Assignment 2 - Coding component

● Graded

Student

Tyler NGUYEN

Total Points

100 / 100 pts

Autograder Score

0.0 / 0.0

Passed Tests

1.1) Test case I (0/0)
1.2) Test case II (0/0)
1.3) Test case III (0/0)
1.4) Test case IV (0/0)
1.5) Test case V (0/0)

Question 2

Clean Code

20 / 20 pts

✓ **– 0 pts** Correct

– 10 pts Insufficient docs in code.

– 20 pts No documents/comments in code.

Question 3

Manual Grader

80 / 80 pts

✓ **– 0 pts** Correct

– 50 pts The code must be an exact implementation of the algorithm covered in the lecture.

– 75 pts Your solution must be recursive.

Autograder Results

1.1) Test case I (0/0)

1.2) Test case II (0/0)

1.3) Test case III (0/0)

1.4) Test case IV (0/0)

1.5) Test case V (0/0)

Submitted Files

▾ array_checker.py   Download

```python
#CPSC 413 Assignment 2 question 4
#Author: Tyler Nguyen
#UCID: 30158563
class Checker(object):
    def __init__(self, array):
        self.array = array

    def max_sums(self, start, end):
        def find_max_sums(start, end): #helper function which uses recursion to find the other sums needed to find the maximum subarray sum
            if start == end: #the base case where if start and end are equivalent it means the subarray has only one element
                num = self.array[start] #gets the number at the specified element
                return (max(0, num), max(0, num), max(0, num), num) #returns the four values max-left-aligned, max-right-aligned,max-sum and total-sum

            middle_index = (start + end) // 2 #determines the middle_indexdle index of the current subarray which divides the problem into two smaller subproblems
            left_half = find_max_sums(start, middle_index) #recursively calls the function for the left half of the subarray
            right_half = find_max_sums(middle_index + 1, end) #recursively calls the function for the right half of the subarray
            max_left_aligned_sum = max(left_half[0], left_half[3] + right_half[0]) #determines the maximum left-aligned sum, by finding the max of the
            #left-aligned sum of the left subarray and the sum of the entire left subarray + the left-aligned sum of the right subarray
            max_right_aligned_sum = max(right_half[1], right_half[3] + left_half[1]) #determines the maximum right-aligned sum, by finding the max of
            #right-aligned sum of the right subarray and the sum of the entire right subarray + the right-aligned sum of the left subarray
            sum_of_all_array = left_half[3] + right_half[3] #determines the total sum of the elements in the current subarray through adding the total
            #sums of the left and right subarrays
            max_subarray_sum = max(left_half[2], right_half[2], left_half[1] + right_half[0]) #determines the maximum subarray sum, by finding the
            #maximum of the maximum subarray sum in th eleft subarray, the maximum subarray sum in the right subarray and the sum of the right-aligned
            #part of the left subarray + the left-aligned part of the right subarray
            return (max_left_aligned_sum, max_right_aligned_sum, max_subarray_sum, sum_of_all_array) #returns the determined sums

        _, _, max_subarray_sum, _ = find_max_sums(start, end) #calls the helper function with the start and end variables as well as
        #takes specifically only the maximum subarray sum
        return max_subarray_sum #returns the maximum subarray sum
```