Comparing Algorithms with Big O Notation

Time Complexity Deep Dive

By: Zafar Ali Khan

Linear Search:

```
// Declare an array of integers and the target value to search for
DECLARE Numbers : ARRAY[1:10] OF INTEGER
DECLARE Target : INTEGER

// Input the elements of the array and the target value
FOR Index ← 1 TO 10
    INPUT "Enter element ", Index, ": ", Numbers[Index]
NEXT Index

INPUT "Enter the target value: ", Target

// Initialize a flag to indicate if the target value is found
DECLARE Found : BOOLEAN
Found ← FALSE

// Perform a linear search for the target value
FOR Index ← 1 TO 10
    IF Numbers[Index] = Target THEN
        Found ← TRUE
        OUTPUT "Target value found at position: ", Index
        EXIT
    ENDIF
NEXT Index

// If the target value was not found, output a message
IF NOT Found THEN
    OUTPUT "Target value not found in the array."
ENDIF
```

Binary Search Function:

```
FUNCTION BinarySearch(Numbers : ARRAY OF INTEGER, Target : INTEGER)
RETURNS INTEGER
    // Declare variables for binary search
    DECLARE Left, Right, Mid : INTEGER

    // Initialize the search range
    Left ← 1
    Right ← LENGTH(Numbers)

    // Perform binary search
    WHILE Left <= Right
        Mid ← (Left + Right) DIV 2
        IF Numbers[Mid] = Target THEN
            RETURN Mid
        ELSE
            IF Numbers[Mid] < Target THEN
                Left ← Mid + 1
            ELSE
                Right ← Mid - 1
            ENDIF
        ENDIF
    ENDWHILE

    // Target not found, return -1 as an indicator
    RETURN -1
ENDFUNCTION
```

Bubble Sort Function:

```
FUNCTION BubbleSort(Numbers : ARRAY OF INTEGER) RETURNS ARRAY OF INTEGER
    // Declare variables for bubble sort
    DECLARE n, i, j : INTEGER
    DECLARE Swapped : BOOLEAN

    n ← LENGTH(Numbers)

    // Bubble sort algorithm
    FOR i ← 1 TO n - 1
        Swapped ← FALSE

        FOR j ← 1 TO n - i
            IF Numbers[j] > Numbers[j + 1] THEN
                // Swap elements
                SWAP Numbers[j], Numbers[j + 1]
                Swapped ← TRUE
            ENDIF
        NEXT j

        // If no elements were swapped, the array is already sorted
        IF NOT Swapped THEN
            EXIT
        ENDIF
    NEXT i

    RETURN Numbers
ENDFUNCTION
```

Insertion Sort

```
FUNCTION InsertionSort(Numbers : ARRAY OF INTEGER) RETURNS ARRAY OF INTEGER
    // Declare variables for insertion sort
    DECLARE n, i, j, key : INTEGER

    n ← LENGTH(Numbers)

    // Insertion sort algorithm
    FOR i ← 2 TO n
        key ← Numbers[i]
        j ← i - 1

        // Move elements of Numbers[1..i-1] that are greater than key to one position ahead
        WHILE j > 0 AND Numbers[j] > key
            Numbers[j + 1] ← Numbers[j]
            j ← j - 1
        ENDWHILE

        Numbers[j + 1] ← key
    NEXT i

    RETURN Numbers
ENDFUNCTION
```

Big O Notation	Time Complexity	Algorithm Example	Description
O(1)	Constant time	Retrieving the first item in a list	Complexity remains the same regardless of dataset size; very efficient
O(n)	Linear time	Linear search, bubble sort on an already sorted list	Complexity grows linearly with dataset size; larger datasets result in proportionally longer times
O(log n)	Logarithmic time	Binary search	Complexity grows logarithmically with dataset size; very efficient for large datasets
O(n^2)	Quadratic time	Bubble sort, insertion sort	Complexity grows quadratically with dataset size; nested iterations, slower for large datasets

Algorithm	Space Complexity	Description
Linear search	O(n)	Only takes the space required for the dataset; doesn't require extra memory for processing larger lists
Binary search	O(n)	Only takes the space required for the dataset; doesn't require extra memory for processing larger lists
Bubble sort	O(1)	Performs sorting operations "in-place," using the same memory that holds the dataset; doesn't require extra memory for larger lists
Insertion sort	O(1)	Performs sorting operations "in-place," using the same memory that holds the dataset; doesn't require extra memory for larger lists