**Mathematical Induction**

**Question 1**

Consider the following iterative algorithm to compute the factorial of a positive integer n:

**factorial(n):**

**result = 1**

**for i from 1 to n:**

**result = result \* i**

**return result**

Show that this algorithm is correct using mathematical induction.

**Solution Q.1**

**Base case:** When n = 1, the algorithm computes 1! = 1, which is correct.

**Inductive step:** Assume that the algorithm computes (k-1)! correctly for some k > 1. Then, when the algorithm computes k!, it multiplies (k-1)! by k, which is the correct formula for k!. Therefore, the algorithm is correct for all positive integers n by mathematical induction.

**Question 2**

Consider the following recursive algorithm to compute the nth Fibonacci number:

**fibonacci(n):**

**if n <= 1:**

**return n**

**else:**

**return fibonacci(n-1) + fibonacci(n-2)**

Show that this algorithm is correct using mathematical induction.

**Solution Q.2**

**Base case:** When n = 0, the algorithm computes the 0th Fibonacci number correctly as 0. When n = 1, the algorithm computes the 1st Fibonacci number correctly as 1.

**Inductive step:** Assume that the algorithm computes the kth and (k-1)th Fibonacci numbers correctly for some k > 1. Then, when the algorithm computes the (k+1)th Fibonacci number, it correctly adds the kth and (k-1)th Fibonacci numbers, which is the correct formula for the (k+1)th Fibonacci number. Therefore, the algorithm is correct for all non-negative integers n by mathematical induction.

**Question 3**

Consider the following iterative algorithm to search for a key value in a sorted array of integers:

**binary\_search(key, array):**

**left = 0**

**right = length(array) - 1**

**while left <= right:**

**mid = (left + right) // 2**

**if array[mid] == key:**

**return mid**

**elif array[mid] < key:**

**left = mid + 1**

**else:**

**right = mid - 1**

**return -1**

Show that this algorithm is correct using mathematical induction.

**Solution Q.3**

**Base case:** When the array has length 1, the algorithm correctly checks if the key value is equal to the only element in the array.

**Inductive step:** Assume that the algorithm correctly searches for a key value in a sorted array of length k for some k > 1. Then, when the algorithm searches for a key value in a sorted array of length (k+1), it first compares the key value to the middle element of the array. If the key value is less than the middle element, the algorithm correctly searches the left half of the array. If the key value is greater than the middle element, the algorithm correctly searches the right half of the array. If the key value is equal to the middle element, the algorithm correctly returns the index of the middle element. Therefore, the algorithm is correct for all sorted arrays of integers by mathematical induction.

**Question 4**

Consider the following iterative algorithm to compute the sum of the first n positive integers:

**sum = 0**

**for i = 1 to n do**

**sum = sum + i**

**end for**

**return sum**

Prove the correctness of this algorithm using mathematical induction.

**Solution Q.4**

**Precondition:** n is a positive integer.

**Postcondition:** The algorithm returns the sum of the first n positive integers.

**Base Case:** For n = 1, the algorithm returns 1, which is the correct sum of the first 1 positive integer.

**Inductive Hypothesis:** Assume that the algorithm returns the correct sum of the first k positive integers for some k >= 1.

**Inductive Step:** We need to show that the algorithm also returns the correct sum of the first k+1 positive integers. By the inductive hypothesis, the sum of the first k positive integers is:

1 + 2 + ... + k = k(k+1)/2

Adding k+1 to both sides, we get:

1 + 2 + ... + k + (k+1) = k(k+1)/2 + (k+1) = (k+1)(k+2)/2

Therefore, the algorithm correctly computes the sum of the first k+1 positive integers. By mathematical induction, the algorithm is correct for all positive integers n.

**Loop Invariants**

**Example of Iterative Algorithm with Loop Invariant:**

Consider the following iterative algorithm to find the sum of an array of integers:

**Algorithm**

1. function array\_sum(array A, size n)

2. sum = 0

3. for i = 1 to n

4. sum = sum + A[i]

5. return sum

To prove the correctness of this algorithm using loop invariants, we can define the following loop invariant:

At the start of each iteration of the for loop, **sum** contains the sum of the first **i-1** elements of array **A**.

We can prove the correctness of the algorithm by showing that the loop invariant holds true before the loop starts, after each iteration of the loop, and at the end of the loop:

**Initialization:** Before the loop starts, **i=1** and **sum=0**, so the loop invariant holds true.

**Maintenance:** After each iteration of the loop, **sum** is updated to include the **i**th element of **A**, so **sum** still contains the sum of the first **i-1** elements of **A**. Therefore, the loop invariant holds true.

**Termination:** When the loop terminates, **i=n+1** and **sum** contains the sum of all the elements of **A**, so the loop invariant holds true.

Since the loop invariant holds true before the loop starts, after each iteration of the loop, and at the end of the loop, we can conclude that the algorithm is correct.

**Example of Recursive Algorithm with Loop Invariant:**

Consider the following recursive algorithm to compute the factorial of a non-negative integer

**Algorithm**

1. function factorial(n)

2. if n == 0

3. return 1

4. else

5. return n \* factorial(n-1)

To prove the correctness of this algorithm using loop invariants, we can define the following loop invariant:

At the start of each recursive call to **factorial**, the function correctly computes the factorial of the input argument **n-1**.

We can prove the correctness of the algorithm by showing that the loop invariant holds true before the recursive call, after the recursive call, and at the base case:

**Initialization:** Before the first recursive call, **n=0**, so the loop invariant holds true.

**Maintenance:** After each recursive call to **factorial(n-1)**, we multiply the result by **n**, so the loop invariant holds true.

**Termination:** When the base case is reached, **n=0**, and the function correctly returns **1**, so the loop invariant holds true.

Since the loop invariant holds true before the recursive call, after the recursive call, and at the base case, we can conclude that the algorithm is correct.