Assignment 2

Coverage: Modules 3 and 4.

This assignment consists of a written component and a programming component. Please read the course website carefully to ensure that you submit each component correctly.

Many of the questions refer to the SetPlus ADT. A SetPlus is a collection of items where there cannot be duplicates. The operations of the ADT are based on the Set ADT plus some additional operations.

Some of these operations depend on knowing when an item in the SetPlus has been accessed. An item is considered accessed when it is added to, or when it has been looked up in, the SetPlus. Deleting or looking at the MostRecent or LeastRecent item are not considered accesses. If you add an item that already exists, this is considered an access, even though the number items in the set is unchanged. For example if you performed the following operations in this order:

\[ S \leftarrow \text{Create()} \]
\[ \text{Add}(S, 15) \]
\[ \text{Add}(S, 7) \]
\[ \text{Add}(S, 10) \]
\[ \text{LookUp}(7) \]
\[ \text{Add}(S, 12) \]

the order of the items from most recently to least recently accessed would be: 12, 7, 10, 15.

A summary of the operations is shown below:

<table>
<thead>
<tr>
<th>Name</th>
<th>Returns</th>
<th>Modifies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create()</td>
<td>a new SetPlus</td>
<td></td>
</tr>
<tr>
<td>IsEmpty(S)</td>
<td>true if empty, else false</td>
<td></td>
</tr>
<tr>
<td>LookUp(S, item)</td>
<td>true if present, else false</td>
<td></td>
</tr>
<tr>
<td>Add(S, item)</td>
<td></td>
<td>adds item to S if it does not already exist</td>
</tr>
<tr>
<td>Delete(S, item)</td>
<td></td>
<td>removes item from S</td>
</tr>
<tr>
<td>DeleteMostRecent(S, n)</td>
<td></td>
<td>removes n the most recently accessed elements from S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If ( n \geq ) the size of the set, then S will be empty</td>
</tr>
<tr>
<td>DeleteLeastRecent(S, n)</td>
<td></td>
<td>removes the n least recently accessed elements from S</td>
</tr>
<tr>
<td></td>
<td></td>
<td>If ( n \geq ) the size of the set, then S will be empty</td>
</tr>
<tr>
<td>MostRecent(S)</td>
<td>the most recently accessed item;</td>
<td>returns None if S is empty</td>
</tr>
<tr>
<td>LeastRecent(S)</td>
<td>the least recently accessed item;</td>
<td>returns None if S is empty</td>
</tr>
</tbody>
</table>
Programming component

Please read the course website carefully to ensure that you are using the correct version of Python and the correct style. Approximately 80% of the marks of each question will be allocated to correctness, with the remaining 20% to style, including aspects of the design recipe. For this assignment you are required to include your own purpose, effects, contract, and requires sections for all of the functions and methods in the programming questions.

For things to work properly, all of the files you are using for a particular question must be saved in the same directory. Be sure you do not modify the names and number of parameters of the functions and methods in the provided files, so the automatic correctness tests will work properly.

1. [20 marks] Implement SetPlus using a doubly-linked list. The most recently accessed item should be at the beginning of the list and the least recently accessed item should be at the end of the list.

   - Start by creating a file `doublenode.py` that defines the class `DoubleNode` which implements a node in the doubly-linked list. Your module should implement `__init__`, as well as any other methods you need. You might find `__str__` useful in debugging your work.
   - Using the provided interface `setplusinterface.py` as a starting point, implement all of the methods of the ADT.

   Files to submit: `setplus.py` and `doublenode.py`

2. [10 marks] Quicksort is a divide and conquer algorithm. It is usually implemented recursively, but it can be implemented without using recursion. One way to implement it without using recursion is to use a stack to keep track of what parts of the list still need to be sorted.

   Write a function `quicksort` that will sort a Python list of numbers in non-descending order. The function will not return anything, but it will mutate the list it consumes. It should essentially implement the pseudocode that is included on the next page. Your solution must be non-recursive and should use a Stack ADT.

   You have been provided a starting file `quicksort.py` and an implementation of the Stack ADT that meets the interface described on Slide 3 of Module 4 (`Stack.py`). Your solution for `quicksort` should work for any implementation of the Stack ADT.
Here is pseudocode for a non-recursive version of Quicksort.

```python
function quicksort(alist)
    S ← Stack()
    if size(alist) > 1
        push the lowest and highest valid index values of alist onto S
    while S is not empty
        pop the low and high index values from the top of S
        pivot ← alist[low]
        i ← low + 1
        j ← high
        while pivot position is not known
            increase i by 1 until pivot >= alist[i]
            decrease j by 1 until pivot <= alist[j]
            if j < i
                pivot position ← j
            else
                swap elements of alist at positions i and j
        swap the pivot with the element at the pivot position
        if low < (pivot position - 1)
            push low and (pivot position - 1) onto S
        if high > (pivot position + 1)
            push (pivot position + 1) and high onto S
```

Any solution that uses recursion will receive 0 marks for correctness.

Files to submit: quicksort.py
**Written component**

For full marks, you are expected to provide a brief justification of any answer you provide. These questions refer to two array-based implementations of a SetPlus ADT.

**Option A**

- keeps track of the next empty position in the array. This value starts at 0, and increases by 1 each time you add or lookup an item.
- adds items to the next empty position in the array
- if the LookUp operation finds the item, then its original position in the array is filled with the value `None` and the item is added to the next empty position in the array.
- all the deleting operations replace deleted items with the value `None`  

**Option B**

- keeps track of two arrays, and the number of items in the set (`num`). The first array contains the items, and the second array is a parallel array containing numbers that indicate how recently the items have been accessed. For example, if you perform the operations described on the first page of the assignment, you would have one array containing the values (15, 7, 10, 12) and a parallel array containing the values, (5, 2, 3, 1).
- adds items to the next available position in the array in a way that is similar to the Add operation implementation described on Slide 28 of Module 3 for a Set. Sets the parallel array value to 1 for that item, and increases all other numbers in the parallel array by 1.
- if the LookUp operation finds the item, then sets the parallel array value to 1 for that item and increases all other numbers in the parallel array by 1.
- **Delete** removes items in the array in a way that is similar to the **Delete** operation implementation described on Slide 28 of Module 3 for a Set and updates the parallel array accordingly.
- **DeleteMostRecent** sorts the items in the array from their highest to lowest values in the parallel array. Reduces the value of `num` by `n` to a minimum of 0.
- **DeleteLeastRecent** sorts the items in the array from their lowest to highest values in the parallel array. Reduces the value of `num` by `n` to a minimum of 0.

All questions about runtime refer to analysis using asymptotic notation (Θ) as a function of `m`, which is the size of the array and `k` which is the number of items in the set. Also, assume that **Option B** uses linear insertion sort when it needs to sort values.
W1. [6 marks] For each option,

(a) briefly describe an algorithm that would implement \texttt{IsEmpty(S)}.
(b) describe the conditions that would produce the best case and worst case for \texttt{IsEmpty(S)}.

W2. [12 marks] For the following cases, which option would be considered the better choice. Briefly justify your answer. Your justification should include informal asymptotic analysis and a description of the situation that would generate a best/worst case. The justification may include any other factors that you think are relevant.

(a) \texttt{LookUp(S, item)} in the best case.
(b) \texttt{LookUp(S, item)} in the worst case.
(c) \texttt{Delete(S, item)} in the best case.
(d) \texttt{Delete(S, item)} in the worst case.
(e) \texttt{DeleteMostRecent(S, n)} in the best case.
(f) \texttt{DeleteLeastRecent(S, n)} in the worst case.