MWEBEMBEZI NICOLE MBABAZI M23B23

1. **Describe your own real-world example that requires sorting. Describe one that requires finding the shortest distance between two points?**

- Sorting example

My real-word example that requires sorting is sorting a list of books alphabetically by their titles. Each book is represented as a dictionary with 'title' and 'author' keys. The key argument in sorted () is set to a lambda function that extracts the 'title' key from each dictionary, instructing the function to compare book titles for sorting. The sorted list is then printed, displaying the books in alphabetical order based on their titles.

- Shortest Distance

My real-world example is finding the shortest route between two buildings. To find the shortest possible distance between the two buildings. This distance can be determined by measuring the straight-line distance between the points on each building by walking. This can be done by calculating the distance in km .

2. **Other than speed, what other measures of efficiency might you need to consider in a real-world setting?**

In real-world settings, efficiency often goes beyond speed. Here are some other important measures to consider:

- Space complexity: The amount of memory an algorithm requires. This is particularly important when dealing with large datasets or limited hardware resources.
- Energy consumption: For devices with limited battery life or energy efficiency constraints, the energy consumed by an algorithm can be a critical factor.
- Real-time performance: In applications where immediate responses are required, algorithms must meet strict real-time constraints.
- Scalability: The ability of an algorithm to handle increasing input sizes efficiently. This is crucial for systems that need to accommodate growth.
- Correctness: The algorithm must produce accurate results. In some cases, even a slightly incorrect result can have significant consequences.

3. **Select a data structure that you have seen, and discuss its strengths and limitations.**

Linked List

A linked list is a linear data structure where elements are connected by pointers. It offers flexibility in terms of insertions and deletions, as elements can be added or removed without shifting the entire list.

Strengths of Linked Lists:

- **Dynamic memory allocation:** Linked lists do not require a fixed size at creation, allowing for efficient insertion and deletion of elements without needing to reallocate memory.
- **Flexible structure:** Linked lists can represent various data structures, such as stacks, queues, and graphs.
- **No need for contiguous memory:** Unlike arrays, linked lists do not require contiguous memory locations, making them more adaptable to memory constraints.

Limitations of Linked Lists:

- **Slower random access:** Accessing a specific element in a linked list requires traversing the list from the beginning, resulting in linear time complexity (O(n)). This can be inefficient compared to arrays, which provide constant-time random access.
- **Additional space overhead:** Each node in a linked list requires extra space to store a pointer to the next node, which can be a disadvantage in memory-constrained environments.

4. How are the shortest-path and traveling-salesperson problems given above similar? How are they different?

The shortest-path and traveling-salesperson problems are both optimization problems that involve finding the best route between nodes in a graph. However, they differ in their specific goals and constraints:

Similarities:

- Both problems operate on graphs, where nodes represent locations and edges represent connections between them.
- Both problems consider the distance or cost associated with each edge, which can be a measure of distance, time, or other relevant factors.
- Both problems aim to find the optimal route, either the shortest path or the shortest tour that visits all nodes exactly once.

Differences:

- Shortest-path problem is solved by finding the shortest path between two specific nodes in the graph. while traveling-salesperson finds the shortest tour that visits all nodes exactly once and returns to the starting node.

- In Shortest-path there’s no specific constraint on the number of times nodes can be visited while Traveling-salesperson each node must be visited exactly once, and the tour must return to the starting node.

- Shortest-path can be solved efficiently using algorithms like Dijkstra's algorithm or the Bellman-Ford algorithm while the Traveling Salesperson Problem is significantly more complex because it is an NP-hard problem hence there is no known algorithm to solve it in polynomial time for all cases.

5.Describe a real-world problem in which only the best solution will do. Then come up with one in which “approximately” the best solution is good enough.

Optimal Solution:

A delivery driver needs to deliver packages to multiple locations. The driver wants to find the shortest possible route to minimize travel time and fuel consumption. In this case, an optimal solution is crucial to ensure efficiency and reduce costs.

Approximate Solution:

While an optimal solution is ideal, an approximate solution might be acceptable in certain scenarios. For example, if the delivery driver is facing tight deadlines or unexpected traffic conditions, finding a route that is *close* to optimal might be sufficient. As long as the driver can complete deliveries within a reasonable timeframe, an approximate solution can be considered good enough.

6.Describe a real-world problem in which sometimes the entire input is available before you need to solve the problem, but other times the input is not entirely available in advance and arrives over time.

Traffic Navigation App:

Imagine using a traffic navigation app to plan your route to work.

- Scenario 1: **Planned Route**: If you have a fixed daily commute, you can plan your route in advance, knowing the typical traffic conditions and potential road closures. In this case, you have the entire input (map data, traffic patterns) available beforehand.
- Scenario 2: **Unexpected Events**: However, unexpected events like accidents, road construction, or severe weather can significantly impact traffic flow. In these situations, the app needs to update its data in real-time to provide accurate route suggestions. This demonstrates how the input (traffic conditions) can arrive over time and influence the solution (optimal route).