**Week-1:**

Exercise:1 Inventory Management System

1. **Understand the Problem:**
   * Explain why data structures and algorithms are essential in handling inventories.
     1. Data Structures are important for retrieving and storing large amounts of data.
   * Discuss the types of data structures suitable for this problem.
     1. Data Structures like ArrayList and Hashmap are suitable for this problem.

**Time Complexity Analysis:**

* **Add Product:**
  + Using HashMap.put(), the average time complexity is O(1).
* **Update Product:**
  + Using HashMap.put(), the average time complexity is O(1). The put method will replace the value if the key already exists.
* **Delete Product:**
  + Using HashMap.remove(), the average time complexity is O(1).
* **Get Product:**
  + Using HashMap.get(), the average time complexity is O(1).

Exercise:2

**Step 1: Understand Asymptotic Notation**

**Big O Notation:**

Big O notation is a mathematical notation used to describe the upper bound of an algorithm's running time. It provides a high-level understanding of the algorithm's efficiency in terms of time (or space) complexity, focusing on how the runtime or space requirements grow with the size of the input.

* **O(1):** Constant time. The operation does not depend on the input size.
* **O(n):** Linear time. The runtime grows linearly with the input size.
* **O(log n):** Logarithmic time. The runtime grows logarithmically with the input size.
* **O(n^2):** Quadratic time. The runtime grows quadratically with the input size.

**Best, Average, and Worst-Case Scenarios for Search Operations:**

* **Best Case:** The scenario where the algorithm performs the minimum number of steps. For a search operation, the best case is when the element is found at the first position (O(1) for linear search).
* **Average Case:** The expected scenario where the algorithm performs an average number of steps, typically considering a uniform distribution of all possible inputs.
* **Worst Case:** The scenario where the algorithm performs the maximum number of steps. For search operations, this could mean searching through the entire dataset without finding the element (O(n) for linear search and O(log n) for binary search).

**Time Complexity Comparison:**

* **Linear Search:**
  + Best Case: O(1) (if the product is at the first position)
  + Average Case: O(n/2) => O(n)
  + Worst Case: O(n) (if the product is at the last position or not found)
* **Binary Search:**
  + Best Case: O(1) (if the product is at the middle position)
  + Average Case: O(log n)
  + Worst Case: O(log n) (if the product is not found)

Exercise:3

**Step 1: Understand Sorting Algorithms**

**Bubble Sort:**

* **Description:** A simple comparison-based algorithm where each pair of adjacent elements is compared, and the elements are swapped if they are in the wrong order. This process is repeated until the list is sorted.
* **Time Complexity:** O(n^2) in the worst and average case, O(n) in the best case (when the array is already sorted).
* **Space Complexity:** O(1) (in-place sorting).

**Insertion Sort:**

* **Description:** Builds the final sorted array one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
* **Time Complexity:** O(n^2) in the worst and average case, O(n) in the best case (when the array is already sorted).
* **Space Complexity:** O(1) (in-place sorting).

**Quick Sort:**

* **Description:** An efficient, comparison-based, divide-and-conquer algorithm. It picks an element as a pivot and partitions the array around the pivot. The process is recursively applied to the sub-arrays.
* **Time Complexity:** O(n log n) on average, O(n^2) in the worst case.
* **Space Complexity:** O(log n) due to the stack space used by recursion.

**Merge Sort:**

* **Description:** A divide-and-conquer algorithm that splits the array into halves, recursively sorts them, and then merges the sorted halves.
* **Time Complexity:** O(n log n) in all cases.
* **Space Complexity:** O(n) due to the temporary arrays used for merging.

**Time Complexity Comparison:**

* **Bubble Sort:**
  + Best Case: O(n) (when the array is already sorted)
  + Average Case: O(n^2)
  + Worst Case: O(n^2)
* **Quick Sort:**
  + Best Case: O(n log n)
  + Average Case: O(n log n)
  + Worst Case: O(n^2) (when the smallest or largest element is always chosen as the pivot)

**Why Quick Sort is Generally Preferred Over Bubble Sort:**

* **Efficiency:** Quick Sort is significantly faster on average compared to Bubble Sort, especially for large datasets.
* **Practical Performance:** Despite its worst-case time complexity of O(n^2), Quick Sort's average-case time complexity of O(n log n) and its ability to sort in-place make it highly efficient in practice.
* **Adaptability:** Quick Sort can be optimized with different pivot selection strategies to avoid the worst-case scenario, making it more versatile than Bubble Sort.

Exercise- 4:

**Step 1: Understand Array Representation**

**How Arrays are Represented in Memory:**

* **Memory Layout:** Arrays are stored in contiguous memory locations. This means that each element of the array is positioned next to its neighboring elements in memory.
* **Indexing:** Because of the contiguous allocation, arrays provide constant time O(1) access to elements using an index. This allows for fast retrieval.
* **Advantages:**
  + **Efficient Access:** Direct indexing provides quick access to any element.
  + **Predictable Iteration:** Iterating through an array is efficient and predictable due to the contiguous memory layout.
  + **Memory Efficiency:** Arrays typically have a lower memory overhead compared to linked lists since they don’t require storage for pointers.

**Time Complexity of Each Operation:**

* **Add Employee:**
  + **Best Case:** O(1) (when there is space available in the array).
  + **Worst Case:** O(1) (adding to the end of the array, no shifting needed).
* **Search Employee:**
  + **Best Case:** O(1) (if the employee is at the first position).
  + **Worst Case:** O(n) (if the employee is not found or at the last position).
* **Traverse Employees:**
  + **Best Case:** O(n).
  + **Worst Case:** O(n).
* **Delete Employee:**
  + **Best Case:** O(1) (if the employee is at the last position).
  + **Worst Case:** O(n) (if the employee is at the first position, requiring shifting all subsequent elements).

**Limitations of Arrays:**

* **Fixed Size:** Once an array is created, its size cannot be changed. This can lead to wasted space if the array is too large or overflow if the array is too small.
* **Insertion and Deletion:** While access is O(1), insertion and deletion (except at the end) require shifting elements, making them O(n) operations.
* **Memory Allocation:** Contiguous memory allocation can be an issue for large arrays, especially if memory is fragmented

Exercise-5

**Step 1: Understand Linked Lists**

**Singly Linked List:**

* **Description:** A linked list where each node contains data and a reference (or link) to the next node in the sequence.
* **Advantages:**
  + Dynamic size: Can grow and shrink in size without the need for reallocation or reorganization of its elements.
  + Ease of insertion/deletion: Adding or removing elements does not require shifting elements as in arrays.
* **Disadvantages:**
  + No direct access: Accessing an element requires traversal from the head, resulting in O(n) access time.

**Doubly Linked List:**

* **Description:** A linked list where each node contains data, a reference to the next node, and a reference to the previous node.
* **Advantages:**
  + Bidirectional traversal: Can be traversed in both directions.
  + Ease of deletion: A node can be deleted more easily as each node has a reference to its previous node.
* **Disadvantages:**
  + Extra memory: Requires more memory to store the additional reference to the previous node.

**Time Complexity of Each Operation:**

* **Add Task:**
  + **Best Case:** O(1) (adding to an empty list).
  + **Worst Case:** O(n) (traversing to the end of the list).
* **Search Task:**
  + **Best Case:** O(1) (if the task is at the head).
  + **Worst Case:** O(n) (if the task is at the end or not found).
* **Traverse Tasks:**
  + **Best Case:** O(n).
  + **Worst Case:** O(n).
* **Delete Task:**
  + **Best Case:** O(1) (if the task is at the head).
  + **Worst Case:** O(n) (if the task is at the end or not found).

**Advantages of Linked Lists Over Arrays for Dynamic Data:**

* **Dynamic Size:** Linked lists can easily grow and shrink in size by adding or removing nodes without the need for reallocation.
* **Ease of Insertion/Deletion:** Inserting or deleting elements in a linked list is more efficient as it does not require shifting elements.
* **Memory Efficiency:** For applications where frequent insertions and deletions occur, linked lists can be more memory efficient as they avoid the need to allocate contiguous memory blocks.

Exercise-7

**Concept of Recursion:**

* **Definition:** Recursion is a method of solving problems where a function calls itself as a subroutine. This allows the function to be repeated several times, as it can call itself during its execution.
* **Base Case:** The condition under which the recursion ends. It prevents infinite recursion.
* **Recursive Case:** The part of the function that calls itself with modified arguments, gradually approaching the base case.

**Advantages of Recursion:**

* Simplifies code for problems that have a natural recursive structure, such as tree traversals, factorial calculation, and the Fibonacci sequence.
* Makes code easier to understand and write for problems that can be broken down into smaller, similar subproblems.

**Disadvantages of Recursion:**

* Can lead to high memory usage due to the call stack.
* May be less efficient than iterative solutions if not properly optimized.

**Step 4: Analysis**

**Time Complexity of Recursive Algorithm:**

* The time complexity of the recursive algorithm is O(n), where n is the number of years. This is because the function calls itself once per year, resulting in a linear relationship between the number of recursive calls and the input size.