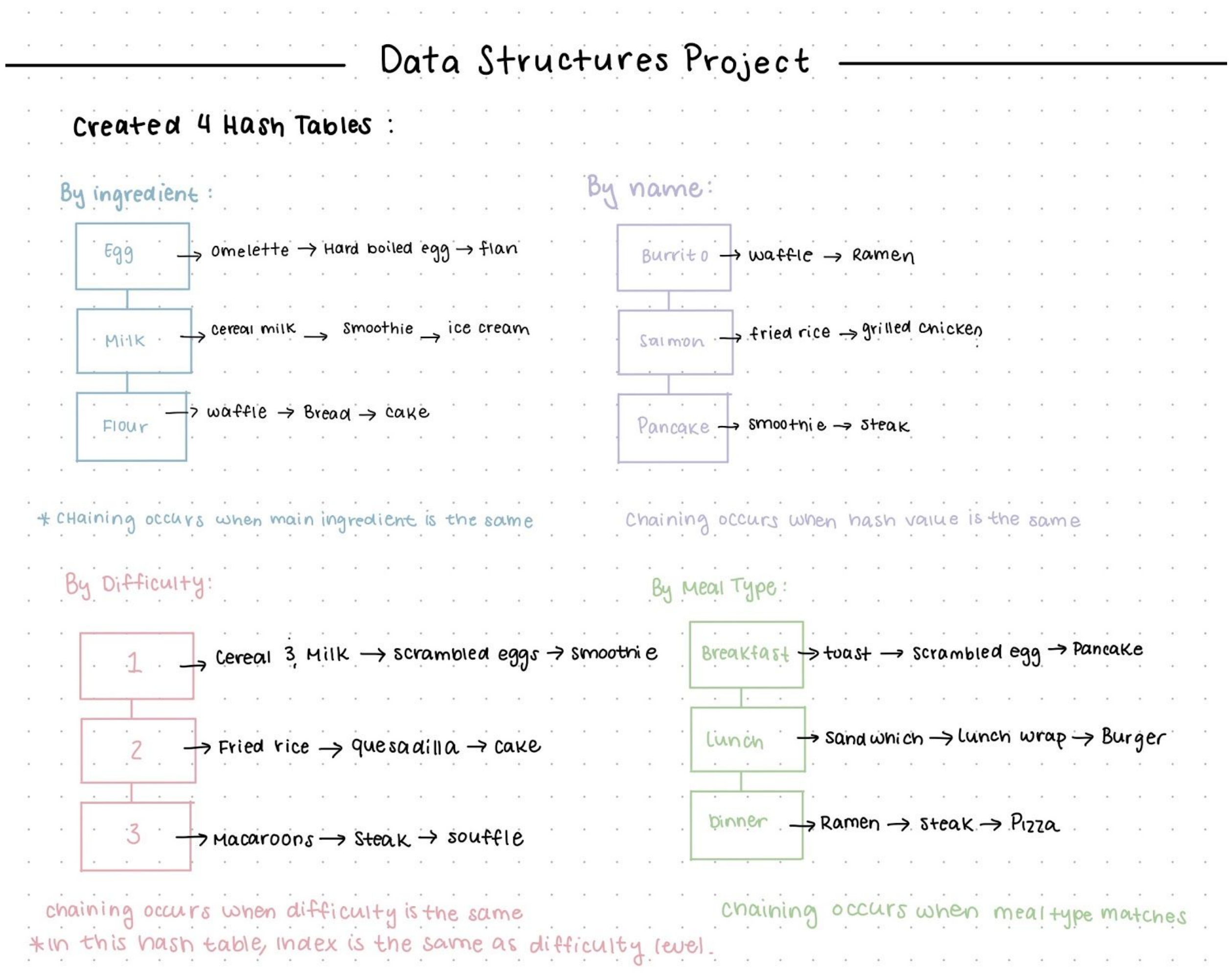
The B(e)ST Meal Prep Project Report- Vienna Wong

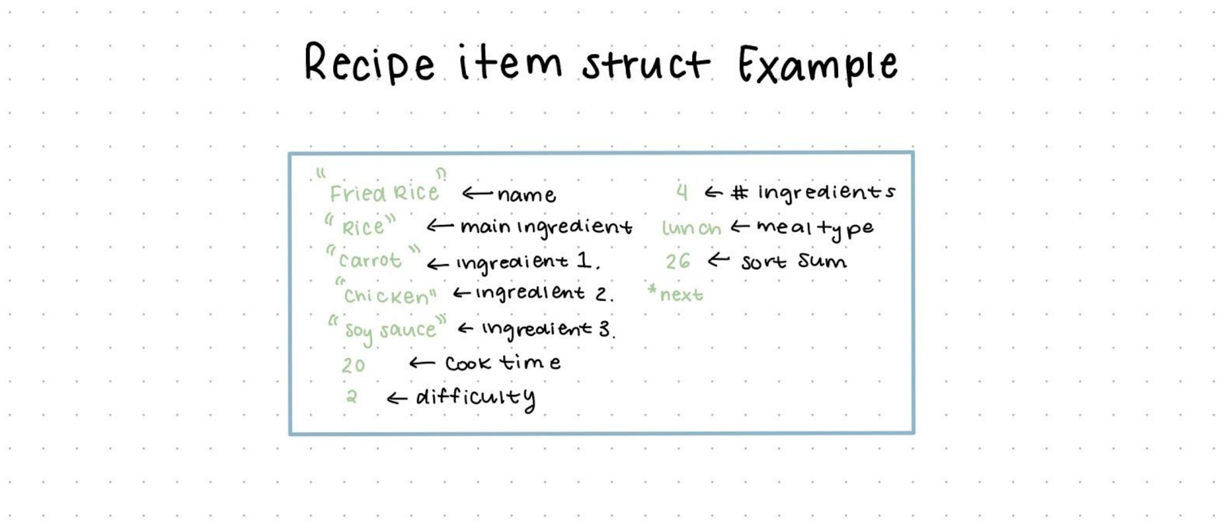
The data structures used to implement this project are structs and hash tables. The recipes will be stored in different **hash tables** to sort the recipes. We will implement this by using four hash tables used by sorting/ filtering by: (1) name, (2), main ingredient, (3) difficulty level, and (4) meal type.


Data Structures Project
Created 4 Hash Tables :
By ingredient :
Egg → omelette → Hard boiled egg → flan
Milk → cereal milk → smoothie → ice cream
Flour → waffle → Bread → cake
* chaining occurs when main ingredient is the same
By name:
Burrito → waffle → Ramen
Salmon → fried rice → grilled chicken
Pancake → smoothie → steak
Chaining occurs when hash value is the same
By Difficulty:
1 → Cereal 3 Milk → scrambled eggs → smoothie
2 → Fried rice → quesadilla → cake
3 → Macaroons → Steak → souffle
chaining occurs when difficulty is the same
*in this hash table, index is the same as difficulty level.
By Meal Type:
Breakfast → toast → scrambled egg → Pancake
Lunch → sandwhich → lunch wrap → Burger
Dinner → Ramen → steak → Pizza
chaining occurs when meal type matches

We implemented our project with four hash tables because we found that it would be the most straightforward way to filter and organize our data. Since each recipe held such a large amount of information and we were sorting/filtering the data in a large variety of ways, we decided that creating a different hash table for each filtering method would be the best way to keep organized and access information about each recipe quickly. The user has the option to find a recipe by the name, what ingredients they have on hand, the difficulty level of the recipe, or the type of meal they are looking to create.

Rather than looking through each recipe item for a recipe's particular information, we can just look into a particular hash table. For example, if the user is only looking for breakfast items, we can print the linked list of nodes that are stored at the hash index for "breakfast". We also decided to use chaining, rather than linear probing because it could account for an unlimited capacity, unlike linear probing. With the understanding that the user can choose to add their own recipes into the index, we want to ensure that the hash table doesn't fill up.

We used **structs** to create recipe items that will hold the recipe's name, ingredient(s), number of ingredients, difficulty level, meal type and cook time/ Since the recipe index is created to help the convenience of college students, we wanted to display the recipes that would take the easiest recipe that took the shortest time and the fewest number of ingredients. To do this, we displayed the recipes with the lowest sum of ingredients, cook time, and difficulty.



We have decided to use hash tables because it was an efficient way to access, store, and search for information. The time complexity to search and insert in a hash table is better than using arrays, stacks and queues, or linked lists, as Hash Tables (on average) run on 0(1) time.