Solution Review: Find Second Maximum Value in a List

This review provides a detailed analysis of the different ways to find second maximum value in a list.

We'll cover the following

Solution #1: Sort and index

```
def find_second_maximum(lst):
    lst.sort()
    if len(lst) >= 2:
        return lst[-2]
    else:
        return None


print(find_second_maximum([9, 2, 3, 6]))

```

We first sort the list, then index and return the second largest element from the sorted list. Note that the second largest element can be accessed at the second last index of the list. We use the generic Python `sort()` function, but in a real interview, you should implement your favorite sorting algorithm here! Remember to always take care of edge cases such as the input list being smaller than two elements here.

Time Complexity#

The time complexity of this solution depends on the running time of `sort()` function. Assuming that Python `sort()` function runs in an optimal time of $O(nlogn)$, our solution also takes $O(nlogn)$ time."

> **Caveat**: Note that this solution won't work if duplicates of the first largest number exist. For instance, it would not work with a list like `[1,2,4,4]` since it would return 4 which is at the second last index of the sorted list. But, it is the largest element and not the correct answer.

Solution #2: Traversing the list twice#

```
def find_second_maximum(lst):
    first_max = float('-inf')
    second_max = float('-inf')
    # find first max
    for item in lst:
        if item > first_max:
            first_max = item
    # find max relative to first max
    for item in lst:
        if item != first_max and item > second_max:
            second_max = item
    return second_max


print(find_second_maximum([9, 2, 3, 6]))
```

We traverse the list twice. In the first traversal, we find the maximum element. In the second traversal, find the greatest element less than the element obtained in the first traversal.

Time Complexity#

The time complexity of the solution is $O(n)$ since the list is traversed twice.

Solution #3: Finding the Second Maximum in one Traversal#

```
def find_second_maximum(lst):
   if (len(lst) < 2):
       return
   # initialize the two to infinity
   max_no = second_max_no = float('-inf')
   for i in range(len(lst)):
       # update the max_no if max_no value found
       if (lst[i] > max_no):
           second_max_no = max_no
           max_no = lst[i]
       # check if it is the second_max_no and not equal to max_no
       elif (lst[i] > second_max_no and lst[i] != max_no):
           second_max_no = lst[i]
   if (second_max_no == float('-inf')):
       return
   else:
       return second_max_no


print(find_second_maximum([9, 2, 3, 6]))
```

We initialize two variables `max_no` and `secondmax` to `-inf`. The traverse the list, and if the current element in the list is greater than the maximum value, then set `secondmax` to `max_no` and `max_no` to the current element. If the current element is greater than the second maximum number and not equal to maximum number, then update `secondmax` to store the value of the current variable. Finally, return the value stored in `secondmax`.

Time Complexity#

This solution is in $O(n)$ since the list is traversed once only.