# Lab 1: Answer Key

Course: Data Structures Instructor Muhammad Kazim, Ph.D.

**Lab Week: 1**

Topics: Algorithm Notation, Data Structure Operations, Complexity Analysis

## Activity 1: Algorithm Design (Sample Answers)

Note: Answers can vary. The key is logical correctness and clarity.

**A. The Even Counter**  
  
**Pseudocode**:  
BEGIN  
 SET count TO 0  
 FOR each number IN the list  
 IF number MOD 2 == 0 THEN  
 INCREMENT count BY 1  
 END IF  
 END FOR  
 RETURN count  
END

**B. The Reverser**  
  
**Pseudocode**:  
BEGIN  
 SET new\_list TO an empty list  
 SET index TO length of list - 1  
 WHILE index >= 0  
 APPEND list[index] TO new\_list  
 DECREMENT index BY 1  
 END WHILE  
 RETURN new\_list  
END

**C. The Palindrome Checker**  
  
**Pseudocode**:  
BEGIN  
 SET left\_index TO 0  
 SET right\_index TO length of word - 1  
 WHILE left\_index < right\_index  
 IF word[left\_index] != word[right\_index] THEN  
 RETURN False  
 END IF  
 INCREMENT left\_index BY 1  
 DECREMENT right\_index BY 1  
 END WHILE  
 RETURN True  
END

**D. The 2D Sum**  
  
**Pseudocode:**  
BEGIN  
 SET total TO 0  
 FOR each row IN the 2D array  
 FOR each number IN the row  
 SET total TO total + number  
 END FOR  
 END FOR  
 RETURN total  
END

## Activity 2: Operation Identification

**Algorithm 1: The Unique Filter**  
unique\_list = [] # Operation: \*\***Creation**\*\* (or Initialization)  
for item in data: # Operation: \*\***Traversal**\*\*  
 if item not in unique\_list: # Operation: \*\***Searching**\*\*  
 unique\_list.append(item) # Operation: \*\***Insertion**\*\*

**Algorithm 2: The Sorted Inserter**  
  
for index in range(len(scores)): # Operation: \*\***Traversal**\*\* (to search for the correct position)  
 if n > scores[index]:  
 continue  
 else:  
 scores.insert(index, n) # Operation: \*\***Insertion**\*\*  
 break

## Activity 3: Complexity Analysis

|  |  |  |
| --- | --- | --- |
| **Algorithm Description** | **Big O Classification** | **Reason** |
| 1. Accessing the 5th element of an array. | O(1) | Array access by index is a constant-time operation. The computer can jump directly to the memory location. |
| 2. Printing every element in a linked list. | O(n) | The algorithm must visit every one of the n nodes exactly once. Time scales linearly with input size. |
| 3. Using nested loops to find all duplicate values in a list. | O(n²) | For each of the n elements, the algorithm potentially checks all other n elements. n \* n = n². |
| 4. Finding a number in a sorted list using Binary Search. | O(log n) | Each step of the algorithm halves the search space. The number of steps needed grows logarithmically with n. |
| 5. Generating every possible subset of a set of items. | O(2ⁿ) | A set of n items has 2ⁿ possible subsets. The algorithm's output itself grows exponentially. |