NPS2001C
Milestone 4

Chong Zhao Yang, Shun Pyae Phyo,
Ho Jia Wen, Jane, Han Xin Ping

1. The purpose of the sorting algorithm is to sort the food item and store based on the expected waiting time while the purpose of the searching algorithm is to search the food item and store. We have chosen **quicksort as the sorting algorithm** and **linear search as the searching algorithm**.

Quicksort is a well-known sorting algorithm known for its average performance. It's efficient for large datasets and has a time complexity of O(n log(n)) in the average case. Other than Quicksort, there are other sorting algorithms available which are miracle sort, bubble sort, and selection sort. Miracle sort is not efficient or optimal because it relies on chance to achieve the sorted state, making it incredibly slow and unreliable. Bubble sort, while easy to understand, is inefficient for large datasets, reflecting in its time complexity of $O(n^2)$. Selection sort is a simpler option to implement, but its $O(n^2)$ time complexity makes it inefficient for large datasets of food stores. Therefore, due to the large data expected in the use of a food ordering service, Quicksort would be the most efficient and ideal.

Other than linear search, binary search is also available. Binary search was not chosen despite its lower time complexity of $O(\log_2 n)$ as the dataset to be searched is not sorted. As such, binary search cannot be implemented to search the unsorted dataset. Therefore, linear search, which can search within an unsorted dataset, was chosen instead as the search algorithm.

2. Flowcharts are added in the appendix for visual representation of function of algorithm..

The input of the algorithm would be pre-defined store of each food item (I.e. Noodle store and Chicken rice store), the list of food items, list of pictures of food items, list of prices of food items, list of cooking time for the food items, and an empty list of waiting time for the food items.

The first section of the algorithm aims to generate a consistently updated list of food waiting times based on the cooking time and waiting time of each store. (In this context, the waiting time for the noodle store was defined to be 10 minutes and the waiting time for the chicken rice store was defined to be 8 minutes.) For each food item in the list containing all the food items, depending on whether they were in the noodle store or chicken rice store, the cooking time was added to the store waiting time to produce an overall waiting time for the food item. This process was repeated for each food item. The overall waiting time for each food item is then added to the empty list of waiting time for food items.

After the overall waiting time for each food item was added to the list, all the relevant lists were converted into arrays using the “np.array” function. All of the arrays were then stacked onto each other to form a final array containing overall waiting time, name of food item, picture of food item, and price of food item.

The final array is then sorted using Quicksort, based on ascending waiting time (I.e. from shortest waiting time to longest waiting time). This sorted array is then printed to display to the user which food item has the shortest waiting time, along with the name, picture, and price of the food items.

If the user wants to search for a specific food item, linear search is used to search the column within the final array containing the name of the food item. The row containing the name of the food item searched is then printed to display the waiting time, picture, and price of the food item searched.

3. What is (are) the limitation(s) of this algorithm(s)?

The limitations to Quicksort are as follows:

a. Worst-case scenario: Quicksort's performance heavily depends on the choice of the pivot element. In the worst-case scenario, where the pivot is consistently chosen poorly (I.e. always the smallest or largest element), particularly when the input array is already sorted or nearly sorted, the algorithm can degrade to quadratic time complexity of $O(n^2)$.
b. Unstable sorting: Quicksort is an unstable sorting algorithm, meaning it does not preserve the relative order of equal elements. If there are duplicate elements in the input array, their order relative to each other might change after sorting. Linking this to our app, due to the constant updating of waiting time, there can be multiple stalls with the same waiting time. This can cause confusion, if the app keeps changing the order of stalls despite similar waiting times.

The limitation to Linear Search is as follows:

a. Linear search has a time complexity of O(n), where n is the number of elements in the list. This means that the time taken to search for an element increases linearly with the size of the list. For large datasets, linear search can be inefficient compared to other search algorithms with lower time complexities like binary search or hash tables. Possible mitigation: to filter the list by venue, this can reduce the size of the array to be filtered thus increasing efficiency of the search function.

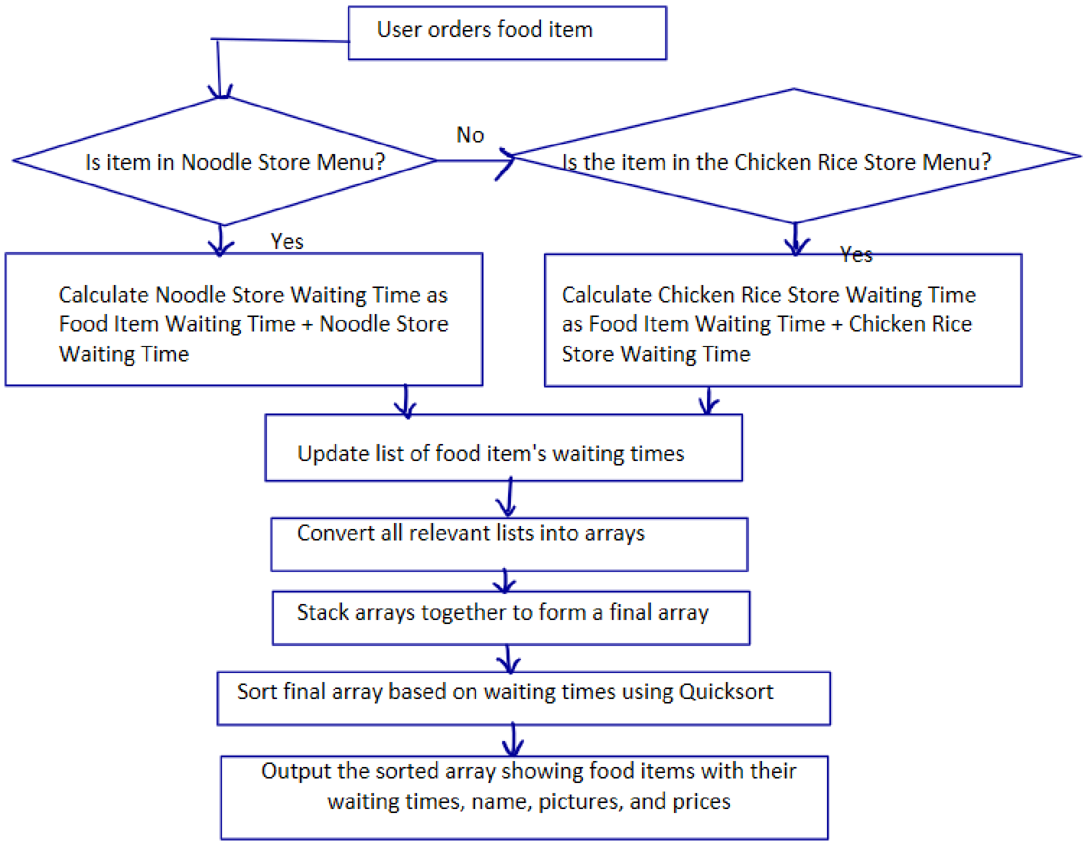
4. Provide an executable code that demonstrates the work of algorithm(s) and tests which you run.

References:
Numpy (2022).Numpy.column_stack.
https://numpy.org/doc/stable/reference/generated/numpy.column_stack.html
Numpy (2022). Numpy: the absolute basics for beginners - NumPy v1.26 Manual.
https://numpy.org/doc/stable/user/absolute_beginners.html

Annex

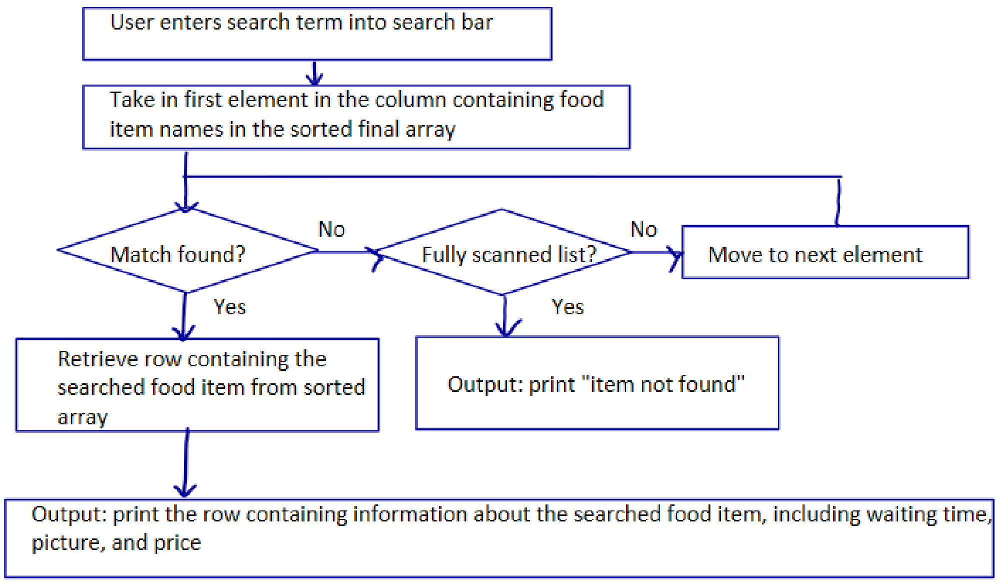
Quicksort

Input: list of food items, their respective cooking times, and store menus


User orders food item
Is item in Noodle Store Menu?
No
Is the item in the Chicken Rice Store Menu?
Yes
Calculate Noodle Store Waiting Time as Food Item Waiting Time + Noodle Store Waiting Time
Yes
Calculate Chicken Rice Store Waiting Time as Food Item Waiting Time + Chicken Rice Store Waiting Time
Update list of food item's waiting times
Convert all relevant lists into arrays
Stack arrays together to form a final array
Sort final array based on waiting times using Quicksort
Output the sorted array showing food items with their waiting times, name, pictures, and prices

Linear search

Input: sorted final array, food item to search for


User enters search term into search bar
Take in first element in the column containing food item names in the sorted final array
Match found?
No
Fully scanned list?
No
Move to next element
Yes
Retrieve row containing the searched food item from sorted array
Yes
Output: print "item not found"
Output: print the row containing information about the searched food item, including waiting time, picture, and price