $n = 3$, $\mathcal{G}_n = \mathcal{G}_3 = 4 = 3 + 1 = n + 1$, as required.

Inductive Step: Let $k \geq 3$ be an integer. It is necessary and sufficient to use

Inductive Hypothesis: Suppose that m is a non-negative integer. Suppose for all integers m such that $0 \leq m \leq k$,

$$\mathcal{G}_m = m + 1$$

to prove

Inductive Claim:

$$\mathcal{G}_{k+1} = (k + 1) + 1$$

Since $k \geq 3$, $k + 1 \geq 4$. According to the parity of $k + 1$, we can classify the value of $k + 1$ into 2 cases.

Case 1 (k + 1 is even):

Since $k+1 \geq 4$ and $k+1$ is even, by the definition of recurrence,

$$\mathcal{G}_{k+1} = 2\mathcal{G}_{(k+1)-1} - 2\mathcal{G}_{(k+1)-3} + \mathcal{G}_{(k+1)-4}$$

That is,

$$\mathcal{G}_{k+1} = 2\mathcal{G}_{k} - 2\mathcal{G}_{k-2} + \mathcal{G}_{k-3}$$

Since $k \geq 3$, we have $0 \leq k, k-2, k-3 \leq k$, thus by the **inductive hypothesis**,

$$\begin{aligned}\mathcal{G}_{k+1} &= 2(k+1) - 2((k-2)+1) + ((k-3)+1) \\ &= (2k+2) - 2(k-1) + (k-2) \\ &= 2k + 2 - 2k + 2 + k - 2 \\ &= k + 2 \\ &= (k+1) + 1\end{aligned}$$

as required in this case.

Case 2 (k + 1 is odd):

Since $k+1 \geq 4$ and $k+1$ is odd, by the definition of recurrence,

$$\mathcal{G}_{k+1} = \mathcal{G}_{(k+1)-1} + 3\mathcal{G}_{(k+1)-2} - 5\mathcal{G}_{(k+1)-3} + 2\mathcal{G}_{(k+1)-4}$$

That is,

$$\mathcal{G}_{k+1} = \mathcal{G}_{k} + 3\mathcal{G}_{k-1} - 5\mathcal{G}_{k-2} + 2\mathcal{G}_{k-3}$$

Since $k \geq 3$, we have $0 \leq k, k-1, k-2, k-3 \leq k$, thus by the **inductive hypothesis**,

$$\begin{aligned}\mathcal{G}_{k+1} &= (k+1) + 3((k-1)+1) - 5((k-2)+1) + 2((k-3)+1) \\ &= (k+1) + 3k - 5(k-1) + 2(k-2) \\ &= (k+1) + 3k - (5k-5) + (2k-4) \\ &= k + 1 + 3k - 5k + 5 + 2k - 4 \\ &= k + 2 \\ &= (k+1) + 1\end{aligned}$$

as required in this case.

In both cases, $\mathcal{G}_{k+1} = (k+1) + 1$ when $k+1 \geq 4$, thus the ***inductive claim*** is established.

Conclusion: Therefore, by strong form of mathematical induction, we can conclude that, for every non-negative integer n, the expression for the n^{th} Grindelwald number is $\mathcal{G}_n = n + 1$.

References

[1] Wayne Eberly, 2019, CPSC 331: Data Structures, Algorithms, and Their Analysis: Spring, 2019, Introduction to the Analysis of Algorithms, Lecture #3: Introduction to the Correctness of Algorithms II — Correctness of Simple Algorithms with a while Loop. Retrieved from http://pages.cpsc.ucalgary.ca/~eberly/Courses/CPSC331/2019a/2_Analysis/L02/L02_correctness.pdf

[2] Wayne Eberly, 2019, CPSC 331: Data Structures, Algorithms, and Their Analysis: Spring, 2019, Introduction and Mathematics Review, Lecture #1: Introduction and Mathematics Review. Retrieved from http://pages.cpsc.ucalgary.ca/~eberly/Courses/CPSC331/2019a/1_Introduction/L01/L01_intro_and_math_review.pdf

[3] Wayne Eberly, 2019, CPSC 331: Data Structures, Algorithms, and Their Analysis: Spring, 2019, Introduction to the Analysis of Algorithms, Lecture #5: Analyzing the Running Times of Algorithms. Retrieved from http://pages.cpsc.ucalgary.ca/~eberly/Courses/CPSC331/2019a/2_Analysis/L05/L05_efficiency.pdf

[4] Wayne Eberly, 2019, CPSC 331: Data Structures, Algorithms, and Their Analysis: Spring, 2019, Introduction to the Analysis of Algorithms, Lecture #3: Introduction to the Correctness of Algorithms II — Correctness of Simple Algorithms with a while Loop. Retrieved from http://pages.cpsc.ucalgary.ca/~eberly/Courses/CPSC331/2019a/2_Analysis/L03/L03_iterative_correctness.pdf