**NAME : TRISHILA SINGHA ROY**

**COLLEGE : UNIVERSITY OF ENGINEERING AND MANAGEMENT, KOLKATA**

**ENROLLMENT NO : 12021002002016**

**SUPERSET ID : 5016812**

**COGNIZANT DIGITAL NURTURE 3.0 PROGRAM**

**WEEK 1 ANSWERS**

**EXERCISE 1 : INVENTORY MANAGEMENT SYSTEM**

1. Data structures and algorithms help us to find efficient and calculative ways to store and retrieve data and also to perform operations on data such as inserting, updating and deleting and to solve specific problems as per our requirements. So, in case of large inventories we can store the data in the form of hashmaps and also use them accordingly to add,update or delete thereby making the whole process easier.

4. The Time Complexity Analysis for the inventory problem is :

1. addProduct() method :

Here we are Adding a product to the Hashmap named products

In the average case, the time complexity is O(1) O(1) O(1) because a hashmap provides constant time performance for the basic operations such as put, get,set, remove assuming the hash function disperses elements properly among the buckets.In the worst case, the time complexity can degrade to O(n )O(n) O(n), where n is the number of products in the map.

2 .updateProduct() method :

Here we are pdating the quantity and price of an existing product.The time complexity is O(1) O(1) O(1) in the average case, as it involves retrieving an element and updating the required fields, both of which are constant time operations in a hashmap.In the worst case , the time complexity can degrade to O(n) O(n) O(n).

3. deleteProduct() method :

This is used to remove a product from the hashmap.The time complexity is O(1) O(1) O(1) in the average case, as it involves removing an element, which is a constant time operation in a hashmap.In the worst case , the time complexity can degrade to O(n) O(n) O(n).

Optimization Discussion

1.Hash Function: We can use a good hash function to minimize collisions. Reducing hash collisions will help maintain the O(1) O(1) O(1) time complexity for addition, updation, and deletion operations.

2.Separate Chaining : We can use separate chaining for collision resolution.Separate chaining (using linked lists or balanced trees) is an efficient strategy to handle collisions efficiently. Hashmap in Java uses separate chaining with linked lists, which are converted to balanced trees when the size exceeds a threshold, maintaining O(1) O(1) O(1) time complexity even in the case of collisions.

**EXERCISE 2 : E-COMMERCE PLATFORM SEARCH FUNCTION**

1. Big O Notation is a mathematical notation in Java which represents an algorithm’s worst case complexity and defines the runtime required to execute an algorithm by identifying how the performance of our algorithm will change according to the size of our input.

Big O Notation gives us a standardized way to describe the time complexity and space complexity of an algorithm. It helps in analysing algorithms in Java by

1. Evaluating the performance thereby giving an idea of how efficient an algorithm is in terms of time and space.
2. It also helps in predictive analysis because it gives us an idea of how an algorithm will perform as the size of the input grows.
3. Finally, it helps in optimization as developers can understand very well which parts of the code are inefficient and they can easily replace those parts with better data structures and algorithms.
4. In terms of searching ,
5. Best Case Scenarios are those cases where the target element has been found at very first or too soon and as a result the algorithm runs exceptionally well with very less time and space.It is denoted as O(1) which means that the operation takes very less time.
6. Average Case Scenarios refer to the performance of the search operation for various inputs and it results according to the statistical distribution of data. It is denoted as O(f(n)) where f(n) is the function which depends on the statistical distribution of the data.
7. Worst Case Scenarios refer to the longest amount of time required either because the target element is located at the end of the data structure or not at all present. It is very inefficient and it is denoted as O(n) where n means the number of elements.
8. The time complexity of linear search algorithm is O(n) because here we search for our target element from the very beginning of our data structure until we find it. However, the best case scenario is O(1) and O(n) is the worst case scenario whereas the time complexity of binary search algorithm is O(log n) because here we keep on dividing the data structure into halves and compare our target element with the middle element eliminating half of the remaining elements each time. However, the best case scenario is O(1) and O(log n) is the worst case scenario. For our platform the most suitable search algorithm would be binary search algorithm because here searches are frequent and the data or elements do not change often and are sorted so searching the list in a faster way gets justified using binary search because the number of inputs would be high.

**EXERCISE 3 : SORTING CUSTOMER ORDERS**

1. Bubble sort is a simple comparison based algorithm and it works by repeatedly going through the list thereby comparing adjacent elements and swapping them if required to bring them into order. In worst case, time complexity is O(n^2) and it can be used for small datasets if simplicity is important.

Insertion sort works by traversing through a sequence of items and comparing adjacent elements and then inserting them into correct positions in the sequence. Worst case time complexity for this sorting is O(n^2) and it is efficient for small datasets.

Quick sort works on the strategy of divide and conquer where it selects a pivot element from the sequence and partitions other elements into two sub arrays according to whether they are less than or greater than the pivot. It is suitable for large datasets and it’s worst case time complexity is O(n^2).

Merge sort technique recursively splits the array into halves until each sub array contains a single element and then merging those sub arrays in a sorted way. Here, the worst case time complexity is O(n) and is mostly used for large datasets.

4. The best case time complexity for bubble sort is O(n) when the array is already sorted and the worst case time complexity is O(n^2) when array is sorted in reverse order. Bubble sort technique is good for small datasets or nearly sorted datasets. Whereas Quick Sort has best case time complexity of O(log n) when the pivot splits the array into two equal halves and the worst case time complexity is O(n^2) when the pivot is the smallest or the largest element itself and thereby causing unbalanced partitions. This method is mostly efficient for large datasets.

Quick Sort is mostly preferred over Bubble Sort because :

* The average time complexity of Quick Sort is O(n log n) than bubble sort which is O(n^2) thereby making it more efficient for large datasets.
* Quick Sort is comparatively better for practical world purposes where the dataset is large and problems are critical rather than bubble sort.
* Java’s in built functions like Arrays.sort() uses Quick Sort technique rather than bubble sort due to it’s efficiency and faster speed.
* Space complexity of Quick Sort is better than Bubble Sort which is adversely affected by the time complexity, thereby affecting memory management.

**EXERCISE 4 : EMPLOYEE MANAGEMENT SYSTEM**

1. The different ways in which arrays are represented in memory would be :

* Contiguos Memory Allocation : Array elements are placed in contiguous blocks of memory one after the other.

For example: If one array element is placed at index 1000 and it has size 2 bytes then rest of the elements would be placed at 1002,1004,… and so on.

* Index-based Access : Each array element can be accessed using it’s index. If the array is a 0 index array then starting position would be from 0 and continue till size-1 of the array.
* Fized Size : This means that the array is of fixed size and cannot be changed dynamically. So, the memory allocated for the array stays constant all throughout.

Advantages of such methods are:

1. Direct access are being given by the array to any element using it’s index so the time complexity becomes O(1) and makes all the operations very fast.

2. In case of limited memory resources, since the size of an array is fixed and known at the time of allocation so it allows for predictable memory usage.

3. Due to the contiguous nature, iteration over arrays become more efficient and faster.

4. Arrays are simple to declare and use thereby making them a fundamental data structure.

4. The time complexity of various array operations are as follows:

* Add : O(1) is the time complexity because if the array has extra capacity then adding one element at the end would be a constant time operation. But, for adding at the beginning or the middle the time complexity would be O(n) because it requires shifting all subsequent elements one position to the right.
* Search : O(n) is the time complexity for linear search because if the array is unsorted then it requires going through all the elements one by one but for binary search on sorted arrays the time complexity is O(log n) because it repeatedly divides the array into two halves.
* Traverse : O(n) is the time complexity for traversal because the array is traversed through all the elements one by one.
* Delete : To delete an element from the end time complexity is O(1) as removing the last element is a constant time operation but to delete an element from beginning or middle the time complexity is O(n) because it requires all the elements to be shifted one by one to the left.

Limitations of arrays are :

* Once an array is created, it’s size cannot be changed and can lead to wastage of memory if the array is very large or insufficient space if the array is too small.
* Adding or removing elements, especially in the middle of the array requires shifting elements which is inefficient
* Arrays require a contiguous block of memory which can be problematic for very large arrays.
* Arrays have limited built-in methods for common operations like insertion, deletion, and resizing.

Arrays can be used for static data, performance-critical applications, contiguous memory requirement, simple data structures, multidimensional data.

**EXERCISE 5 : TASK MANAGEMENT SYSTEM**

1. There are 3 types of linked lists :

* Singly Linked List : A singly linked list is a unidirectional linked list. So, you can only traverse it in one direction, i.e., from head node to tail node. It is a linear data structure where each element, called a node, contains two parts: data and a reference to the next node in the sequence. The first node is called the head, and the last node points to null, indicating the end of the list.
* Doubly Linked List : A doubly linked list of singly linked lists is a data structure that consists of a set of singly linked lists each of which is doubly linked. It is used to store data in a way that allows for fast insertion and deletion of elements. It is similar to a singly linked list, but each node contains an additional reference to the previous node. This allows traversal in both directions.
* Circular Linked List : A circular linked list is a type of data structure that uses linked list technology to store data in a linear, sequential fashion. Circular linked lists are typically implemented using a singly linked list data structure. This means that each node in the list is connected to the next node via a pointer. The last node in the list is then connected back to the first node, creating the ring-like structure.

4. The time complexity for a singly linked list for the various operations are as follows:

* Add : Time Complexity for adding a node at the head involves changing the head pointer, which is a constant-time operation so it is O(1) whereas adding a node at the tail involves traversing the list to find the last node and then updating its next pointer, which takes O(n) time in the worst case.
* Search : Since we traverse the Linked List to search for the element, the time complexity is O(n).
* Traversal : Traversing the list to print or process each node takes O(n) time, where n is the number of nodes in the list.
* Delete : Deleting a node involves finding the node (which takes O(n) time) and then adjusting the pointers. The overall time complexity is O(n).

Advantages of linked lists over arrays for dynamic data would be :

* A linked list is a dynamic data structure, meaning its size can change during the execution of a program. This is a significant advantage over arrays, which have a fixed size once they are created. If the array needs to grow beyond its initial size, a new, larger array must be allocated and the existing elements copied to the new array, which is an O(n) operation.
* In an array, inserting or deleting an element requires shifting all subsequent elements, which can be computationally expensive whereas in insertion and deletion operations can be performed efficiently at any position within a linked list. Adding or removing an element only involves changing a few pointers, which is an O(1) operation if the position is known.
* In linkedlists memory is allocated for each element individually, which means there is no wasted space. The memory overhead is only for the pointers used to link the nodes whereas with an array, if we overestimate the size you need, the unused spaces are wasted.
* Linked lists can easily be made into more complex data structures like stacks, queues, and hash tables. This is more difficult with arrays, as they lack the flexibility and dynamic nature of linked lists.

**EXERCISE 6 : LIBRARY MANAGEMENT SYSTEM**

1. The two types of searching methods are :

* Linear Search : Linear search, also known as sequential search, is the simplest search algorithm. It checks each element in the collection one by one until it finds the target element or reaches the end of the collection. Worst case time complexity is O(n) and best case time complexity is O(1) (if the target element is at the first position).
* Binary Search : This type of searching algorithm is used to find the position of a specific value contained in a sorted array. The binary search algorithm works on the principle of divide and conquer and it is considered the best searching algorithm because it's faster to run. **The best case** occurs when the target element is the middle element of the array. The number of comparisons, in this case, is 1. So, the time complexity is O(1). **The worst case** occurs when the target element is not in the list or it is away from the middle element. So, the time complexity will be O(log n).

4. Difference between binary search and linear search time complexities are :

* Worst case for linear search is O(n) and occurs when the target element is at the last position or not present in the array at all. The algorithm has to check each element one by one and worst case for binary search is O(log n) and occurs when the target element is not present, and the algorithm has to reduce the search interval to zero.
* Average case for linear search is O(n) and on average, the algorithm will need to check half of the elements in the array and average case for binary search is O(log n) and on average, the algorithm will need to divide the search interval approximately log(n) times.

Linear Search can be used for small datasets, the simplicity of linear search often makes it the better choice. The performance difference between O(n) and O(log n) is negligible for small n and for datasets that are not sorted and cannot be sorted due to constraints or because sorting would disrupt the order of insertion, linear search is appropriate whereas Binary Search can be used for large datasets, binary search offers a significant performance improvement due to its O(log n) time complexity. As the size of the dataset grows, the efficiency of binary search becomes more pronounced. Binary search requires the dataset to be sorted. If the dataset is already sorted or if sorting can be performed efficiently and maintained, binary search should be used.

Since linear search works on unsorted data, it is ideal for small datasets where sorting would be unnecessary overhead whereas the dataset must be sorted for binary search to be applicable. If the dataset is already sorted or can be maintained in sorted order, binary search is ideal.

**EXERCISE 7 : FINANCIAL FORECASTING**

1. Recursion is a programming technique where a function calls itself to solve smaller instances of the same problem. This approach can simplify the solution of complex problems by breaking them down into more manageable sub-problems. Recursion leverages the idea of solving a problem using solutions to smaller instances of the same problem until a base case is reached.

All recursive algorithms must have the following:

Base Case (i.e. when to stop)

Work toward Base Case

Recursive Call

The main advantage of using recursion in the context of data structures is that it allows the algorithm to closely mirror the structural properties of the data, leading to more elegant, concise, and often more efficient solutions.

4. In Financial Forecasting java code since each recursive call involves a constant amount of work beyond the recursive call (a single multiplication and return statement), the time complexity is directly proportional to the number of recursive calls.Thus, the time complexity of the recursive algorithm is O(n)O(n)O(n), where nnn is the number of periods.

Ways to optimize the recursive solution to avoid excessive computation :

* Tail Recursion Optimization : **It** is a form of recursion where the recursive call is the last operation in the function. Some compilers or interpreters can optimize tail-recursive functions to avoid adding new frames to the call stack, thus reducing space complexity.
* Dynamic Programming : **It** is a technique that solves complex problems by breaking them down into simpler subproblems and storing the results of these subproblems to avoid redundant computations. It often involves either memoization (top-down approach) or tabulation (bottom-up approach).
* Iterative Solution : Converting a recursive solution to an iterative one can be more efficient. Iterative solutions avoid the overhead of recursive function calls and stack management.