MTH 3300 Problem Set 7

Due May 11th at 11:59 PM

For the programming assignment, please follow the following naming convention for your Python files.
`mth3300_<lastname>_<firstname>_pset7_part<#>.py`

Make sure that your functions pass the initial tests I provide here!

1. (10 points) Write a recursive function `reverse_string(s)` that takes a string `s` and returns the string reversed. For example:
   - `reverse_string("hello")` should return `"olleh"`.
   - `reverse_string("recursion")` should return `"noisrucer"`.

   Constraints:
   - The function must be implemented recursively.
   - Do not use loops or built-in functions like `reversed()` or slicing (`[::-1]`).

   ```
   def recursive_string(s: str) -> str:
       ...
   ```

2. (10 points) Write a recursive function `nested_sum(lst)` that takes a list, which may contain integers or other lists of integers, and returns the sum of all integers in the list, including those in nested lists. For example:
   - `nested_sum([1, 2, [3, 4], 5])` should return `15`.
   - `nested_sum([[1, 2], [3, [4, 5]]])` should return `15`.

   Constraints:
   - The function must be implemented recursively.
   - Do not use loops or built-in functions like `sum()` to process the entire list.

   ```
   def nested_sum(lst: list) -> int:
       ...
   ```

3. (10 points) Write a function `search_rotated_array(arr, target)` that takes a sorted array that has been rotated at an unknown pivot and a target value. The function should return the index of the target in the array, or `-1` if it does not exist. For example:
   - `search_rotated_array([4, 5, 6, 7, 0, 1, 2], 0)` should return `4`.
   - `search_rotated_array([4, 5, 6, 7, 0, 1, 2], 3)` should return `-1`.

   Constraints:
   - The function must use **binary search** logic.
   - Do not use linear search or built-in functions like `index()`.

   ```
   def search_rotated_array(nums: list[int], target: int) -> int:
       ...
   ```

4. (10 points) Write a function `search_insert_position` that takes a sorted array of distinct integers and a value into insert. The function should return the index at which the target should be inserted to maintain the sorted order. If the target is already present, return its index. For example:
   - `search_insert_position([1, 3, 5, 6], 5)` should return `2`.
   - `search_insert_position([1, 3, 5, 6], 2)` should return `1`.

   Constraints:
   - The function must use **binary search** logic.
   - Do not use linear search or built-in functions like `index()`.

   ```
   def search_insert_position(nums: list[int], value: int) -> int:
       ...
   ```

5. (10 points) Write a function `count_nodes` that takes the head of a singly linked list and returns the number of nodes in the list. For example:
   - `count_nodes(1 -> 2 -> 3 -> 4 -> 5)` should return `5`.
   - `count_nodes(1 -> 2 -> 3)` should return `3`.

   In the example above assume that `1 -> 2 -> 3 -> 4 -> 5` is a linked list where `1` is the head and `2`, `3`, `4`, and `5` are the next nodes.

   ```
   class ListNode:
       def __init__(self, value=0, next=None):
   ```

```
        self.value = value
        self.next = next

def count_nodes(head: ListNode) -> int:
    ...
```

6. (10 points) Write a function `find_middle` that takes the head of a singly linked list and returns the middle node. If there are two middle nodes, return the second middle node. For example:
   - `find_middle(1 -> 2 -> 3 -> 4 -> 5)` should return `3`.
   - `find_middle(1 -> 2 -> 3 -> 4 -> 5 -> 6)` should return `4`.

   In the example above assume that `1 -> 2 -> 3 -> 4 -> 5` is a linked list where `1` is the head and `2`, `3`, `4`, and `5` are the next nodes.

   Hint: Feel free to look up the "Tortoise and Hare" algorithm for this problem.

```
class ListNode:
    def __init__(self, value=0, next=None):
        self.value = value
        self.next = next

def find_middle(head: ListNode) -> ListNode:
    ...
```