```
import math
import sys
import random

import matplotlib.pyplot as plt
import numpy as np

recursion_counter = [0]

def generate_random_number(min: int, max: int):
    """
    Accepts min and max integer value as range to return a random
number within that range
    """
    return random.randint(min, max)

def get_random_list(length: int):
    """
    Accepts length integer and return a random list of numbers of the
specified length
    """
    arr = []
    for i in range(0, length):
        arr.append(generate_random_number(-999, 999))

    return arr

def max_sub_array_using_brute_force(arr):
    """
    Accepts a random array and returns the maximum sum, beginning
index and last index of the sub using brute-force method
    """
    counter # 0

    # Initialize the max as the first element of the array
    max_sum = -sys.maxsize

    left_index = right_index = 0

    for i in range(0, len(arr)):
        counter += 1

        # Initialize current sum with every iteration of the outer-
loop
        current_sum = 0

        for j in range(i, len(arr)):
            counter += 1

            # Add array elements to current-sum to maintain
```

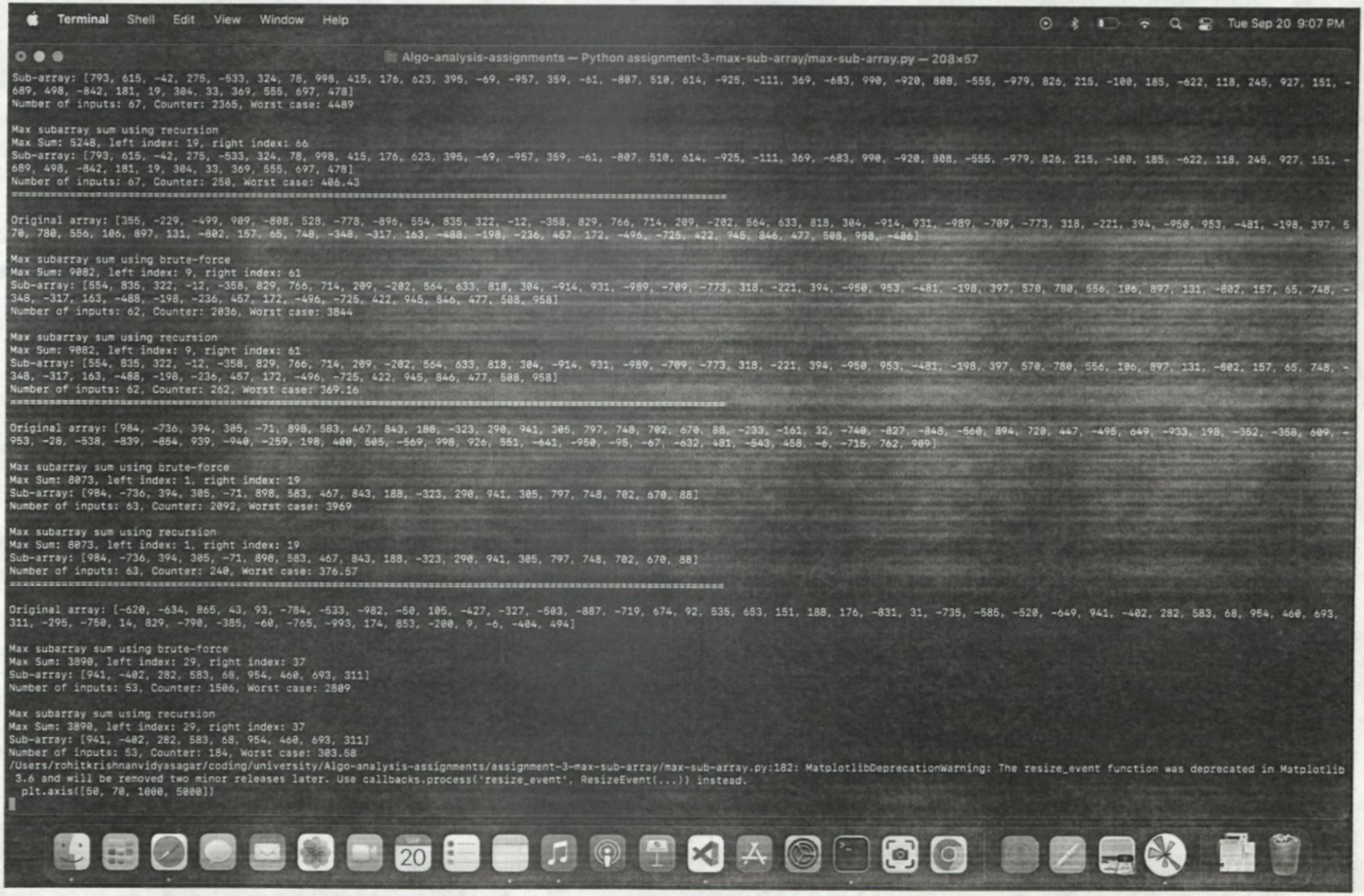
```
            current_sum += arr[j]

            # Replace the max, left_index and right_index if current-
sum is greater than max
            if current_sum > max_sum:
                counter += 1

                max_sum = current_sum

                left_index = i
                right_index = j

    return max_sum, left_index, right_index, counter

def max_crossing_sub_array_sum(arr, left_index, mid_index,
right_index):
    """
    Accepts the array, left index, mid index and right index and
returns the max sum of a sub array and it's indices
    """
    current_sum = 0
    left_max_sum = -sys.maxsize
    left_max_begin_index = mid_index

    # Iterate from the mid to left index and find the max left sum
(Iterating back)
    for i in range(mid_index, left_index - 1, -1):
        current_sum = current_sum + arr[i]

        if current_sum > left_max_sum:
            recursion_counter[0] += 1
            left_max_sum = current_sum
            left_max_begin_index = i

    current_sum = 0
    right_max_sum = -sys.maxsize
    right_max_end_index = mid_index

    # Iterate from the mid to right index and find the max right sum
(Iterating forward)
    for j in range(mid_index, right_index + 1):
        current_sum = current_sum + arr[j]

        if current_sum > right_max_sum:
            recursion_counter[0] += 1
            right_max_sum = current_sum
            right_max_end_index = j

    # Calcuate the crossing sum by adding the left max sum, right max
sum
```

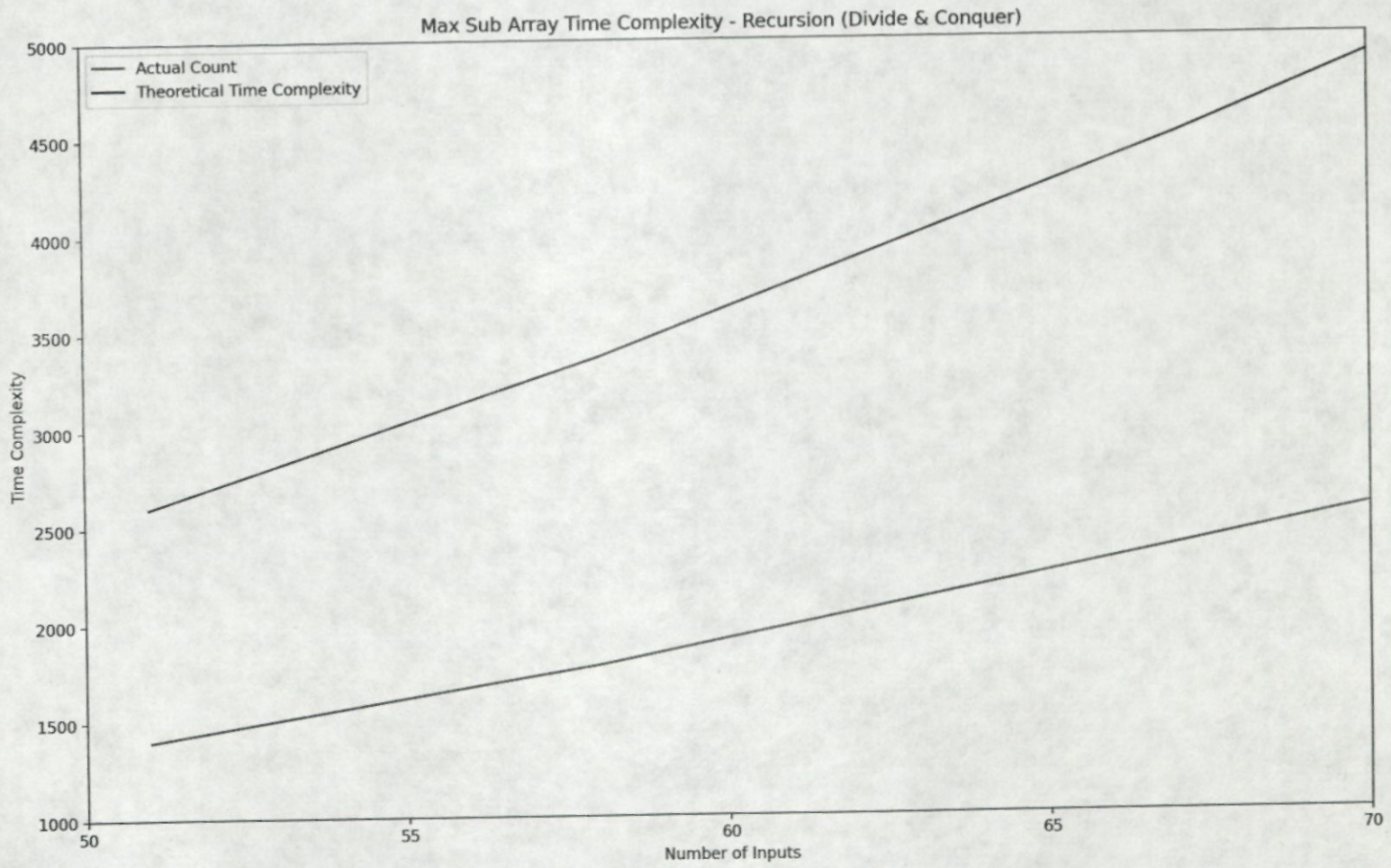
```
    # Subtracting the mid-index because it is getting added twice
 (both in left max sum and right max sum)
    crossing_sum = left_max_sum + right_max_sum - arr[mid_index]

    # Return the max sum from crossing sum, left sum and right sum
    if crossing_sum > left_max_sum and crossing_sum > right_max_sum:
        return crossing_sum, left_max_begin_index, right_max_end_index
    elif left_max_sum > crossing_sum and left_max_sum > right_max_sum:
        return left_max_sum, left_max_begin_index, mid_index
    else:
        return right_max_sum, mid_index, right_max_end_index

def max_sub_array_using_recursion(arr, left_index, right_index):
    """
    Accepts and array, the left index(begin) of the array and the
right index (end) of the array and returns
    the maximum sum of a subarray and the beginning index and the
final index of the subarray
    """
    if left_index >= right_index:
        return arr[left_index], left_index, right_index

    mid_index = (left_index + right_index) // 2

    left_sub_array_sum, left_begin_index, left_end_index =
max_sub_array_using_recursion(arr, left_index, mid_index)

    right_sub_array_sum, right_begin_index, right_end_index =
max_sub_array_using_recursion(arr, mid_index+1, right_index)

    crossing_sub_array_sum, crossing_begin_index, crossing_end_index =
max_crossing_sub_array_sum(arr, left_index, mid_index, right_index)

    # Return the max sum from crossing sum, left sum and right sum
    if left_sub_array_sum > right_sub_array_sum  and
left_sub_array_sum > crossing_sub_array_sum:
        return left_sub_array_sum, left_begin_index, left_end_index
    elif right_sub_array_sum > left_sub_array_sum and
right_sub_array_sum > crossing_sub_array_sum:
        return right_sub_array_sum, right_begin_index, right_end_index
    else:
        return crossing_sub_array_sum, crossing_begin_index,
crossing_end_index


def __init__():
    min_number_of_sets = 6
    max_number_of_sets = 10

    total_number_of_sets = generate_random_number(min_number_of_sets,
```

```
max_number_of_sets)

    brute_force_input_to_counter_dict = dict()

    recursion_input_to_counter_dict = dict()

    for num in range(0, total_number_of_sets):
        array_length = generate_random_number(50, 70)

        array = get_random_list(array_length)


print('==========================================================================
==========================================')
        print(f'\nOriginal array: {array}')

        max_sum_brute_force, left_index, right_index,
brute_force_counter = max_sub_array_using_brute_force(array)
        brute_force_input_to_counter_dict[array_length] =
brute_force_counter

        print('\nMax subarray sum using brute-force')

        print(f'Max Sum: {max_sum_brute_force}, left index:
{left_index + 1}, right index: {right_index + 1}')
        print(f'Sub-array: {array[left_index: right_index + 1]}')
        print(f'Number of inputs: {array_length}, Counter:
{brute_force_counter}, Worst case: {array_length * array_length}')

        recursion_counter[0] = 0

        print('\nMax subarray sum using recursion')
        max_sum_recursion, left_index, right_index =
max_sub_array_using_recursion(array, 0, array_length - 1)
        recursion_input_to_counter_dict[array_length] =
recursion_counter[0]

        print(f'Max Sum: {max_sum_recursion}, left index: {left_index
+ 1}, right index: {right_index + 1}')
        print(f'Sub-array: {array[left_index: right_index + 1]}')
        print(f'Number of inputs: {array_length}, Counter:
{recursion_counter[0]}, Worst case: {round((array_length *
math.log(array_length, 2)), 2)}')


    input_length_array = []
    counter_array = []
    theoretical_time_complexity_array = []

    for key in brute_force_input_to_counter_dict:
```

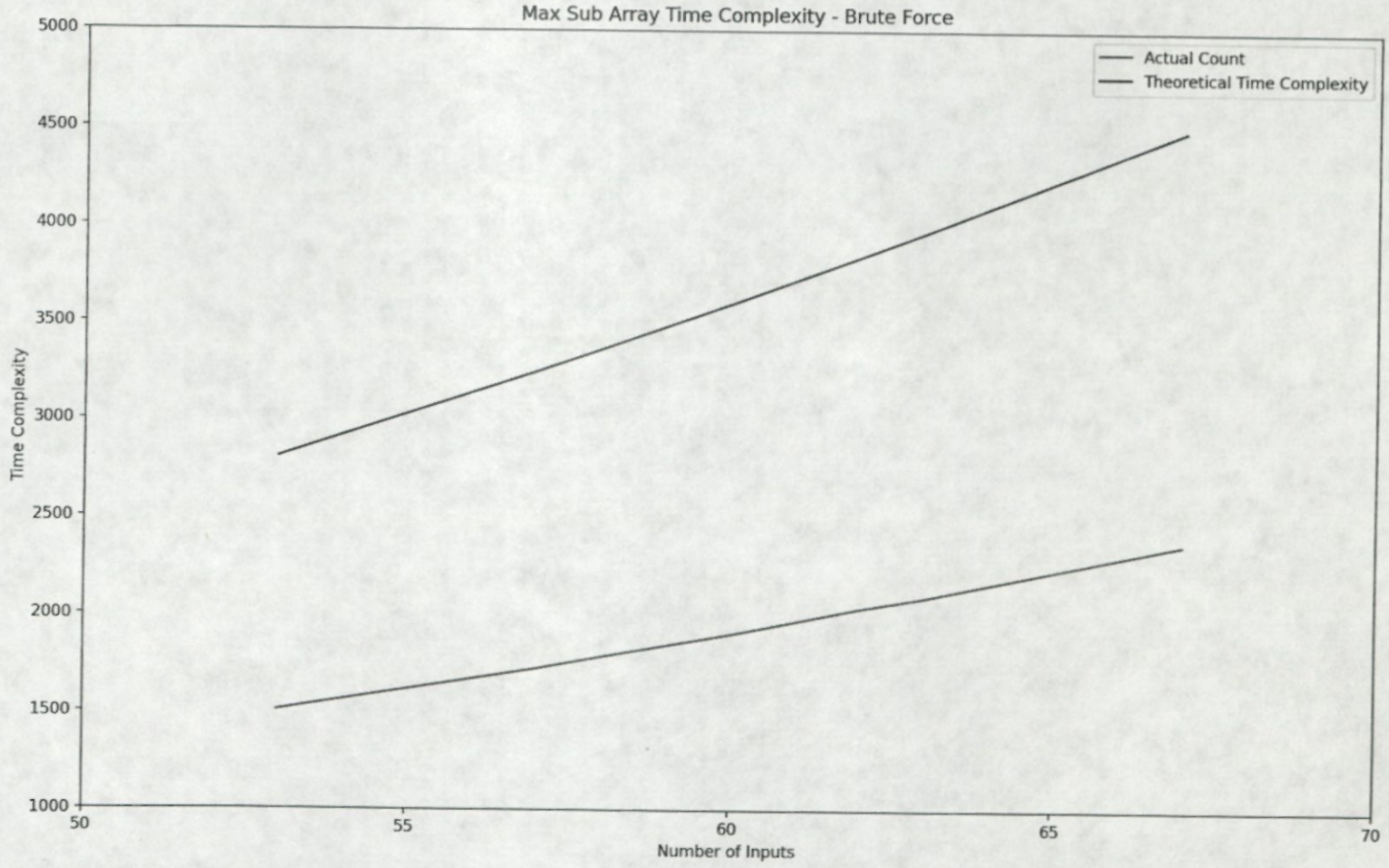
```
            input_length_array.append(key)
            counter_array.append(brute_force_input_to_counter_dict[key])
            theoretical_time_complexity_array.append(key * key)

        input_length_array.sort()
        counter_array.sort()
        theoretical_time_complexity_array.sort()

        x = np.array(input_length_array)
        y1 = np.array(counter_array)
        y2 = np.array(theoretical_time_complexity_array)

        plt.axis([50, 70, 1000, 5000])

        plt.xticks([50, 55, 60, 65, 70])


        plt.xlabel('Number of Inputs')
        plt.ylabel('Time Complexity')
        plt.title('Max Sub Array Time Complexity - Brute Force')

        plt.plot(x, y1, color='red')
        plt.plot(x, y2, color='blue')
        plt.legend(['Actual Count', 'Theoretical Time Complexity'])

        plt.show()

        recursion_counter_array = []
        recursion_theoretical_time_complexity_array = []

        for key in recursion_input_to_counter_dict:

    recursion_counter_array.append(recursion_input_to_counter_dict[key])
            recursion_theoretical_time_complexity_array.append(round((key
    * math.log(key, 2)), 2))

        recursion_counter_array.sort()
        recursion_theoretical_time_complexity_array.sort()

        y3 = np.array(recursion_counter_array)
        y4 = np.array(recursion_theoretical_time_complexity_array)

        plt.axis([50, 70, 1000, 5000])

        plt.xticks([50, 55, 60, 65, 70])


        plt.xlabel('Number of Inputs')
        plt.ylabel('Time Complexity')
        plt.title('Max Sub Array Time Complexity - Recursion (Divide &
```

```
Conquer)')

    plt.plot(x, y3, color='yellow')
    plt.plot(x, y4, color='green')
    plt.legend(['Actual Count', 'Theoretical Time Complexity'])

    plt.show()


__init__()
```


Terminal Shell Edit View Window Help
Tue Sep 20 9:07 PM
Algo-analysis-assignments — Python assignment-3-max-sub-array/max-sub-array.py — 208×57
Sub-array: [793, 615, -42, 275, -533, 324, 78, 998, 415, 176, 623, 395, -69, -957, 359, -61, -807, 510, 614, -925, -111, 369, -683, 990, -920, 808, -555, -979, 826, 215, -100, 185, -622, 118, 245, 927, 151, -689, 498, -842, 181, 19, 304, 33, 369, 555, 697, 478]
Number of inputs: 67, Counter: 2365, Worst case: 4489
Max subarray sum using recursion
Max Sum: 5248, left index: 19, right index: 66
Sub-array: [793, 615, -42, 275, -533, 324, 78, 998, 415, 176, 623, 395, -69, -957, 359, -61, -807, 510, 614, -925, -111, 369, -683, 990, -920, 808, -555, -979, 826, 215, -100, 185, -622, 118, 245, 927, 151, -689, 498, -842, 181, 19, 304, 33, 369, 555, 697, 478]
Number of inputs: 67, Counter: 250, Worst case: 406.43
Original array: [355, -229, -499, 909, -808, 528, -778, -896, 554, 835, 322, -12, -358, 829, 766, 714, 209, -202, 564, 633, 818, 304, -914, 931, -989, -709, -773, 318, -221, 394, -950, 953, -481, -198, 397, 570, 780, 556, 106, 897, 131, -802, 157, 65, 748, -348, -317, 163, -488, -198, -236, 457, 172, -496, -725, 422, 945, 846, 477, 508, 958, -486]
Max subarray sum using brute-force
Max Sum: 9082, left index: 9, right index: 61
Sub-array: [554, 835, 322, -12, -358, 829, 766, 714, 209, -202, 564, 633, 818, 304, -914, 931, -989, -709, -773, 318, -221, 394, -950, 953, -481, -198, 397, 570, 780, 556, 106, 897, 131, -802, 157, 65, 748, -348, -317, 163, -488, -198, -236, 457, 172, -496, -725, 422, 945, 846, 477, 508, 958]
Number of inputs: 62, Counter: 2036, Worst case: 3844
Max subarray sum using recursion
Max Sum: 9082, left index: 9, right index: 61
Sub-array: [554, 835, 322, -12, -358, 829, 766, 714, 209, -202, 564, 633, 818, 304, -914, 931, -989, -709, -773, 318, -221, 394, -950, 953, -481, -198, 397, 570, 780, 556, 106, 897, 131, -802, 157, 65, 748, -348, -317, 163, -488, -198, -236, 457, 172, -496, -725, 422, 945, 846, 477, 508, 958]
Number of inputs: 62, Counter: 262, Worst case: 369.16
Original array: [984, -736, 394, 305, -71, 898, 583, 467, 843, 188, -323, 290, 941, 305, 797, 748, 702, 670, 88, -233, -161, 32, -740, -827, -848, -560, 894, 720, 447, -495, 649, -933, 198, -352, -358, 609, -953, -28, -538, -839, -854, 939, -940, -259, 198, 400, 505, -569, 998, 926, 551, -641, -950, -95, -67, -632, 481, -543, 458, -6, -715, 762, 989]
Max subarray sum using brute-force
Max Sum: 8073, left index: 1, right index: 19
Sub-array: [984, -736, 394, 305, -71, 898, 583, 467, 843, 188, -323, 290, 941, 305, 797, 748, 702, 670, 88]
Number of inputs: 63, Counter: 2092, Worst case: 3969
Max subarray sum using recursion
Max Sum: 8073, left index: 1, right index: 19
Sub-array: [984, -736, 394, 305, -71, 898, 583, 467, 843, 188, -323, 290, 941, 305, 797, 748, 702, 670, 88]
Number of inputs: 63, Counter: 240, Worst case: 376.57
Original array: [-620, -634, 865, 43, 93, -784, -533, -982, -50, 105, -427, -327, -503, -887, -719, 674, 92, 535, 653, 151, 188, 176, -831, 31, -735, -585, -520, -649, 941, -402, 282, 583, 68, 954, 460, 693, 311, -295, -750, 14, 829, -790, -385, -60, -765, -993, 174, 853, -200, 9, -6, -484, 494]
Max subarray sum using brute-force
Max Sum: 3890, left index: 29, right index: 37
Sub-array: [941, -402, 282, 583, 68, 954, 460, 693, 311]
Number of inputs: 53, Counter: 1506, Worst case: 2809
Max subarray sum using recursion
Max Sum: 3890, left index: 29, right index: 37
Sub-array: [941, -402, 282, 583, 68, 954, 460, 693, 311]
Number of inputs: 53, Counter: 184, Worst case: 303.58
/Users/rohitkrishnanvidyasagar/coding/university/Algo-analysis-assignments/assignment-3-max-sub-array/max-sub-array.py:182: MatplotlibDeprecationWarning: The resize_event function was deprecated in Matplotlib 3.6 and will be removed two minor releases later. Use callbacks.process('resize_event', ResizeEvent(...)) instead.
plt.axis([50, 70, 1000, 5000])


Max Sub Array Time Complexity - Recursion (Divide & Conquer)
Actual Count
Theoretical Time Complexity
Time Complexity
Number of Inputs
5000
4500
4000
3500
3000
2500
2000
1500
1000
50
55
60
65
70

x=57.580 y=2782.


Max Sub Array Time Complexity - Brute Force
Actual Count
Theoretical Time Complexity
Time Complexity
Number of Inputs
5000
4500
4000
3500
3000
2500
2000
1500
1000
50
55
60
65
70