CS566Lab2-2024

September 11, 2024

0.1 Lab 2. Introduction to algorithms

This is second Lab for CS 566. This problem was given in lecture.

0.2 Task 1. Solve the problem "Majority element" from https://leetcode.com/problems/majority-element/description/ using Python3

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[ ]:
```python
from typing import List

class Solution:
    def majorityElement(self, nums: List[int]) -> int:
        # implement Working Algorithm
        hash_map = {}
        for num in nums:
            if num in hash_map:
                hash_map[num] += 1
            else:
                hash_map[num] = 1
        threshold = len(nums)//2
        for element, occurrences in hash_map.items():
            if occurrences > threshold:
                return element
```

0.2.1 Do not modify the testing code below. If you get message "Mistake in test case #", it means that you algorithm is incorrect.

[ ]:
```python
#test_case_1
expected, nums = 3, [3,2,3]
actual = Solution().majorityElement(nums)
assert expected==actual, "Mistake in test case 1"

#test_case_2
expected, nums = 2, [2,2,1,1,1,2,2]
actual = Solution().majorityElement(nums)
```

```
assert expected==actual, "Mistake in test case 2"
print('All tests pass')
```

```
All tests pass
```

0.2.2 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O - O(n)

Time Analysis: O - O(n)

0.3 Task 2. Implement Insertion Sort (4 points)

[ ]:
```
# Insertion Sort
# Please implement the function which takes as an input list of numbers and␣
 ↪returns the sorted list
# Use Insertion Sort algorithm
def insertion_sort(nums):
    # implement Working Algorithm
    n = len(nums)
    for i in range(1,n):
      key = nums[i]
      j = i-1
      while j>=0 and nums[j]>key:
        nums[j+1] = nums[j]
        j-=1
      nums[j+1] = key
    return nums
```

[ ]:
```
# testing code
# do NOT modify testing code below
def test_sorting(sorting_algorithm):
    nums = [5,4,3,2,10]
    expected = [2,3,4,5,10]
    actual = sorting_algorithm(nums)
    assert expected == actual, "Mistake in implementation"
```

[ ]:
```
# if your implementation is correct, you should see "OK"
# if your implementation is not correct, you will see "Mistake in␣
 ↪implementation"
test_sorting(insertion_sort)
print('OK')
```

```
OK
```

0.3.1 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O - O(1)

Time Analysis: O - O(n^2)

Invariant: The left side of the array before the key (ie. the elements before the key) will always be sorted.

Describe the 3 pieces of invariant:

Initialization: Before the first iteration, the first element of the array is considered sorted.

Maintenance: The body of the for loop works by moving the values in A[i-1], A[i-2], etc.

Termination: The loop terminates when i reaches the end and the entire array is sorted.

0.4 Task 3. Write the function which will compute factorial. Use recursion, meaning call the function itself inside of your function.

Use the box below, to paste the working code. The format of the code should be identical to LeetCode platform. (4 points)

[ ]:
```
class Solution:
    def factorial(self, n: int) -> int:
        # implement Working Algorithm
        if n == 1 or n == 0:
          return 1
        return n * self.factorial(n-1)
```

[ ]:
```
#test_case_1
expected, nums = 120, 5
actual = Solution().factorial(nums)
assert expected==actual, "Mistake in test case 1"

#test_case_2
expected, nums = 24, 4
actual = Solution().factorial(nums)
assert expected==actual, "Mistake in test case 2"

#test_case_3
expected, nums = 6, 3
actual = Solution().factorial(nums)
assert expected==actual, "Mistake in test case 2"
```

0.4.1 Write analysis of the Memory Complexity and Time Complexity using Asymptotic Notation O. (1 point)

Memory Analysis: O - O(n)

Time Analysis: O - O(n)