MAXIMUM SUM SUBARRAY

GROUP MEMBERS:

KODI PRAVALLIKA - IIT2019234
VISHWAM SHRIRAM MUNDADA - IIT2019235
NOONSAVAATH SAMYUKTA - IIT2019236

ABSTRACT

The project aims at finding the maximum sum subarray in the given 2D array.

CONTENTS

INTRODUCTION

We are given given a 2D array, our aim is to find the maximum sum subarray in it.

For solving this we have analyzed two different approaches:-

- Brute force.
- Dynamic Programming.

These algorithms are tested for different sample test cases for time complexity and space complexity. In our report, we tend to focus on establishing a rule or a relationship between the time and input data.

ALGORITHM DESIGN

Brute force(Naive method)

- In the brute force approach we try to check every possible rectangle in the given n X m 2D array(where n,m are number of rows and columns respectively).
- Set the position of the top-left and bottom-right corners of the sub-rectangle and adding the integers within it while iterating through all the rows sequentially.
- Parallelly we try to find the maximum subarray sum value.

PSEUDO CODE (Naive approach)

```
int A[101][101]
function MAIN()
    maxSum ←  INT_MIN
    tempSum ←  0
    x ← 0, y ← 0, z ← 0, w ← 0,n ← 0, m← 0
    input n, m
    for i ← 0 to n-1
        for j ← 0 to m-1
                input A[i][j]
        end
    end

for i ← 0 to n-1
        for j ← 0 to m-1
                for k ← 0 to n-1
                        for l ← 0 to m-1
                                tempSum = FINDSUM(i, j, k, l)
                                if tempSum > maxSum
                                x ← i
                                y ← j
                                z ← k
                                w ← l
                                maxSum = tempSum
                                end if
                        end
                end
        end
end

print x, y, z, w, maxSum
```

MAXSUM PSEUDO CODE

```
function FINDSUM(x, y, z, w)
    sum ← 0
    for i ← x to z
        for j ← y to w
            sum ← sum + A[i][j]
        end
    end
return sum
```

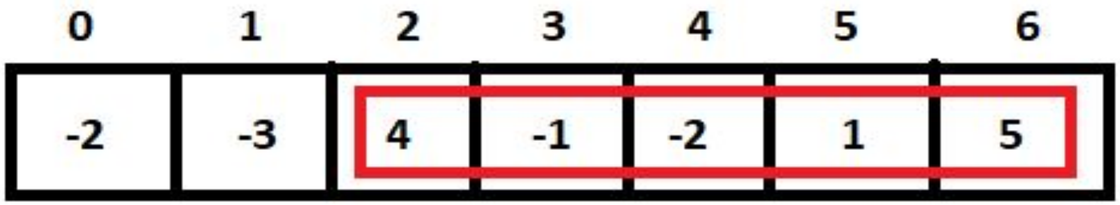
KADANE’S ALGORITHM FOR 1D ARRAY

- This is an efficient approach to find the sum of contiguous subarray within a one-dimensional array of numbers that has the largest sum.

- The idea of Kadane’s algorithm is to look for all positive contiguous segments of the array.

- Keep track of maximum sum contiguous segment among all positive segments. Each time we get a positive-sum compare it with maximum sum so far and update it, if it is greater than maximum sum till then.

EXAMPLE



In this example maximum contiguous array sum is = 4 + (-1) + (-2) + 1 + 5
= 7

ALGORITHM DESIGN
Dynamic programming approach

- We use Kadane's algorithm and prefix sum to reduce time complexity to $O(n^2 \times m)$.
- The idea is to fix the top and bottom rows one by one and find the maximum sum contiguous columns for every top and bottom row pair.
- We basically find left and right column numbers (which have maximum sum) for every fixed up and bottom row pair. To find the left and right column numbers, calculate the sum of elements in every column from top to bottom and store these sums in an array say temp[].
- temp[i] indicates sum of elements from top to bottom in column i. If we apply Kadane's 1D algorithm on temp[], and get the maximum sum subarray of temp, this maximum sum would be the maximum possible sum with top and bottom as boundary rows. To get the overall maximum sum, we compare this sum with the maximum sum so far

EXAMPLE

1	2	-1	-4	-20
-8	-3	4	2	1
3	8	10	1	3
-4	-1	1	7	-6

In the following 2D array, the maximum sum subarray is highlighted with red rectangle and sum of this subarray is = (-3)+4+2+8+10+1+(-1)+1+7

= 29.

DYNAMIC PROGRAMMING PSEUDO CODE

```
function KADANE(V)
    maxSum ← INT_MIN, tempSum ← 0
    st ← 1, end ← 1, localSt ← 0

    for i ← 0 to length[V]
        tempSum ← tempSum + V[i]
        if maxSum < tempSum
            st ← localSt
            end ← i
            maxSum ← tempSum
        end if
        if tempSum < 0
            localSt ← i+1
            tempSum ← 0
        end if
    end
vector<int> res ← {st, end, maxSum}
return res
```

DYNAMIC PROGRAMMING PSEUDO CODE

```
function MAIN()
    r ← 0, c ← 0, maxSum ← INT_MIN
    x ← 0, y ← 0, z ← 0, w ← 0
    input r, c


    for i ← 0 to r-1
        for j ← 0 to c-1
            input A[i][j]
        end
    end
```

```
for i ← 0 to r-1
        vector<int> sum(c)
        for j ← 0 to r-1
            for col ← 0 to c-1
                sum ← sum + A[j][col]
            end

            vector<int> res ← KADANE(sum)
            if maxSum < res[2]
                x ← i
                y ← res[0]
                z ← j
                w ← res[1]
                maxSum = res[2]
            end if
        end
    end

    print x, y, z, w, maxSum
```

ALGORITHM ANALYSIS
Brute force(Naive method)

In the brute force approach we try to check every possible rectangle in the given n X m 2D array(where n,m are number of rows and columns respectively).

This solution requires 6 nested loops :

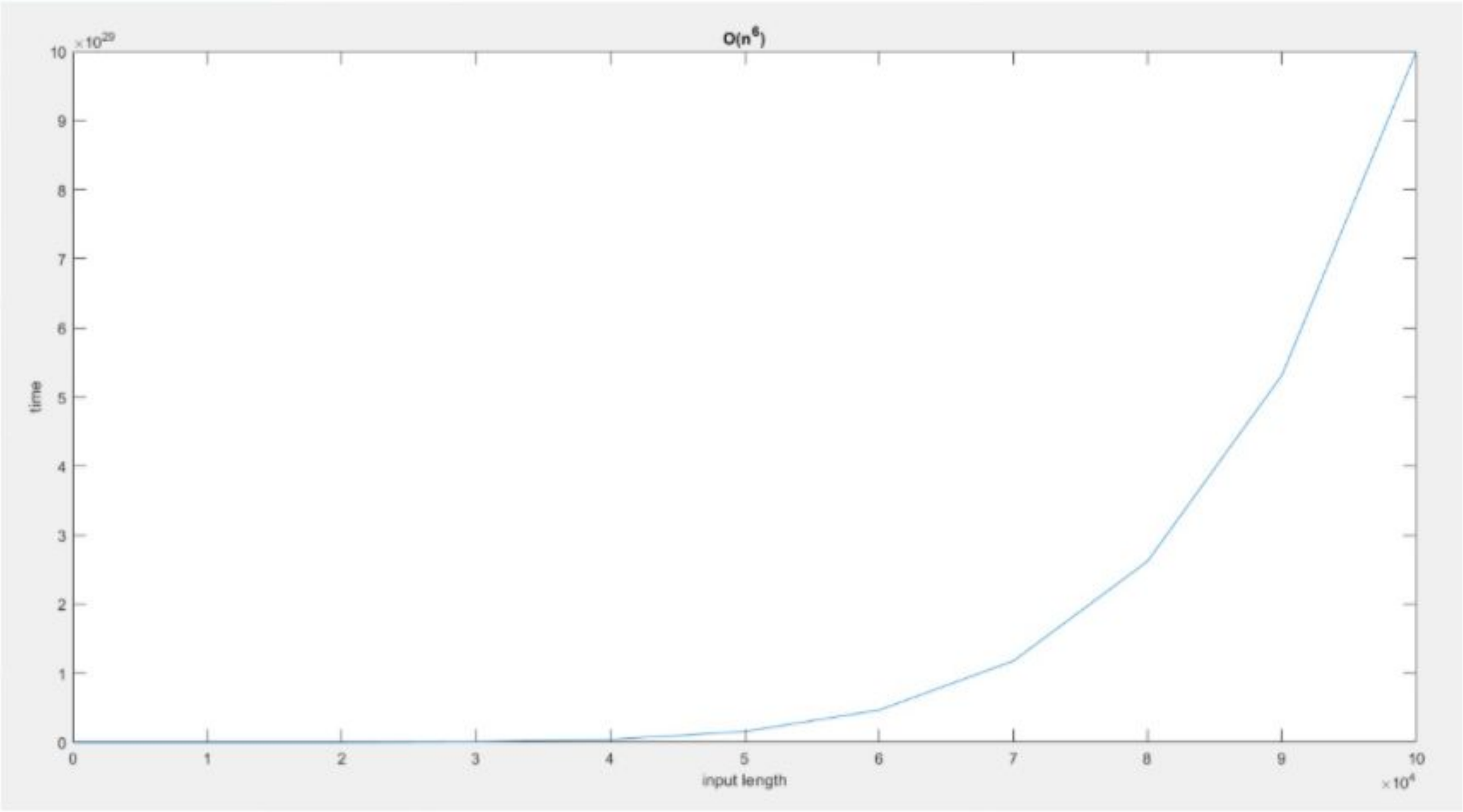
- 4 for start and end coordinate of the 2 axis $O(n^2 \times m^2)$
- 2 for the summation of the sub-matrix $O(n \times m)$

- The overall time complexity is **$O(n^3 \times m^3)$**
- As we did not allocate extra space in this approach, space complexity is $O(1)$.

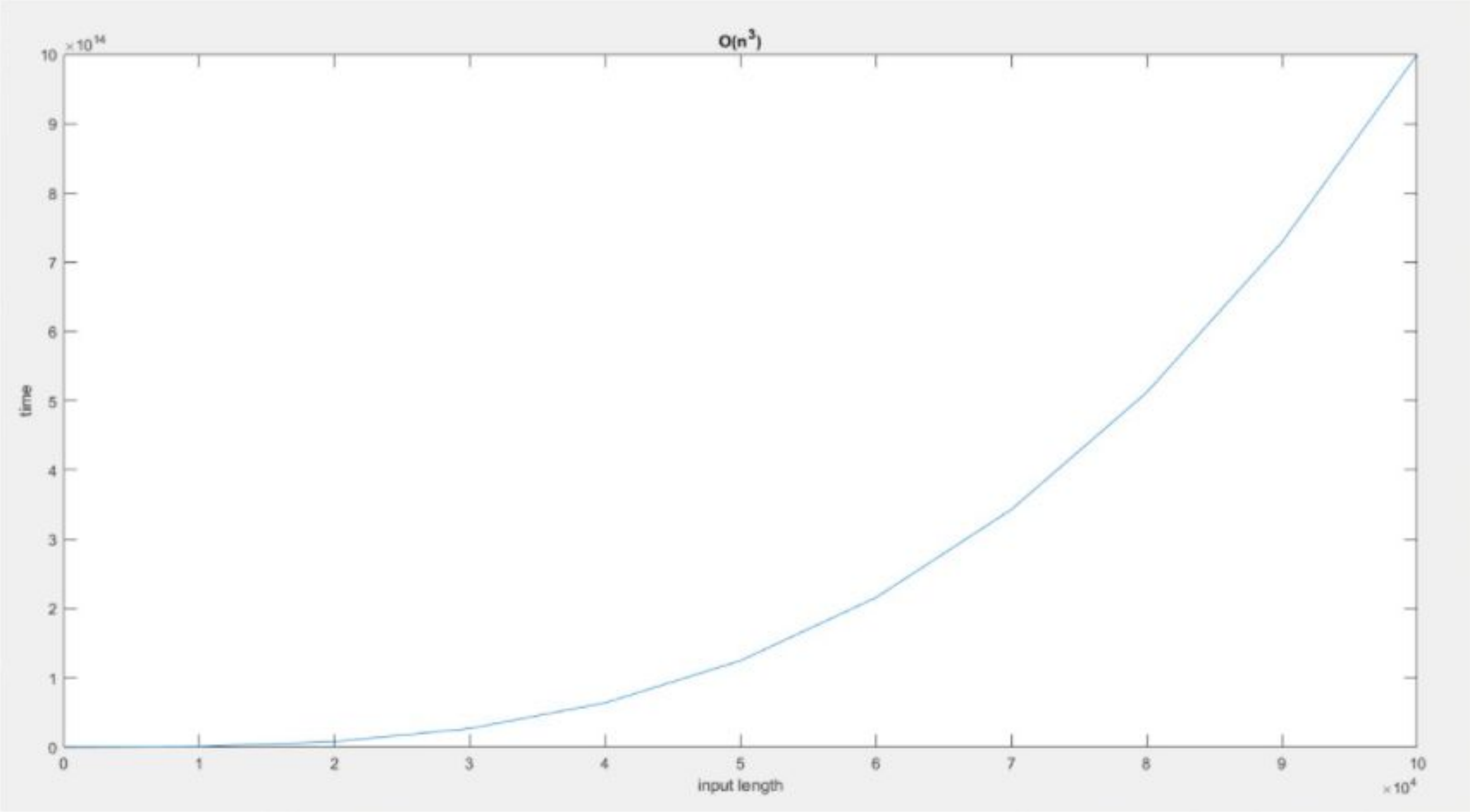
ALGORITHM ANALYSIS
DYNAMIC PROGRAMMING APPROACH

- Basically, kadane's dynamic algorithm (complexity: O(m) since we are doing it for column size) is used inside two loops of size n (complexity: $O(n^2)$).
- This gives a total complexity of $O(n^2 \times m)$
- In this approach we have used temp array, where it’s size is equal to number of columns.Therefore space complexity is O(m)

EXPERIMENTAL STUDY (NAIVE APPROACH)



EXPERIMENTAL STUDY(DP APPROACH)



CONCLUSION

From the experimental study we concluded that the average running time of dynamic approach using kadane’s algorithm is best, which can be observed from the mutual graph of kadane’s dynamic programming algorithm and Brute Force algorithm as shown.

IMPORTANT LINKS

Reference links:-

https://www.geeksforgeeks.org/maximum-sum-rectangle-in-a-2d-matrix-dp-27/

https://www.geeksforgeeks.org/largest-sum-contiguous-subarray/

Code link:-

Kadane’s Dynamic programming approach: https://ideone.com/Zw0Ape

Brute force:https://ideone.com/W0rVpg

THANK YOU