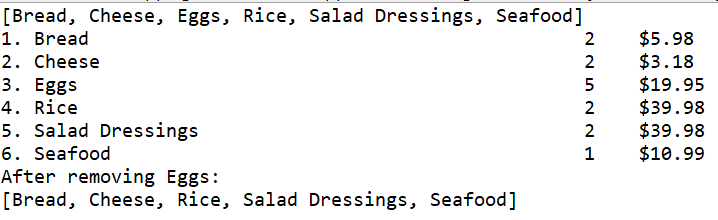
**1: Shopping List**

*Main Output*



*Analysis of Method insert*

Consider first a shopping list with *N* items sorted by name. Let a unit of complexity be an access of an item reference.

If an input item to the insert method is null, the method performs one comparison to determine this, and returns. The complexity of the insert method in this case is O(1).

If the internal array of the shopping list is full, the internal array is doubled in capacity. This operation has a complexity of O(*N*).

During each insertion, indexOf(Item item) is called. indexOf compares each item in the shopping list’s internal array with the item to be inserted, looking for equal names. In the best case, the item to be inserted has the same name as the first item in the shopping list, and the complexity of indexOf is O(1). In the worst case, the item to be inserted either has the same name as the last item in the shopping list, or does not have the same name as any item in the shopping list, and the complexity of indexOf is O(*N*).

An integer comparison determines whether the item was found in the shopping list. If so, an assignment, two accesses, a summation, an integer comparison, and a set are made, and the insert method returns. If the item was found in the shopping list, the worst-case complexity of insert is O(*N*).

If the comparison immediately above returns false, the item to be inserted is compared to every item in the shopping list until either items are exhausted or the name of the item to be inserted is found to be less than the name of some item in the shopping list, and an insertion index is found. In the best case, this “paragraph” is O(1), and in the worst case, O(*N*).

Shifts of references have a complexity of O(*N*) if the immediately above paragraph had a complexity of O(1), and O(1) if the paragraph had a complexity of O(*N*).

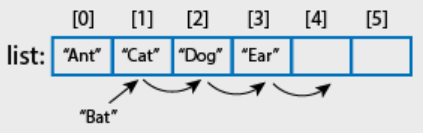
There is one additional item-reference assignment.

Overall, the insert method has worst-case complexity O(*N*).

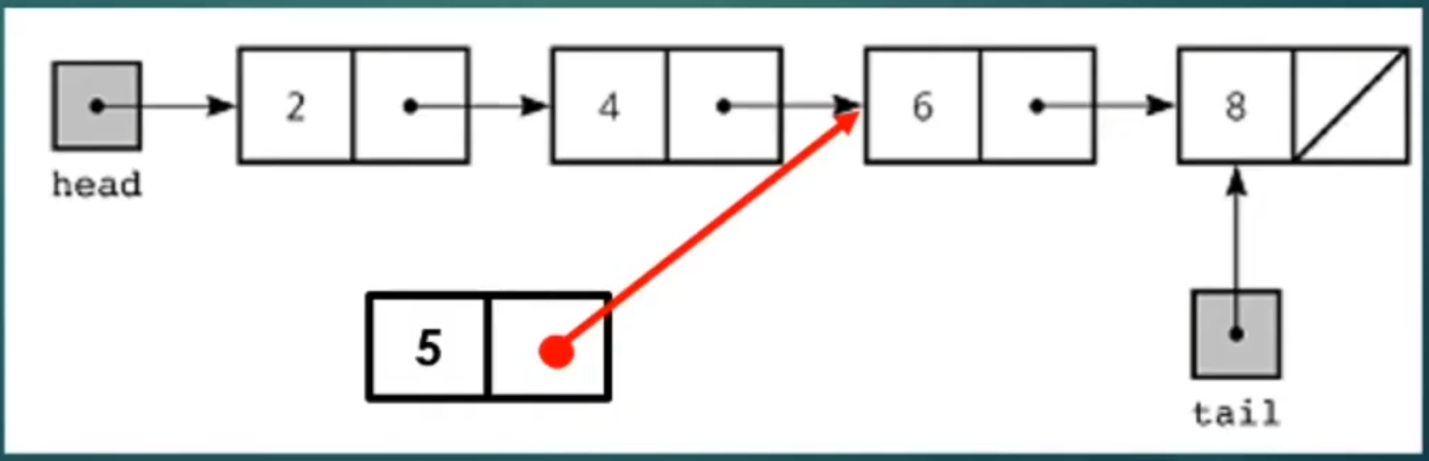
Considering a simplified algorithm, we want to insert an item with name “Bat” into a shopping list containing items with the following names.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **0** | **1** | **2** | **3** | **4** | **5** |
| “Ant” | “Cat” | “Dog” | “Ear” |  |  |

In the worst case, we want to create a reference to a new array, copy *N* items to the new array, initialize *N* items to null, set the internal array of the shopping list to be the new array, shift *N* items to the right, and add an item. The complexity of such an insert is O(*N*). Shifting and adding are depicted in the following diagram.

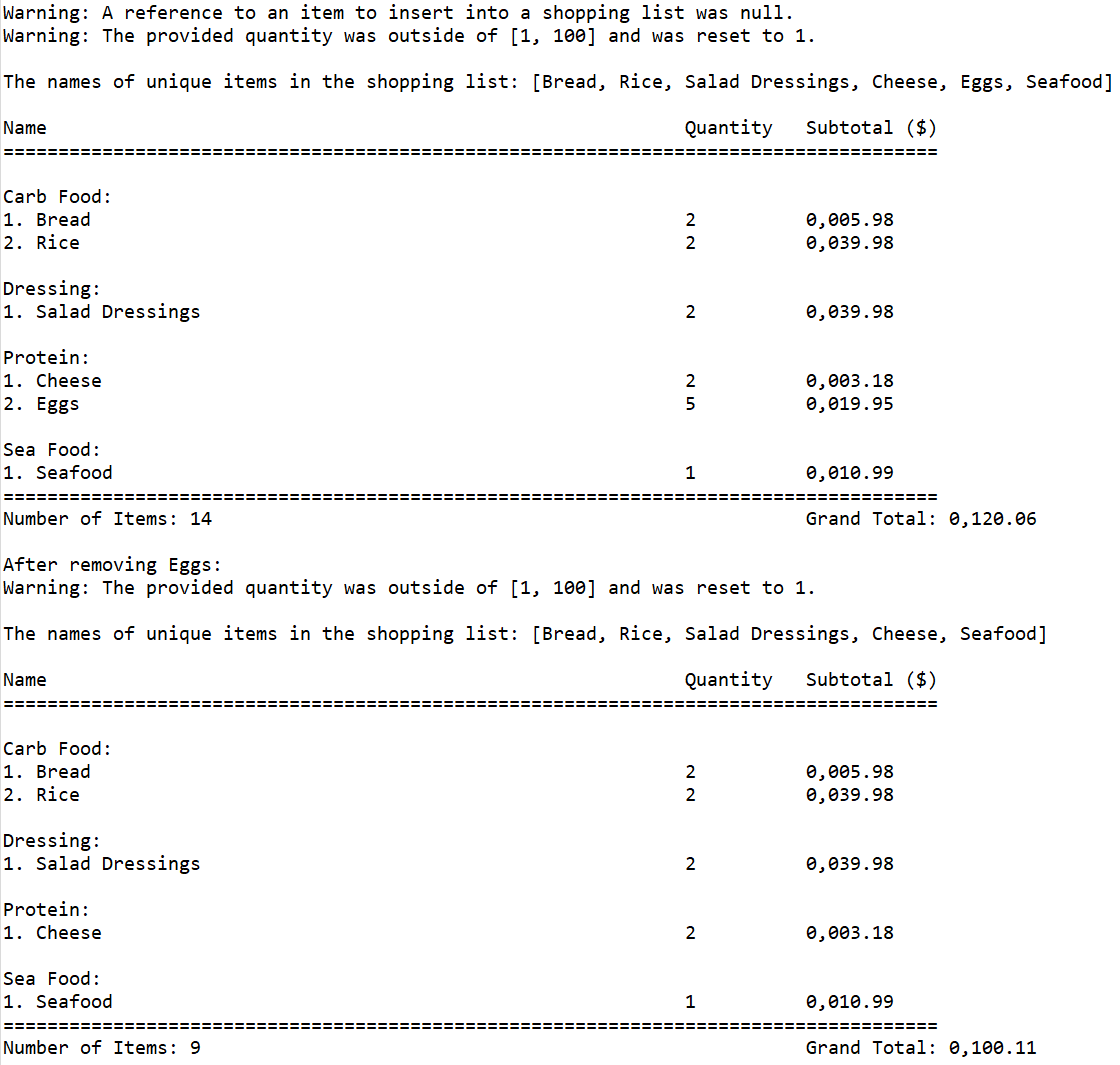


Considering a linked list based shopping list, for each item to be inserted, the linked list is traversed, starting from the head node, until a node with an item with a name greater than the name of the item to be inserted, and an insertion location, are found. In the best case, there is one pair of accesses of item names; in the worst case, *N*; the worst case complexity of traversal is O(*N*). At most complex, once an insertion location has been found, the reference of the node M containing the item to be inserted is set to the node N with the item with the greater name (or null); the reference of the node that previously referenced node N (if it exists) is set to node M. These are constant-complexity assignments. The worst-case complexity of linked list insertion is O(*N*).



**2: Advanced Shopping List**

*Main Output*



**3: Doubly Linked List**

*Main Output*

The doubly linked list is,

after creating a new doubly linked list,

[

]

after adding "A" to its head,

[

A

]

after adding "B" to its head,

[

B

A

]

after adding "C" to its tail,

[

B

A

C

]

after adding "D" to its tail,

[

B

A

C

D

]

after inserting "E" at its midpoint,

[

B

A

E

C

D

]

after removing the node at its head,

[

A

E

C

D

]

after adding "B" to its tail,

[

A

E

C

D

B

]

after adding "F" to its tail,

[

A

E

C

D

B

F

]

after setting its current node to its tail, an object with a current node with a string F.

after moving its current node away from its tail and towards it head, an object with a current node with a string B.

after removing the node at its midpoint,

[

A

E

C

B

F

]

after adding "D" to its head,

[

D

A

E

C

B

F

]

an object that contains a node with an element "B".

an object that does not contain a node with an element "G".

sleeps as

[

D

A

E

C

B

F

]