

**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.

**Note:** In order to win the game, we must make sure when we have 1, 4, 7, 10, 13, 16, 19, 22… coins, it is customer’s turn. If the customer picks one, computer should pick two and vice versa. Which means in every turn total 3 coins are removed from an array.

1. **What pre-condition must you place on n in order to guarantee that you will win every time?**

Ans:

The customer gives 5 rupees to enter the game. An array **arr[1 ... 3n-2]** of 1-rupee coins of size **3n+1** where n > 0. Customer should begin the game.

1. **What is the post- condition for your algorithm?**

Ans:

Customer must lose and that is possible only when single coin is left and customer has to pick that.

1. **Write a pseudo-code for algorithm CoinGame (n)**

Ans:

//input: n should be positive number greater than 0 that satisfies the condition 3\*n+1

//output: a single coin left which is picked by the customer

n <- generate random number between 1-9

n <- n+5 #add 5 entry coins given by customer

coins <- 3\*n+1

Arr <- Arr[1……… 3\*n+1]

turn <- true

while(coins > 0)do

    pick <- user input

    coins <- coins - pick

    if(coins == 0)then

        break

end if

    turn <- False

    if pick == 1 then

        coins <- coins - 2

    else

        coins <- coins – 1

end if

    if(coins == 0)then

        break

end if

    turn <- True

if turn then

    print("Customer loss")

else then

    print("Computer loss")

1. **What is the loop invariant for your algorithm?**

Ans:

**3\*n + 1**

Whenever the size of an array is 3\*n + 1, turn is true. When turn is false the size is never equal to 3\*n + 1.

1. **Prove the correctness of your algorithm. Be sure to explicitly include all steps required to prove correctness.**

Ans:

**Initialization:**

Before the loop executes, the variables has the following value

turn = true , n = randomly generated value

coins = 3n + 1, where n > 0

When the size of the array is 3\*n + 1 turn is always true. Hence, the predicate is true before the execution of the loop.

**Maintenance:**

Assuming that P(k) is true for kth iteration, turnk = true and coinsk = 3\*n + 1 where n > 0.

For P(k+2), turnk+2 is true, since we know for P(k) turnk is true, then after two iterations we get true again. So, turnk+2 is also true.

**Termination:**

After loop is terminated coins are 0 and turn is true which shows it was customer’s turn and it was picked by customer so the customer will lose and if we insert 3\*0 + 1 = 1 which shows that the loop invariant is true after the loop termination.

**QUESTION #2**

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

1. **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.

1. **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.

1. **Clearly define your loop invariant for the given algorithms.**

Ans:

Loop invariant: xn + yn\*zn, it is the property of loop that is true during the entire execution of the code. Hence, it is considered the loop invariant.

**Algorithm 1:**

Loop invariant= P(n): xn + yn\*zn

**Algorithm 2:**

Loop invariant= P(n): xn <= yn

**Algorithm 3:**

Loop invariant= P(n): xn + yn\*zn

1. **Prove correctness of above algorithms by showing initialization, maintenance and termination. For Algorithm-3, c ≥ 2**

Ans:

**Algorithm 1:**

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, x0 = 0 (Statement 1). So, P(0) is true.

Maintenance:

Assuming P(k) is true, that is P(k): xk + yk\*zk ------------- (a)

If P(k) is true then P(k+1) should also be true that is P(k+1): xk+1 + yk+1 \* zk+1-------------(b)

After (k+1)th iteration

xk+1 = xk + yk

yk+1 = 2 \* yk

zk+1 = zk / 2

Let y = 3, z = 3, x = 0

3> 0 so loop begins

1st iteration: 3% 2 = 1{x1 ­= 0+3}

y1 = 2 \* 3

z1 = 3 / 2

1> 0 so loop continues

2nd iteration: 1% 2 = 1{x2 ­= 3+6}

y2 = 2 \* 6

z2 = 1 / 2

P(k): xk + yk\*zk

P(1): 3 + 6 \* 1 = 9

P(2): 9 +12 \* 0 = 9

P(k): xk + yk\*zk = 9

P(k+1): xk+1 + yk+1\*zk+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): xn + yn \* zn is true after the loop termination.

**Algorithm 2:**

Initialization:

Before the execution, the value of z >=0 and y >=0. The initial value of x is 0, so in this case x will be less than or equal y (y=0) before the execution.

Maintenance:

Assuming P(k) is true, that is P(k): xk<= yk. So, P(k+1) at k+1 iteration we have,

yk+1 = yk\*2

zk+1 = zk/2

If z is an odd value then,

xk+1= xk+ yk

Therefore, we can state that xk+1 is less than or equal to yk+1.

Predicate P(k+1) is true as P(k+1): xk+1<= yk+1.

Termination:

Loop will terminate once z reaches the value 0. During each iteration, y is being doubled, while x is not always being increased. Therefore x<=y.

**Algorithm 3:**

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, x0 = 0 (Statement 1). So, P(0) is true.

Maintenance:

Assuming P(k) is true, that is P(k): xk + yk\*zk ------------- (a)

If P(k) is true then P(k+1) should also be true that is P(k+1): xk+1 + yk+1 \* zk+1-------------(b)

After (k+1)th iteration

xk+1 = xk + yk \* (zk % c)

yk+1 = c \* yk

zk+1 = zk / c

Let c = 2, y = 3, z = 3, x = 0

3> 0 so loop begins

1st iteration: x1 = 0 + 3\*(3%2)

y1 = 2 \* 3

z1 = 3 / 2

1> 0 so loop continues

2nd iteration: x1 = 3 + 6\*(1%2)

y1 = 2 \* 6

z1 = 1 / 2

P(k): xk + yk\*zk

P(1): 3 + 6 \* 1 = 9

P(2): 9 +12 \* 0 = 9

P(k): xk + yk\*zk = 9

P(k+1): xk+1 + yk+1\*zk+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): xn + yn \* zn 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**

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.

Predicate: mn + nn = 100 (n is the number of iteration)

Before Loop:

According to the pre-condition, the sum of the initial values of m and n equals 100. Which shows that before the loop, the sum of m and n is 100, therefore the loop invariant P(0) is true.

During Loop:

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

P(k) = mold + nold = 100

mnew = mold + 1 and nnew = nold -1

mnew + nnew = mold + 1 + nold – 1

mnew + nnew = mold + nold

from P(k) we know that mk + nk= 100, therefore

mnew + nnew = 100

This proves that the P(k+1): mnew+ nnew= 100 is true

After Loop:

The loop terminates when the value of m > 100. During the iteration of the loop the values were added into m and subtracted from n. Therefore, the sum of m and n is 100 after the loop terminates

1. **loop: while(m > 0 and m < 100)**

**m := m + 4**

**n := n – 2**

**end while**

Ans:

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.

Predicate: P(n): mn + nn is odd

Before Loop:

Based on the precondition the initial values of m and n are such that their sum is odd. Therefore P(0) is true as at 0 iteration the sum is odd.

During Loop:

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

mnew = m­old + 4 and nnew = nold - 2

mnew + nnew = mold + nold + 2

We know that P(k) mold + nold is odd, so adding an odd value to 2 will give mnew + nnew = odd.

This matches with the predicate P(k+1): mk+1 + nk+1 is odd.

After Loop:

When m > 100 the loop terminates. m + n = odd is true after the loop termination as even number is being added and subtracted from m and n so it remains odd or even whatever it is before the loop. So, the predicate is true i.e. m+n=odd.

1. **loop: while(m > 0 and m < 100)**

**m := 3\*m**

**n := 5\*n**

**end while**

Predicate: m3 > n2

Ans:

Pre-condition: m >= 0 and m <= 100 and n is smaller than m. Such that m3 > n2. Which means that predicate is true before the entry of the loop.

Predicate: P(n): m3n > n2n

Before Loop:

Based on the precondition, the m > 0 and n > 0 such that the m3 is greater than n2. This shows that P(0) is true (iteration 0), values of m and n are based on the precondition.

During Loop:

m3 > n2 should be true for kth iteration, if it is then it should also be true for (k + 1)th iteration.

mnew = 3 \* mold and nnew = 5 \*nold

mnew > nnew

(3\*m)3 > (5 \*n)2

27m3 > 25n2

Hence, m3 > n2 is the property that is true for all the iterations.

When m > 100 the loop terminates. During the loop, m is being tripled while n is being increased 5 times. According to the pre-condition n <= m, though n is being increased more than m still cube of m is greater. Hence, m3 > n2 is true after the loop termination as well.

1. **loop: while(n > 0 and n < 100)**

**n := n + 1**

**end while**

Predicate: 2n < (n + 2)!

Ans:

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

Predicate: P(n): 2nn < (nn + 2)!

Before Loop:

The n=0, P(0)= 20<(0+2)! = 1<2 which is true.

After Loop:

The while loop terminates when the n >100. Checking at the loop invariant at the termination of the loop we get

P(101): 2101 < (101+2)!

2.5 \* 103< (103)!

1. **loop: while(n > 3 and n < 100)**

**n := n + 1**

**end while**

Ans:

Pre-condition: n >= 3 and n <= 100.

Predicate: P(n): 2n + 1 <=2n

Before Loop:

For initial value 3, 2\*3 + 1 <= 23

P(0): 7 <= 8. So 2n + 1 <= 2n is true.

During Loop:

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

2nnew + 1 <= 2nnew

nnew = nold + 1

2(nold + 1) + 1 <= 2nnew

2nold + 1 + 2 <= 2nnew

2n +2 <= 2nnew

2nnew <= 2nnew

After Loop:

The loop terminates when the value of n > 100.

So, value of n after loop termination will be 101;

P(101): 2(101) + 1 <= 2101

203 <= 2.53 \* 1030

The predicate is true after the termination as well.

**QUESTION #4**

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

1. **Binary Search Algorithm**

Pre-condition: Takes a positive integer n and search it in Arr[]

Post-condition: return true if found else false

Loop invariant:

The array is sorted in ascending order. The Search Key n is present in the array. Low <= High.

Initialization:

Our array before the first iteration is always sorted in ascending order. The search key is always present in the array before the first iteration. Before the first iteration Lower index is always less than higher index

Maintenance:

Since our current array is sorted in ascending thus the sub array will also be sorted. If the search key is present in the current array thus it will also be present in the sub array. Lower index is still less than higher.

Termination:

When the value of low is greater than high index then the loop will terminate. Since the middle will be average of high and low it is still in between or equal to the high and low indexes.

1. **Merge Sort**

Pre-condition:

Array of integers.

Post-condition:

Merges the two subarrays into the main array

Loop invariant:

Array will be divided into sub array until each number becomes an array and then by comparing it will merge array to become sorted

Initialization:

At start array is divided from mid array will be divided into sub array.

Maintenance:

Loop will satisfy on each step that array is sub divided until each sub array contains only 1 number. After that each number will be compared and merged until sorted

Termination:

The loop terminates, when the value of i=right+1 which means there is Arr[left to right] with (right - left + 1) smallest elements in sorted order. This shows that loop invariants hold even after the termination of the loop.

1. **Quick Sort**

Pre-condition:

Takes an array Arr[] of size n along with the left and right index for this array

Post-condition:

Returns the index for the pivot

Loop invariant:

The elements of array left to the pivot are less than pivot, and the elements of array on its right are larger than the pivot.

Initialization:

When n =0, the list contains just one element and hence is clearly sorted, So P(0) is true

Maintenance:

Assuming that P(k) is true i.e. left side of an array contains elements smaller than pivot. Then P(k+1) is also said to be true if for k+1 iteration it contains smaller elements on the left of pivot. In case, it’s not the case swapping is done to ensure smaller elements on the left.

Termination:

The loop terminates when j=right. So, array is sorting in increasing order

1. **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:

The value min will contain the minimum value from the array of size n, where n >= 1.

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, that min holds the minimum value in the array, then P(k+1) should also be true. Min has the minimum value until kth iteration. On k+1th iteration, 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. Showing the condition is true after termination of the while loop.