CS566HW3-2024

September 21, 2024

0.1 Lab 3. Introduction to algorithms

This is third Homework for CS 566.

0.2 Task 1. Solve the problem "Maximum Subarray" from https://leetcode.com/problems/maximum-subarray/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 maxSubArray(self, nums: List[int]) -> int:
        # implement working algorithm
        def maxMiddleSum(arr, mid):
          left_sum = float('-inf')
          temp_sum = 0
          for i in range(mid - 1, -1, -1):
              temp_sum += arr[i]
              left_sum = max(left_sum, temp_sum)

          right_sum = float('-inf')
          temp_sum = 0
          for i in range(mid, len(arr)):
              temp_sum += arr[i]
              right_sum = max(right_sum, temp_sum)

          return left_sum + right_sum

        if len(nums) == 1:
          return nums[0]

        mid = len(nums) // 2
        left_max = self.maxSubArray(nums[:mid])
        right_max = self.maxSubArray(nums[mid:])
        cross_max = maxMiddleSum(nums, mid)
```

```python
        return max(left_max, right_max, cross_max)
```

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 = 6, [-2,1,-3,4,-1,2,1,-5,4]
actual = Solution().maxSubArray(nums)
assert expected==actual, "Mistake in test case 1"

#test_case_2
expected, nums = 1, [1]
actual = Solution().maxSubArray(nums)
assert expected==actual, "Mistake in test case 2"

#test_case_3
expected, nums = 23, [5,4,-1,7,8]
actual = Solution().maxSubArray(nums)
assert expected==actual, "Mistake in test case 3"
print('OK')
```

```
OK
```

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

Memory Analysis: O(1)

Time Analysis: O(nlogn)

0.2.3 Task 2. Theoretical problem. (5 points)

####Analyzing Strassen's Matrix Multiplication

Analyze the time complexity of Strassen's Matrix Multiplication algorithm using the Master Theorem. Details: Strassen's algorithm reduces the number of multiplications in matrix multiplication from the standard $O(n^3)$ to $O(n^{2.81})$. The recurrence relation of Strassen algorithm is: $T(n) = 7 * T(n/2) + O(n^2)$

Provide your answer in Latex, or write it on paper and embed screenshot into colab

Your solution goes here

$a = 7, b = 2, f(n) = O(n^2)$

$lob__ba = log__27 2.81 $

Since $n^{log_b a} = n^{2.81}$ which grows faster than $O(n^2)$, therefore we apply case 1 of Master Theorem.

Case 1:

$T(n) = \Theta(nlog_b\ a)$

$= \Theta(n^{2.81})$

0.2.4 Task 3. Theoretical problem (5 points)

Analyzing Karatsuba Algorithm for multiplyng Large Integers Analyze the time complexity of Karatsuba Algorithm. Its recurrence relation is:

$T(n) = 3 * T(n/2) + O(n)$

Provide your answer in Latex, or write it on paper and embed screenshot into colab

Your solution goes here

$a = 3, b = 2, f(n) = O(n)$

$lob_b a = lob_2 3 \approx 1.59$

$n^{lob_b a} = n^{1.59}$, which grows faster than O(n) so apply case 1 of Master Theorem

Case 1:

$f(n) = O(nlog\ ba - \epsilon) f(n) = O(nlog_b\ a - \epsilon)$ for >0, then:

$T(n) = \Theta(nlog\ _b a)$

$= \Theta(n^{1.59})$