## **Recipe Manager Application (DSA Mini Project Report)**

### **Introduction**

The **Recipe Manager Application (RPM)** is a comprehensive project developed to manage recipes for meal planning, providing users with an intuitive interface for recipe sorting, searching, viewing, and managing. It integrates **Data Structures** such as **Binary Search Tree (BST)**, **Stack**, **Circular Queue**, and **Linked List** to implement various features like recipe searching, storing favorites, managing recent views, and meal planning.

The objective of the project was to demonstrate how data structures can be applied to efficiently manage and sort data in a real-world scenario, in this case, recipes.

### **Data Structures Used**

This Recipe Manager Application integrates several data structures to store and manipulate recipes effectively. The key data structures used are:

1. **Binary Search Tree (BST)** for storing recipes in an ordered manner.
2. **Stack** to track recently viewed recipes.
3. **Circular Queue** for meal planning with a fixed size.
4. **Linked List** to store favorite recipes dynamically.

### **1. Binary Search Tree (BST)**

#### **Purpose of BST:**

A **Binary Search Tree (BST)** is an ordered tree data structure in which each node contains a key and two subtrees. The left subtree contains nodes with keys smaller than the node’s key, while the right subtree contains nodes with keys greater than the node’s key. In this project, the BST is used to store and manage recipes by their names in sorted order.

#### **Why BST?**

The BST structure is ideal for this project because it allows efficient searching, insertion, and traversal of recipes based on their names. By storing recipes in sorted order, we can quickly find, add, and display them with time complexities that are efficient compared to unsorted lists or arrays.

#### **Key Operations:**

* **Insert**: Recipes are inserted into the BST by comparing their names. Each new recipe is placed in the correct position, maintaining the sorted order of the tree.
* **Search**: Searching for a recipe by name is efficient with BST, as we can skip large portions of the tree based on lexicographic ordering, yielding O(log n) time complexity in balanced trees.
* **Display**: To display all recipes, we perform an **in-order traversal** of the BST, which visits nodes in ascending order of their names.

#### **Time Complexity:**

* **Insert**: O(log n) for balanced trees.
* **Search**: O(log n) for balanced trees.
* **In-order Traversal (Display)**: O(n), as we visit every node once.

#### **Example:**

* If we insert recipes such as **"Pizza"**, **"Burger"**, and **"Pasta"** into the BST, they will be placed in alphabetical order, allowing for efficient retrieval and display.

### **2. Stack**

#### **Purpose of Stack:**

A **Stack** is a **Last-In, First-Out (LIFO)** data structure where the most recently added element is the first to be removed. In the Recipe Manager Application, the stack is used to store **recently viewed recipes**, allowing users to easily revisit recipes they have recently searched or viewed.

#### **Why Stack?**

The stack is an ideal data structure for tracking recent actions because it aligns perfectly with the requirement of displaying the most recently accessed recipes first. The stack provides an efficient way to implement the feature where users can "go back" to the last viewed recipe.

#### **Key Operations:**

* **Push**: Adds a recipe to the top of the stack whenever it is searched or viewed.
* **Pop**: Removes the most recently viewed recipe from the stack.
* **Display**: Displays all recipes in the stack, starting from the most recent, in LIFO order.

#### **Time Complexity:**

* **Push**: O(1), constant time operation.
* **Pop**: O(1), constant time operation.
* **Display**: O(n), where n is the number of recipes in the stack.

#### **Example:**

* If a user searches for the recipe **"Pasta"**, it is added to the stack. If they later search for **"Pizza"**, it replaces **"Pasta"** at the top of the stack. The user can quickly "pop" the stack to go back to **"Pasta"**.

### **3. Circular Queue**

#### **Purpose of Circular Queue:**

A **Circular Queue** is a type of **Queue** where the last element is connected to the first element, forming a circle. It is used to manage **meal planning recipes**, where a fixed-size list of recipes needs to be maintained. Once the queue reaches its capacity, new recipes replace the oldest ones, ensuring the queue size remains constant.

#### **Why Circular Queue?**

The circular queue is well-suited for meal planning because we want to store a fixed number of recipes for future use. When the queue is full, adding a new recipe should automatically replace the oldest recipe, maintaining an efficient rotation.

#### **Key Operations:**

* **Enqueue**: Adds a recipe to the queue. If the queue is full, it overwrites the oldest recipe.
* **Dequeue**: Removes the oldest recipe from the front of the queue.
* **Display**: Displays all recipes in the queue, starting from the front and moving to the rear in a circular manner.

#### **Time Complexity:**

* **Enqueue**: O(1), constant time operation.
* **Dequeue**: O(1), constant time operation.
* **Display**: O(n), where n is the number of recipes in the queue.

#### **Example:**

* Suppose the meal planning queue has a capacity of 5. If the queue contains 5 recipes and a 6th recipe is added, the recipe at the front of the queue is removed, and the new recipe takes its place, ensuring that only the latest 5 recipes are stored.

### **4. Linked List**

#### **Purpose of Linked List:**

A **Linked List** is a linear data structure where each element (node) contains data and a reference (or link) to the next node. In the Recipe Manager Application, the linked list is used to store **favorite recipes** dynamically. Each time a user adds a recipe to their favorites, the recipe is inserted into the linked list.

#### **Why Linked List?**

The linked list is a flexible data structure because it allows for easy insertion and deletion of nodes. It does not require resizing or shifting elements, making it ideal for storing a growing list of favorite recipes, where users can continually add recipes without worrying about array resizing.

#### **Key Operations:**

* **Add to Favorites**: A recipe is added to the linked list, either at the head or the tail of the list.
* **Display Favorites**: All recipes in the favorites list are displayed in the order they were added.

#### **Time Complexity:**

* **Add to Favorites**: O(1), assuming we add at the head or tail.
* **Display Favorites**: O(n), where n is the number of recipes in the linked list.

#### **Example:**

* If a user adds the recipe **"Burger"** to their favorites list, it will be inserted as a new node in the linked list. If they later add **"Pizza"**, it will be appended as the next node.

### **Algorithmic Operations in the Application**

The Recipe Manager Application incorporates several key features that rely on the data structures discussed above:

1. **Add Recipe**: Recipes are added to both the BST (for efficient searching) and the list of recipes.
2. **Display All Recipes**: Recipes stored in the BST are displayed in alphabetical order via an **in-order traversal**.
3. **Search Recipe by Name**: Users can search for a recipe by name in the BST, which allows for efficient retrieval of recipes.
4. **Add Recipe to Meal Planning Queue**: A recipe is added to the Circular Queue for meal planning, with automatic overwriting of the oldest recipe when the queue is full.
5. **Display Meal Planning Queue**: The Circular Queue is displayed to show the current list of planned meals.
6. **Display Recently Viewed Recipes**: The most recently viewed recipes are displayed from the Stack in LIFO order.
7. **Sort Recipes**: Users can sort recipes using various algorithms (Bubble Sort, Insertion Sort, Quick Sort) based on ingredients, popularity, or cuisine.
8. **Add Recipe to Favorites**: Recipes are added to a linked list, maintaining the order in which they were added.
9. **Display Favorites**: Displays all recipes in the favorites list.

### 

### 

### 

### **Conclusion**

The Recipe Manager Application showcases how core data structures such as **Binary Search Trees (BST)**, **Stacks**, **Circular Queues**, and **Linked Lists** can be applied to manage and organize recipes effectively. Each data structure is chosen based on its strengths for specific tasks, ensuring that the application is both efficient and user-friendly. By leveraging these structures, the application is able to perform a variety of operations, such as efficient searching, sorting, and meal planning, that would otherwise be slow or cumbersome with simpler data structures.

The use of these data structures not only improves the application's performance but also demonstrates practical applications of **Data Structures and Algorithms (DSA)** in solving real-world problems. The system is highly scalable and can easily be extended to accommodate additional features like user reviews, ratings, or recipe sharing.