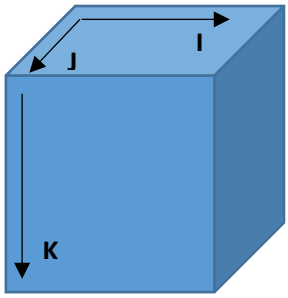
Name: Mudit Gupta
Student Number: A0153196B

CS3230 Assignment 1
Galactus Problem

Description of Algorithm: The task at hand is to find the cuboid with the maximum sum. The naïve approach is to calculate the sum of all possible cuboids in the cube and find the maximum one. In order to be more efficient we can make use of **Dynamic programming** and **Memoization**.

Pseudocode:

```
1.  For i = 1 to n                  // these 3 nested for loops calculate the power of all empires with
2.      For j = 1 to n              // origin as top left corner
3.          For k = 1 to n
4.              If (i > 0)  then    memo[i][j][k] += memo[i-1][j][k]
5.              If (j > 0)  then    memo[i][j][k] += memo[i][j-1][k]
6.              If (k > 0)  then    memo[i][j][k] += memo[i][j][k-1]
7.              If (i > 0 and j > 0)  then  memo[i][j][k] – = memo[i-1][j-1][k]
8.              If (j > 0 and k > 0) then memo[i][j][k] – = memo[i][j-1][k-1]
9.              If (i > 0 and k > 0) then memo[i][j][k] – = memo[i-1][j][k-1]
10.             If (i > 0 and j > 0 and k > 0) then memo[i][j][k] += memo[i-1][j-1][k-1]
11.  // The next 6 nested for loops examine every possible cuboid and calculate its sum in O(1)
12.  // time using the cumulative memo array calculated above
13. For i = 1 to n
14.     For j = 1 to n
15.         For k = 1 to n
16.             For i_s  = 1 to i
17.                 For j_s  = 1 to j
18.                     For k_s  = 1 to k
19.                         score = memo[i][j][k]
20.                         If (i_s > 0)   then   score – = memo[i_s - 1][j][k]
21.                         If (j_s > 0)   then   score – = memo[i][j_s - 1][k]
22.                         If (k_s > 0)  then   score – = memo[i][j][k_s - 1]
23.                         If (i_s > 0 and j_s > 0)  then  score  + = memo[i_s - 1][ j_s - 1][k]
24.                         If (j_s > 0 and k_s > 0) then  score + = memo[i][j_s - 1][k_s - 1]
25.                         If (i_s > 0 and k_s > 0) then  score + = memo[i_s - 1][j][k_s - 1]
26.                         If (i_s > 0 and j_s > 0 and k_s > 0)  then
27.                             score – = memo[i_s - 1][j_s - 1][k_s - 1]
28.                         If ( score > max_score )    // Update max_score in every iteration
29.                             max_score  = score
```



First we store the power of each cube in a **3D array (memo)**. The index directions are shown above.

- In **lines 1 – 10** we calculate the total sum (power) of every cuboid whose **top left corner (starting point)** is **origin (0,0,0)** and **bottom right corner (ending point)** is **(i , j , k)**
- Now we have the sum of all cuboids whose starting point is origin. In **lines 13-27** we take all possible combinations of starting points (i_s , j_s , k_s) and ending points (i , j , k).
- For a particular value of (i , j , k) the value in memo[i][j][k] is higher since our **starting points (i_s , j_s , k_s)** may not be the origin this time. Thus, we need to subtract the volumes not belonging to the cuboid currently under consideration. In **lines 20-22 and 27 we subtract the excluded volumes**.
- We also need to **add the overlapping volumes in lines 23-25** to ensure they are not subtracted twice.

Name: Mudit Gupta
Student Number: A0153196B

- Finally **lines 28-29 update the max_score encountered so far** in every iteration. Once the algorithm ends the **max_score is the required answer.**

Correctness Proof: This section provides a proof sketch for the algorithm. The ultimate aim is finding the cuboid with the largest power.

Lines 1-10:

We start off by building a **cumulative sum table (memo)** where the value of the array element memo[i][j][k] is the power of the empire whose **starting point (top left corner) is origin** and **ending point (bottom right corner).** We use **3 nested for loops** to account for every possible ending point
Let's **represent a cuboid as (starting point , ending point).** The correctness of cumulative sum table is discussed below:

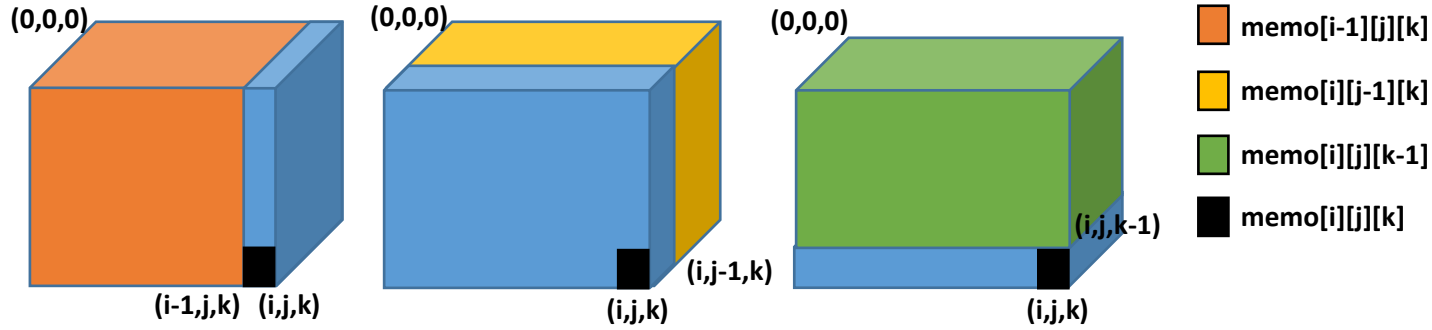
Base Case (i = j = k = 0): This is a cuboid starting and ending at origin which is basically the cube at origin.
Thus, **memo[0][0][0] = power of cube at origin**

Inductive Case:

Consider a general table index (i , j , k). We **assume that** $\forall\ \mathbf{0} \le \boldsymbol{i'} < \boldsymbol{i}\ , \mathbf{0} \le \boldsymbol{j'} < \boldsymbol{j}\ , \mathbf{0} \le \boldsymbol{k'} < \boldsymbol{k}$

$$\boldsymbol{memo[i'][j'][k'] = power\ of\ cuboid}\ (\ (\mathbf{0},\mathbf{0},\mathbf{0}), (\boldsymbol{i'}, \boldsymbol{j'}, \boldsymbol{k'})\ )$$

The cuboid ((0,0,0) , (i,j,k)) is shown below:



From the figure above it is clear $memo[i][j][k]$ can be obtained by summing $memo[i-1][j][k]$, $memo[i][j-1][k]$ and $memo[i][j][k-1]$. We also need to subtract the overlapping areas $memo[i-1][j-1][k]$, $memo[i][j-1][k-1]$ and $memo[i-1][j][k-1]$

$$\begin{aligned} memo[i][j][k] = {} & cubevalue(i,j,k) + memo[i-1][j][k] + memo[i][j-1][k] + memo[i][j][k-1] \\ & - memo[i-1][j-1][k] - memo[i][j-1][k-1] - memo[i-1][j][k-1] \\ & + memo[i-1][j-1][k-1] \end{aligned}$$

$$\boldsymbol{where}: \mathbf{0} < \boldsymbol{i,j,k} \le \boldsymbol{n}$$

Thus, $memo[i][j][k]$ correctly calculates power of cuboid ((0,0,0) , (i,j,k)). By **principle of induction memo table correctly calculates power of all cuboids with end points ranging from (0,0,0) to (n,n,n) and start point fixed at origin**.

Lines 11-29:

Now that we have the power of all cuboids which start at (0,0,0) **we need to calculate power of cuboids with starting points other than (0,0,0)**. For this we use **6 nested for loops**, 3 for all possible starting points and another 3 for all possible ending points.

Consider a general case where we have a cuboid $(\ (i_s\ , j_s\ , k_s), (i,j,k)\ )$. In our memo table that we built above $memo[i][j][k]$ is the power of cube starting from origin. But in this case **we don't need to include the powers of units cubes whose any index is less than $(\boldsymbol{i_s}\ , \boldsymbol{j_s}\ , \boldsymbol{k_s})$.**

Name: Mudit Gupta
Student Number: A0153196B

By **Inclusion-Exclusion principle** we can calculate score of cuboid ($(i_s, j_s, k_s), (i, j, k)$) by excluding all areas which lie outside our cuboid. Again we need to make sure overlapping areas are not subtracted twice. Thus, to get the score for the current cuboid

$$\begin{aligned} score = {} & memo[i][j][k] - memo[i_s - 1][j][k] - memo[i][j_s - 1][k] - memo[i][j][k_s - 1] \\ & + memo[i_s - 1][j_s - 1][k] + memo[i][j_s - 1][k_s - 1] + memo[i_s - 1][j][k_s - 1] \\ & - memo[i_s - 1][j_s - 1][k_s - 1] \end{aligned}$$

$$\boldsymbol{where}: \quad \mathbf{0} < \boldsymbol{i_s} \leq \boldsymbol{i} \leq \boldsymbol{n} \; and \; \mathbf{0} < \boldsymbol{j_s} \leq \boldsymbol{j} \leq \boldsymbol{n} \; and \; \mathbf{0} < \boldsymbol{k_s} \leq \boldsymbol{k} \leq \boldsymbol{n}$$

In every iteration of the for loop we compute the score for the current cuboid and update the max_score variable (lines 28-29) which keeps track of the maximum power encountered so far.
Once the algorithm ends **max_score is the required answer** and contains the cosmic power of the strongest empire.

Time Complexity:

Let the time taken by the **algorithm be** $(\boldsymbol{n})$, by **lines 1-10 be** $\boldsymbol{T_1(n)}$ and **lines 11-29 be** $\boldsymbol{T_2(n)}$

Lines 1-10:

In lines 1-10 we have 3 nested for loops iterating from 1 to n each. **Each iteration takes constant time** since we perform 4 additions and 3 subtractions. Previous array elements used in the calculation have already been computed in bottom-up fashion. Thus the time taken is:

$$T_1(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} O(1) = n^3 * O(1) = O(n^3)$$

Lines 11-29:

In lines 11-29 we have 6 nested for loops the first 3 iterating from 1 to n. The last 3 iterating from 1 to the current value of their respective counterpart. This basically implies that in the first three loops we fix an index (i, j, k) and then explore all other indexes less than or equal to (i, j, k).
For each iteration it is important to note here that **memoization** via the **cumulative sum table** we built in lines 1-10 **ensures that the score for each cuboid is calculated in constant time.** Since we already have the solution of subproblems stored in the memo table, we don't need to recalculate them. In short, we do 3 additions and 4 subtractions using already computed elements which takes constant time.

$$T_2(n) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n}\sum_{i_s=1}^{i}\sum_{j_s=1}^{j}\sum_{k_s=1}^{k} O(1) = \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{n} i * j * k = \frac{n^3(n+1)^3 O(1)}{8}$$

$$T_2(n) = \frac{n^3(n^3 + 3n^2 + 3n + 1)O(1)}{8} = \frac{(n^6 + 3n^5 + 3n^4 + n^3)O(1)}{8} = O(n^6)$$

The total time taken $T(n)$ is:

$$T(n) = T_1(n) + T_2(n) = O(n^3) + O(n^6) = O(n^6)$$

Thus, the overall time taken by the algorithm is $\boldsymbol{O(n^6)}$.

Space Complexity:

For memoization we use a **n x n x n table (memo) with $\boldsymbol{n^3}$ elements** (total number of unit cubes in galaxy of size length n). Apart from that we use 2 local variables to store score of current cuboid and maximum score seen so far.

$$Space\ Complexity = O(n^3)$$