**Assignment 1 CPSC 331**

**Vladislav Chestnykh ID 10147034**

**Ben Coom ID 30037563**

Citation: Before grading this assignment please note that some of the proofs may be formatted in a similar manner to the proofs provided in the lecture notes for the course. This is for the sake of providing a professional and comprehensive proof.

1.     To prove f(n) = n is a bound function for the recursive algorithm sHuff, we must prove all 3 properties of recursive bound functions:

a.     Since n is an specified to be an integer input then this is an integer valued-function.

b.   On line 9, it is clear that any time the function is called recursively, the value of the parameter n is reduced by at least one, thus the bound function f(n) = n is being reduced by at least one.

c.     In the sHuff algorithm, the value of n must be >= 0 as stated in the precondition.

Thus the only way for the function to be called with n <= 0 and for the precondition (n 0) to be satisfied is when n = 0

When n = 0, the test at line 1 passes, the program continues on to line 2 where execution ends, thus it is not called recursively when n <= 0.

Since all 3 conditions are satisfied, then it is a bound function for the recursive sHuff algorithm.

2. Proof of the following claim:

**Claim**: For every non-negative integer n, if the sHuff algorithm is executed with n as input then this this execution of the algorithm eventually ends, and the n th Hufflepuff number, Hn, is returned as output.

**Theorem**: Suppose n is a non-negative integer and that the algorithm sHuff is executed given n as input. Then the execution ends and returns the nth Hufflepuff number Hn as output.

**Proof**: 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 10, which is the H0th  Hufflepuff number. Thus the 0th Hufflepuff number 10 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 9, which is the H1st  Hufflepuff number. Thus the 1st Hufflepuff number 9 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 8, which is the H2nd  Hufflepuff number. Thus the 2nd Hufflepuff number 8 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 7, which is the H3rd  Hufflepuff number. Thus the 3rd Hufflepuff number 7 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 sHuff executes using n as input, then it eventually ends and  the Hn Hufflepuff number will be returned as output.

**Inductive Claim:** If the algorithm sHuff is executed with n = k+1 as input, then the algorithm eventually ends and the Hk+1 Hufflepuff number is returned as output.

**Proof:**  Suppose that the algorithm sHuff 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), (n2), (n-3), (n-4) as arguments.

·   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 Hn-1 = Hk is returned as output.

·   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 Hn-2 = Hk-1 is returned as output.

·   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 Hn-3 = Hk-2 is returned as output.

·   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 Hn-4 = Hk-3 is returned as output.

Thus one can see that by inspection of line 9 that the algorithm eventually ends. Furthermore, since k+1 >= 3, then by the definition of Hufflepuff numbers

Hk+1 = 4 \* Hk -6 Hk-1 + 4 \* Hk-2 – Hk-3

Thus the inductive claim is established.

Therefore by the property of strong induction, it is clear that if the algorithm sHuff is executed given a non-negative integer n as input, then the execution eventually ends and the nth Hufflepuff number Hn is returned as output.

3. In the previous question, we proved the following claim:

Claim: For every non-negative integer n, if the sHuff algorithm is executed with n as input then this this execution of the algorithm eventually ends, and the n th Hufflepuff number, Hn, 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 n th Hufflepuff 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 sHuff correctly solves the Hufflepuff number problem.

4. See the Java file. **Note: The Following code was pasted in as requested by the instructor at the last minute. Please look at the Java Files submitted on D2l for proper code.**

package cpsc331.A1;

//Submission for Ben Cook, 30037563

//and Vladislav Chestnykh, 10147034

import java.util.\*;

class Hufflepuff{

   Hufflepuff(){}

   public static int eval(int n)throws IllegalArgumentException{

   if (n < 0) {

    throw new IllegalArgumentException();

   }else if (n == 0) {

   //Assertion a non negative integer n has been passed to the method

   //Assertion n == 0

  return 10;

} else if (n == 1) {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n == 1

  return 9;

} else if (n == 2) {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n == 2

  return 8;

} else if (n == 3) {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n == 3

  return 7;

   } else {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n >= 4

  int hocus = 10;

      int pocus = 9;

      int abra = 8;

  int kadabra = 7;

  int i = 3;

      while(i < n){

/\* Loop Invariant Assertions:

n is a non negative integer >= 4

i is an integer such that 3 <= i < = n

kadabra is an integer such that kadabra = Hi for i >= 3

abra is an integer such that abra = Hi-1 for i >= 3

pocus is an integer such that pocus= Hi-2 for i >= 3

hocus is an integer such that hocus= Hi-3 for i >= 3 \*/

int shazam = 4 \* kadabra - 6 \* abra + 4 \* pocus - hocus;

hocus = pocus;

pocus = abra;

abra = kadabra;

kadabra = shazam;

i++;

  }

  return kadabra;

   }

  //Assertion a non negative integer n has been provided to the method as input

  //Assertion nth Hufflepuff number has been returned as output

   }

   public static void main(String[] args){

//Assertion an array of strings is passed

   if (args.length != 1){

//Assertion an array of strings passed of length not equal to 1

      System.out.println("Silly muggle! One integer input is required.");

       }else{

//Assertion length of the array passed is 1

       try{

//Assertion first member of array might be an integer

   int inp = Integer.parseInt(args[0]);

   if (inp < 0){

//Assertion first member of array converted is an integer

//Assertion inp is a negative integer

   System.out.println("Silly muggle! The integer input cannot be negative.");

   }else{

//Assertion first member of array converted is an integer

//Assertion inp is a positive or 0 integer

   System.out.println(eval(inp));

       }

       }catch(Exception e){

          System.out.println("Silly muggle! One integer input is required.");

       }

   }

   }

}

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

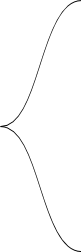
When (n = 0): Then test at line 1 passes, and line 2 is executed, thus ending the execution. So then tsHuff (0) = 2.

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 tsHuff (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 tsHuff (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 tsHuff (3) = 5.

When (n >= 4), then the tests at line 1, 3, 5 and 7 fail, so line 9 is executed with the recursive calls. Thus the tsHuff (n) when n > 3 is TsHuff (n-1)+  TsHuff (n-2)+  TsHuff (n-3) +  TsHuff (n-4) + 5.



2 when n = 0

3 when n = 1

4 when n = 2

tsHuff(n)

5 when n = 3

TsHuff (n-1)+  TsHuff (n-2)+  TsHuff (n-3) +  TsHuff (n-4) + 5. when n >= 4

6. Proof that  tsHuff(n) >= (3/2)n  for all n >= 0. We will prove this by strong mathematical induction.

**Basis:** (n = 0), then  tsHuff(n) = 2 >= 1 =  (3/2)0

(n = 1), then  tsHuff(n) = 3 >= 3/2 = (3/2)1

(n = 2), then  tsHuff(n) = 4 >= 9/4 = (3/2)2

(n = 3), then  tsHuff(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 tsHuff(i) >= (3/2)i.

**Inductive Claim:**  The inductive claim is  tsHuff(k+1) >= (3/2)k+1.

**Proof:** tsHuff(k+1)  = TsHuff (k) +  TsHuff (k-1) +  TsHuff (k-2) +  TsHuff (k-3) + 5. By the recurrence

Then by the inductive hypothesis:

TsHuff (k)+  TsHuff (k-1)+  TsHuff (k-2) +  TsHuff (k-3) + 5 >= (3/2)k + ( 3/2)k-1 + (3/2)k-2+ (3/2)k-3 + 5 =

(3/2)k+1 ( (3/2)-1 + ( 3/2)-2 + (3/2)-3+ (3/2)-4) + 5  = (3/2)k+1 (130/81) + 5 >= (3/2)k+1

Thus this proves that for all integers n >= 0,  TsHuff (n) >= (3/2)n

7.The loop invariant for the hufflepuff algorithm provided in Figure 5 is:

Loop invariant:

I. n is a non negative integer >= 4

II. i is an integer such that 3 <= i < = n

III. kadabra is an integer such that kadabra = Hi for i >= 3

IV. abra is an integer such that abra = Hi-1 for i >= 3

V. pocus is an integer such that pocus= Hi-2 for i >= 3

VII. hocus is an integer such that hocus= Hi-3 for i >= 3

Proof of loop invariant:

1. 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.

1. 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 8 and 9.

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

On line 13 i is assigned 3.

On line 9 hocus is assigned 10 which is H0  = H3-3 = Hi-3

On line 10 pocus is assigned 9 which is H1  = H3-2 = Hi-2

On line 11 abra is assigned a value of 8 which is H2 = H3-1 = Hi-

On line 12 kadabra is assigned a value of 7 which is H3=Hi.

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.

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

From the proof in part b, 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.

Since n >= 4 as stated in part b, 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 = n - 1 because if i = n, then the loop would terminate.

At the end of the final execution of the loop on line 20, variable i is incremented by 1, so that i = n, thus 3 <= i <= n at the end of the loop

On the line 15 shazam is assigned a value of 4 \* kadabra - 6 \* abra + 4 \* pocus − hocus.  It follows that this equation can be rewritten as 4 \* Hi - 6 \* Hi-1 + 4 \* Hi-2 - Hi-3 thus assigning it a value of Hi+1 . Since i = n-1 on the final iteration, then  n = i+1, thus

shazam = Hn. After line 20 i is incremented by 1 so i = n, thus shazam Hn = Hi.

On line 16 hocus is being assigned a value of Hi-2 = Hn-3 on the final iteration, when i is incremented so that i = n, then hocus = Hi-3.

On line 17 pocus is assigned a value of abra and thus can be written as  Hn-2 = Hi-2 once i is incremented so that i = n at the end of the loop.

On line 18 abra is assigned value of kadabra which is equal to Hn-1 = Hi-1 once i is incremented so that i = n at the end of the loop.

On line 19 kadabra is assigned shazam which is Hn = Hi when i is incremented so that

i = n at the end of the loop.

From the proofs above it follows that the loop invariant holds and is correct by loop theorem #1.

8. Proof of partial correctness of algorithm provided in Figure 5:

To prove partial correctness, we will prove that

A. the algorithm eventually ends with the number Hn 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 H0 = 10.

If n = 1, then execution ends at line 4 which returns H1 = 9.

If n = 2, then execution ends at line 6 which returns H2 = 8.

If n = 3, then execution ends at line 8 which returns H3 = 7

If n >= 4, then the loop will execute on line 14 will execute. Thus this loop either terminates, or it does not:

If the loop terminates, then the execution will end with the variable kadabra returned on line 21. By the loop invariant proved in question 7, variable kadabra = Hi = Hn on the final iteration of the loop.

Thus if the loop ends, then the Hn Hufflepuff number will be returned as output satisfying the problems post-condition proving part A.

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

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 in Figure 5 is: f(n,i)=n-i

Proof:

1. 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 13, i is declared to be an integer with a value of 3. In order for the loop to be reached and the test on line 14 to pass, n must be an integer >= 4.

Since the value of n is never changed, and the value of i is increases after each iteration, then the bound function n-i decreases by 1 each loop iteration.

1. 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 20 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 = 0. If n = i, then the loop test on line 14 fails. Thus the loop test fails when the bound function n - i = 0.

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

10. 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

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

In question 8, we proved that the algorithm is partially correct.

In question 9, we proved that a bound function exists.

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

11. See the Java file. **Note: The Following code was pasted in as requested by the instructor at the last minute. Please look at the Java Files submitted on D2l for proper code.**

package cpsc331.A1;

import java.util.\*;

//Submission for Ben Cook, 30037563

//and Vladislav Chestnykh, 10147034

class SHufflepuff{

   SHufflepuff(){}

   protected static int sHuffle(int n) throws IllegalArgumentException{

      if (n < 0) {

       throw new IllegalArgumentException();

  }else if (n == 0) {

   //Assertion a non negative integer n has been passed to the method

   //Assertion n == 0

  return 10;

} else if (n == 1) {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n == 1

  return 9;

} else if (n == 2) {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n == 2

  return 8;

} else if (n == 3) {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n == 3

  return 7;

   } else {

  //Assertion a non negative integer n has been passed to the method

  //Assertion n >= 4

  return 4 \* sHuffle(n - 1) - 6 \* sHuffle(n - 2) + 4 \* sHuffle(n - 3) - sHuffle(n - 4);

   }

  //Assertion a non negative integer n has been provided to the method as input

  //Assertion nth Hufflepuff number has been returned as output

   }

   public static void main(String[] args){

//Assertion args is an array of strings

   if (args.length != 1){

//Assertion length of the array passed is not 1

      System.out.println("Silly muggle! One integer input is required.");

       }else{

//Assertion length of args is 1

       try{

//Assertion first member of args might be an int

       int inp = Integer.parseInt(args[0]);

       if (inp < 0){

//Assertion inp is a negative integer

           System.out.println("Silly muggle! The integer input cannot be negative.");

       }else{

//Assertion inp is a positive or 0 integer

           System.out.println(sHuffle(inp));

       }

        }catch(Exception e){

        System.out.println("Silly muggle! One integer input is required.");

        }

   }

   }

}

12. loop theorem 2 will be used to determine the upper bound.

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

From question 8,9 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 tHuff (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 tHuff (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 tHuff (2) = 4.

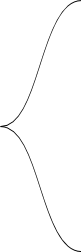
When (n = 3) : Tests at line 1, line 3 and line 5 fail,, but the test on line 7 passes, and line 8 is executed, thus ending the execution. So then tHuff (3) = 5.

When (n >= 4) : Tests at line 1, line 3, line 5 and line 7 fails, so lines 9 to 13 are executed before entering the while loop. (9 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 6 instructions, Thuff (n) can be written as:

Thuff (n) = 6(n-3)  + (n-2) + 10 = 6n - 18 + n - 2 + 10 = 7n - 10 when n >= 4.

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



Thuff (n) = 2 when n=0,

Thuff (n) = 3 when n=1,

Thuff (n) = 4 when n=2,

Thuff (n)

Thuff (n) = 5 when n=3,

Thuff (n) = 7n - 10 when n>=4,

**Bonus:**

The relationship between Hn and n is as follows:

Hhuff (n) = 10-n for any integer n >= 0

We will prove this by Strong induction:

Claim: Hhuff (n) = 10-n for any integer n >= 0

**Proof:**

Basis (n = 0): 10 - n =10 - 0 =10 = Hhuff (n)

Basis (n = 1): 10 - n = 10 - 1 = 9 = Hhuff (n)

Basis (n = 2): 10 - n = 10 - 2 = 8 = Hhuff (n)

Basis (n = 3): 10 - n = 10 - 3 = 7 = Hhuff (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 Hhuff (x) = 10 - x.

**Inductive Claim:** Hhuff (k+1) = 10 - (k+1)

**Proof:**

From the definition of  Hhuff (k+1) = 4Hk -6Hk-1 +4Hk-2 -Hk-3

From the IH: Hhuff  (k+1) = 4(10 - k) -6(10 - (k - 1)) + 4( 10 - ( k - 2)) -( 10 - (k-3))

= 40 - 4k - 60 + 6(k-1) + 40 - 4k + 8 - 10 + k - 3

=9 - k

=10 - (k+1)

**Conclusion**: Thus we can conclude by the property of induction that Hhuff (n) = 10-n for any integer n >= 0.