Matric Number: A0093896H
Name: Desmond Ang

Description of Algorithm
The algorithm that I employ is based on the Kadane's algorithm. Kadane's algorithm is used to find the maximum subarray of an 1D array and it's time complexity is O(n)

We can extend it to 2D by fixing a pair of left and right columns for every combination of left column and right column. Once we do that, we go to every row and sum the elements from the fixed left and right columns and store the sum into a temp array. By doing so, we have reduced the problem to 1D and can use the Kadane1D to solve it.

Similarly, we can extend the 2D version to the 3D version by fixing a top and bottom layer for every combination of top and bottom layers and reduce the problem to 2D version which uses Kadane2D to solve it.

Pseudo-code

```
KADANE-3D
// Input: 3D cuboid A[0..n-1, 0..n-1, 0..n-1]
highest ← 0
for i ← 0 to n-1 do
   temp ← A[0..n-1, 0..n-1]
   for j ← i to n-1 do
      for x ← 0 to n-1 do
         for y ← 0 to n-1 do
            // compress the layers between layer i and layer j into 1 layer
            temp[x][y] += cuboid[x][y][d]
         od
      od
      current ← KADANE-2D(temp)
      if current > highest then
          highest = current
   od
od
return highest

KADANE-2D
// Input: 2D matrix A[0..n-1, 0..n-1]
highest ← 0
for i ← 0 to n-1 do
   temp ← A[0..n-1]
   for j ← i to n-1 do
      for x ← 0 to n-1 do
          // compress the rows between column i and column j into 1 row
          temp[x] += cuboid[x][j]
      od
      current ← KADANE(temp)
```

```
        if current > highest then
            highest = current
    od
od
return highest
```

Correctness Proof

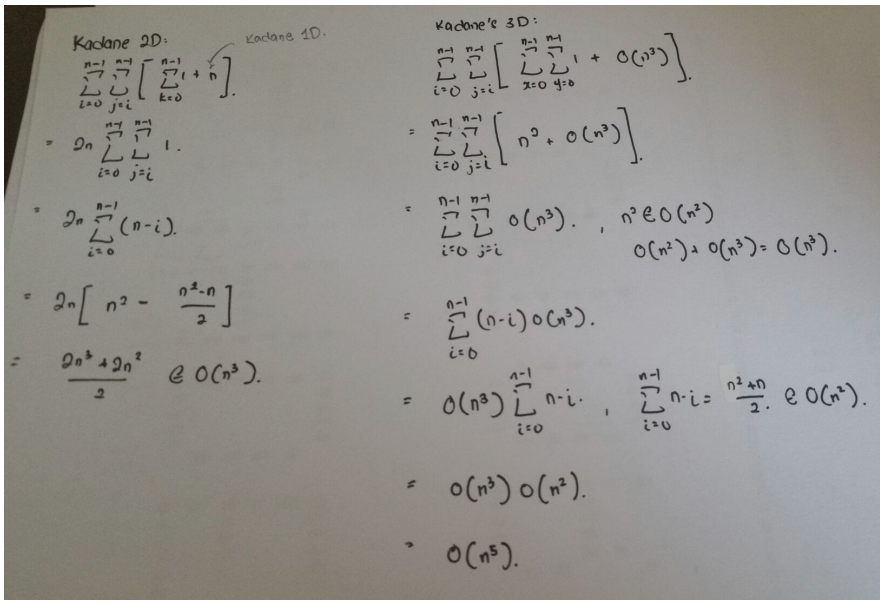
We have to find the sub-cuboid with the largest sum given the input cuboid. Therefore, a correct way to solve this problem is to find a way to construct all the possible sub-cuboids and compare them one by one to see which one is the highest value.

However, a more efficient way will be to realise that the maximum subarray will be positive contiguous. Hence, we can essentially transverse through the array only once and keep track of the highest positive contiguous subarray values and update it if it is increasing and positive using Kadane's algorithm. This gives us the maximum sub-array sum.

Because the problem is of 3D, we need to convert it to 2D where we find the maximum rectangle in the matrix and then to find the maximum rectangle, we convert the 2D matrix to a 1D array where the Kadane's algorithm is meant for.

Time complexity Analysis

Suppose we define a single addition as the basic operation then:
The time complexity for Kadane1D is O(n) since the array is only traversed once.
The time complexity for Kadane2D will be $O(n^3)$
Finally, the time complexity for the Kadane3D will be $O(n^5)$



Space complexity Analysis

Space complexity for Kadane2D is $O(n^2)$ to store the temp matrix.
Space complexity for Kadane1D is O(n) to store the temp array.
Hence overall space complexity is $O(n^2)$