gsi.py:

- This is the Main python file which is already provided and contains main and test procedure which calls methods implemented on "simp_arr_gsi.py" or "hash_gsi.py" file to manage inventory items

simp_arr_gsi.py

- This is a file that includes the class of a simple array-based inventory manager. This class contains such methods as init, insert_item, print_items, find_item. Please keep in mind the following notes for each method during implementation:
    - Init(): initializes two simple arrays or simple 2D array to be used throughout object life. One array might be used for storing items while another array is used for storing price with synced index.
    - insert_item(item, price): if the item is **already in** the list, this method **updates** the price of it. If the item is not in the list, inserts the item and price at the **front** of each or 2D array. **Parameters:** item name and price. **Note:** you are allowed to use the **insert()** method from Python Array Module.
    - print_items(): prints item in pair of item name and price. **Ex. [(item_name, item_price),...]**
    - find_item(item): returns the price of a given item. If the item is not found, simply use "returns" statement. **Parameters:** item name to be looked up.

hash_gsi.py

- This is a file that includes the class of hash tables based inventory manager. This class contains such methods as init, insert_item, print_items, find_item. In addition, this class requires the inner class to hold onto hash tables data as a linked list. Please keep in mind the following notes for each method during implementation:
    - Example illustration: hash tables based inventory structure should look like the diagram below, where each array slot is used to contain a linked list. Each item and price is stored as objects in the linked list. Each node holds onto the item name and price.
    - Init(tableSize): initializes array 'tableSize' size array with None object to be used to store a linked list. **Parameters:** table size. **Note:** think this as the load factor.
    - hash_func(key): uses item name as a key to compute the hash and returns the result. For this exercise purpose, we will simply sum up each character's ASCII decimals from the item name to compute the hash value. Ex. apple: a (97) + p (112) + p (112) + l (108) + e (101) = 530 **Parameters:** item name.

- insert_item(item, price): if the item is **already in** the list, this method **updates** the price of it. If the item is not in the list, inserts the item at the **end** of the linked list on the corresponding array slot.
    - **Parameters:** item name and price
    - **Pseudocode:**
        - Compute hash_key(hash_func_result % table size) to identify slot.
        - Check if the given slot is None object. If None, start a linked list with given parameters.
        - Check if the item is at the hash_key slot. If the item is found, update the price.
        - If the item is not found, create a new node and attach it at the end of the linked list.
    - print_items(): prints items throughout the hash tables with a linked list. Note: try to print as **[(item1, price1) ...]** \n by using some combinations of "print(item, end = " ")" at each hash tables slot as shown below.
    - find_item(key): returns the price of a given item. If the item is not found, simply use the "returns" statement. Try to **reuse** the **hash_func** method. **Parameters:** item name to be looked up.

1. Compare the actual runtime of hash tables operation between two inventory structures and justify in a short paragraph on how it performs.

Notes

In the given output, we can see a comparison between the Simple Array-based and Hash Table-based inventory manager implementations. Both the Simple Array-based and Hash Table-based implementations store the same set of items and prices. However, the insertion and lookup time for each implementation is different.

For the Simple Array-based inventory manager:

- Insert time: 2.6200010324828327e-05 seconds
- Lookup time: 8.699993486516178e-06 seconds

For the Hash Table-based inventory manager:

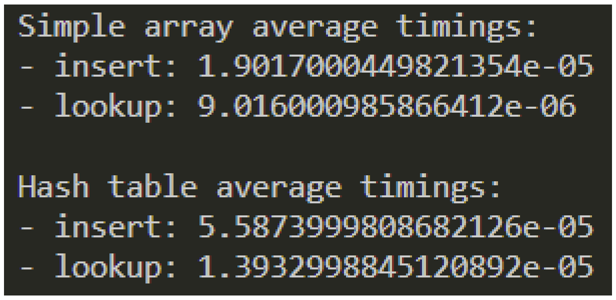
- Insert time: 5.8299992815591395e-05 seconds
- Lookup time: 8.400005754083395e-06 seconds

From the output, we can observe that the Hash Table-based inventory manager has a slightly faster lookup time compared to the Simple Array-based inventory manager. However, the insert time for the Hash Table-based inventory manager is slightly slower.

In general, hash tables offer faster average-case performance for insertions and lookups compared to simple arrays. Hash tables have an average-case time complexity of O(1) for both insertions and lookups, while simple arrays have an average-case time complexity of O(n) for lookups and O(1) for insertions at the end.

It is important to note that the performance improvement provided by hash tables comes at the cost of increased memory usage due to the use of an array with a specified load factor, as well as the possibility of collisions in the hash table. Collisions can be resolved using a linked list, as demonstrated in the script.

In conclusion, the output showcases the time difference between the two data structures. Hash tables are a better choice when faster lookups are a priority, while simple arrays can be more appropriate when memory usage is a concern.



```
Simple array average timings:
- insert: 1.9017000449821354e-05
- lookup: 9.016000985866412e-06

Hash table average timings:
- insert: 5.5873999808682126e-05
- lookup: 1.3932998845120892e-05
```