ASSIGNMENT #1

SUBMITTED BY: Kulsoom Khurshid

REGISTRATION #: SP20-BCS-044

COURSE: Design and Analysis of Algorithms

SUBMITTED TO: Tanveer Ahmed

QUESTION #1

You are running a game booth at your local village festival. In your game you lay out an array of Rs.1 coins on a table. For a Rs. 5 charges, anyone can challenge you to a game. In this game, you and your customer alternately pick coins from the table, either 1 or 2 at a time. If your customer can make you pick up the last coin, (s) he walks away with all the coins. You graciously allow your customer to go first. Being an enterprising sort, you want to arrange the game so that you will always win. To do this, you write an iterative algorithm CarniCoin (n) that tells you how many coins to pick up on every turn. The algorithm takes one parameter n, the number of coins on the table at the beginning of the game.

a) **What pre-condition must you place on n in order to guarantee that you will win every time?**
   Ans:
   n > 0 which satisfies the form 3*n + 1 (n is the input).
b) **What is the post- condition for your algorithm?**
   Ans:
   Customer must lose and that is possible only when single coin is left and he has to pick it.
c) **Write a pseudo-code for algorithm CoinGame (n)**
   Ans:

```
//input: n should be positive number that satisfies the condition 3*n + 1
//output: a single coin left which is picked by the customer
C <- 5 #customer coins to enter game
n <- generate any number randomly between 1-9 and it is assigned to it
C <- C + 3*n-1 #total coins for the game
turn <- True #to ensure customer's turn first
n <- n + 1
#Number should always be at 3n-2
while(C > 0) do
    p <- int(input("Enter 1 or 2 ")) #customer enter the turn
    C <- C - p
    if turn and coins == 0 then
        break #exit the loop
    turn <- False #for computer's turn

    num <- 3*n-2
    p <- C - num
    C <- C - p
    n <- n - 1
    if turn == False and coins == 0 then
        break
    turn = True
end while
if turn then
   Customer loss
else
   Computer loss
```

d) What is the loop invariant for your algorithm?
Ans:
3*n - 2

e) Prove the correctness of your algorithm. Be sure to explicitly include all steps required to prove correctness.
Ans:
Initialization:
Since, the number that is generated randomly is between the range of 1-9 so the minimum value of n = 2. So total coins for the game becomes, **5** coins of the customer and **3*2 + 1** which is equal to **7**. So C=12. To ensure customer's first turn is kept = true.
Maintenance:
The customer should pick when it is 3*n-2 like 1,4,7,10,13,16,19 and so on as these are the terms where customer loses. If customer picks two coin computer will pick one.
Termination:
After loop is terminated coins are 0 and it was picked by customer so the customer will lose.

QUESTION #2

Consider the following three different algorithms that solve the same problem:

a) What do these algorithms compute?
Ans:
These algorithms compute the product of two positive integers(y and z) i.e. p = y*z.
y >= 0 and z >= 0.

b) What are their pre-condition and post condition?
Ans:
Pre-condition: The integers should be positive i.e. y >= 0 and z >= 0.
Post-condition: The product of the two positive integer y and z has been returned in variable x.

c) Clearly define your loop invariant for the given algorithms.
Ans:
Loop invariant: $x_n + y_n*z_n$, it is the property of loop that is true during the entire execution of the code. Hence, it is considered the loop invariant.

d) Prove correctness of above algorithms by showing initialization, maintenance and termination. For Algorithm-3, c ≥ 2
Ans:
Initialization:
To prove P(n) = P(0) is true which means when n = 0 at 0th iteration. It corresponds to the situation before the loop begins to execute.
When n = 0, $x_0 = 0$ (Statement 1). So, P(0) is true.
Maintenance:
Assuming P(k) is true, that is P(k): $x_k + y_k*z_k$ ------------- (a)
If P(k) is true then P(k+1) should also be true that is P(k+1): $x_{k+1} + y_{k+1} * z_{k+1}$-------------(b)
After (k+1)th iteration
$x_{k+1} = x_k + y_k * (z_k \% c)$
$y_{k+1} = c * y_k$
$z_{k+1} = z_k / c$

Let c = 2, y = 3, z = 3, x = 0
3> 0 so loop begins
1st iteration: $x_1 = 0 + 3*(3\%2)$
$y_1 = 2 * 3$
$z_1 = 3 / 2$
1> 0 so loop continues
2nd iteration: $x_1 = 3 + 6*(1\%2)$
$y_1 = 2 * 6$
$z_1 = 1 / 2$

P(k): $x_k + y_k*z_k$
P(1): 3 + 6 * 1 = 9
P(2): 9 +12 * 0 = 9
P(k): $x_k + y_k*z_k = 9$
P(k+1): $x_{k+1} + y_{k+1}*z_{k+1} = 9$
Hence, if P(k) is true then P(k+1) is also true.
Termination:
The loop terminates when z <= 0 after terminating the loop x is returned which basically holds the product of the two positive integers.
P(n): $x_n + y_n * z_n$ is true after the loop termination.

QUESTION #3

Problems 1–5 contain a while loop and a predicate. In each case show by using principle of mathematical induction that if the predicate is true before entry to the loop, then it is also true after exit from the loop.

```
1. loop:    while(m ≥ 0 and m ≤ 100)
                    m := m + 1
                    n := n – 1

            end while
```

Predicate: m + n = 100

Ans:

Pre-condition: m >= 0 and m <= 100 and n is a real number. Such that m + n = 100. Which means that predicate is true before the entry of the loop.

m + n = 100 should be true for kth iteration, if it is then it should also be true for (k + 1)th iteration.

$m_{new} = m + 1$ and $n_{new} = n - 1$

$m_{new} + n_{new} = 100$

m + 1 + n – 1 = 100

m + n =100

Hence, m + n = 100 is the property that is true for all the iterations.

When m > 100 the loop terminates. m + n = 100 is true after the loop termination as well

2. **loop:**
```
while(m ≥ 0 and m ≤ 100)
            m := m + 4
            n := n – 2

end while
```

<u>Predicate: m + n is odd</u>

<u>Ans:</u>

Pre-condition: m >= 0 and m <= 100 and n is a real number. m or n one has to be odd. Which means that predicate is true before the entry of the loop.

m + n = odd num should be true for kth iteration, if it is then it should also be true for (k + 1)th iteration.

$m_{new} = m + 4$ and $n_{new} = n - 2$

$m_{new} + n_{new} = $ odd

m + 4 + n – 2 = odd

m + n + 2 = odd

Hence, m + n = odd and if 2 is added, it will still remain odd. This is the property that is true for all the iterations.

When m > 100 the loop terminates. m + n = odd is true after the loop termination as well

3. **loop:**
```
while(m ≥ 0 and m ≤ 100)
            m := 3*m
            n := 5*n

end while
```

<u>Predicate: $m^3 > n^2$</u>

<u>Ans:</u>

Pre-condition: m >= 0 and m <= 100 and n is smaller than m. Such that $m^3 > n^2$. Which means that predicate is true before the entry of the loop.

$m^3 > n^2$ should be true for kth iteration, if it is then it should also be true for (k + 1)th iteration.

$m_{new} = 3 * m$ and $n_{new} = 5 *n$

$m_{new} > n_{new}$

$(3*m)^3 > (5 *n)^2$

$27m^3 > 25n^2$

Hence, $m^3 > n^2$ is the property that is true for all the iterations.

When m > 100 the loop terminates. $m^3 > n^2$ is true after the loop termination as well

4. **loop:**

```
while(n ≥ 0 and n ≤ 100)
        n := n + 1

end while
```

Predicate: $2^n < (n + 2)!$

Ans:

Pre-condition: n >= 0 and n <= 100. Such that $2^n < (n + 2)!$.

Let n = 2, $2^2 < (2+2)!$

$4 < 24$

2^n is smaller as compared to the factorial. Which means that predicate is true before the entry of the loop.

$2^n < (n + 2)!$ should be true for kth iteration, if it is then it should also be true for (k + 1)th iteration.

$n_{new} = n + 1$

$2^n{}_{new} < (n_{new} + 2)!$

$2^{n+1} < (n + 3)!$

$2^{n+1}/ (n + 3)! < (n + 3)! / (n + 3)!$

$2^{n+1}/ (n + 3)!$ (this will be less than 0) < 1

Hence, $2^n < (n + 2)!$ is the property that is true for all the iterations.

When n > 100 the loop terminates. $2^n < (n + 2)!$ is true after the loop termination as well

5. **loop:**

```
while(n ≥ 3 and n ≤ 100)

        n := n + 1

end while
```

Predicate: $2n + 1 <= 2^n$

Ans:

Pre-condition: n >= 3 and n <= 100. For initial value 3, $2*3 + 1 <= 2^3$

$7 <= 8$. So $2n + 1 <= 2^n$ is true.

$2n + 1 <= 2^n$ should be true for kth iteration, if it is then it should also be true for (k + 1)th iteration.

$2n_{new} + 1 <= 2^{n}_{new}$

$n_{new} = n + 1$

$2(n + 1) + 1 <= 2^{n+1}$

$2n + 3 <= 2^{n+1}$

Let n = 3

$2(3) + 3 <= 2^{3+1}$

9 <= 16

Hence, $2n + 1 <= 2^{n}$ is the property that is true for all the iterations.

When n > 100 the loop terminates. $2n + 1 <= 2^{n}$ is true after the loop termination as well

QUESTION #4

Prove the correctness of following algorithm. Clearly define loop invariant of each algorithm.

1. **Binary Search Algorithm**
   Pre-condition: sorted array size n > 0
   Post-condition: the desired element is found (if element in the subarray)
   Loop invariant: if element is in the list no need to go through the entire array, it can be found in the sub array.
   Initialization:
   The actual array is the subarray initially. If element is present in array it will be found in subarray as well.
   Maintenance:
   Assume P(k) is true, then P(k+1) should also be true. The key value is compared with the mid value of the subarray, if it is equal to key the element is found. The next condition is if it is not equal to key, it is checked if key is smaller than mid value or greater than mid value and accordingly the array is divided into subarray. If the key is smaller than mid, the left side of array is the new subarray else right side.
   Termination:
   Loop terminate when key found or the size of subarray becomes 0.
2. **Merge Sort**
   Pre-condition: The input variable A[1], A[2], . . . A[n] is a one-dimensional array of real numbers.
   Post-condition: The input variable B[1], B[2], . . . B[n] is a one-dimensional array of real numbers with same elements as A[1], A[2], . . . A[n] but with the property that $B[i] \leq B[j]$ whenever $i \leq j$.
   Loop invariant: array A[0,1,….,n-1] has values, (k-n+1) is the smallest element of both sides left and right sorted in order.
   Initialization:
   When n =1, the list contains just one element and hence is clearly sorted, So P(1) is true
   Maintenance:
   As we have proved that P(k): k-n+1 has sorted elements on both sides of arrays. So, P(k+1) should also be true which means the next value that will be inserted in array must be larger than the value of P(k).

Termination:
When the loop variable i=n(size of array) it will terminate since all values will be inserted and sorted.

3. **Quick Sort**
Pre-condition: The input variable A[1], A[2], . . . A[n] is a one-dimensional array of real numbers.
Post-condition: The input variable B[1], B[2], . . . B[n] is a one-dimensional array of real numbers with same elements as A[1], A[2], . . . A[n] but with the property that B[i] ≤B[j] whenever i ≤ j.
Loop invariant: Let P(n) = The algorithm sorts every list of size n ≥ 1.

Initialization:
When n =1, the list contains just one element and hence is clearly sorted, So P(1) is true
Maintenance:
Assume P(k) is true; that is, the algorithm sorts correctly every list of k (≥ 1)
To show that P(k+1) is true, consider a list X = [x_1, x_2 x_3, x_4.... x_k, x_{k+1}]
Since k+1 ≥ 2, for loop in line 1 is entered.
Array is sorted using divide and conquer. The element on the left is smaller than the mid value of an array and right value is greater than mid value. Eventually array is sorted as the left side and right side are already sorted
Termination:
if P(k) is true then k<=n is true for all values.

4. **Finding Minimum number in a list**
Pre-condition: an array of integer size n > 0, A[0,1,….,n-1]
Post-condition: smallest value in the array is returned
Loop invariant:
Initialization:
Initially, P(0): min is the only element in the list . so it is the smallest element in the list as there is no other.
Maintenance:
If P(k) is true then P(k+1) should also be true.
As i < n, min is compared with each new element. If min > new element in the list, min value is replaced with that value.
Termination:
When the loop variable i=n(size of array) it will terminate since all values will be inserted and smallest value is stored in min.