**Professor: Amotz Bar-Noy**

**Course: Analysis of Algorithms**

**College: Brooklyn College (City University of New York)**

**Name: Kishan Shah**

**Id: 23970203**

**Matrix Multiplication**

**Objective:** Assume n = 2^k is a power of 2 for some k >= 0. Let A and B be two n \* n matrices each containing n^2 positive integers. Code a program that outputs the product C = A \* B.

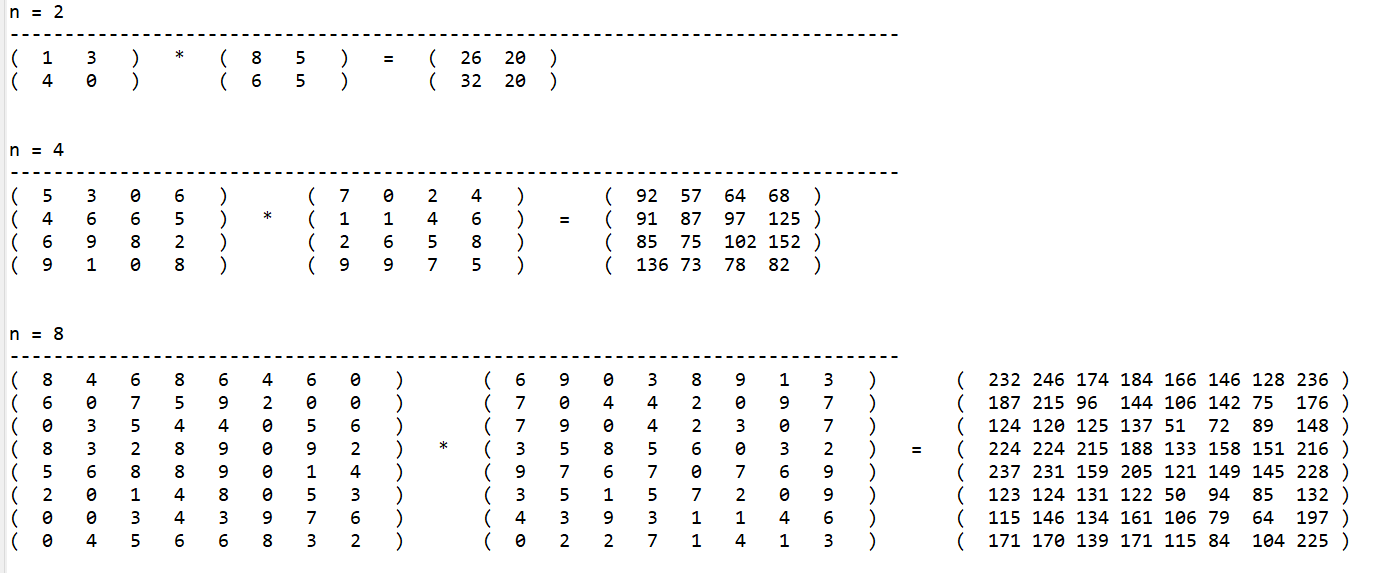
**Method:** Implement the direct Theta(n^3)-operations method and the Strassen Theta(n^log\_2 (7))-operations recursive algorithm.

**Output:** Display all three matrices as C = A \* B.

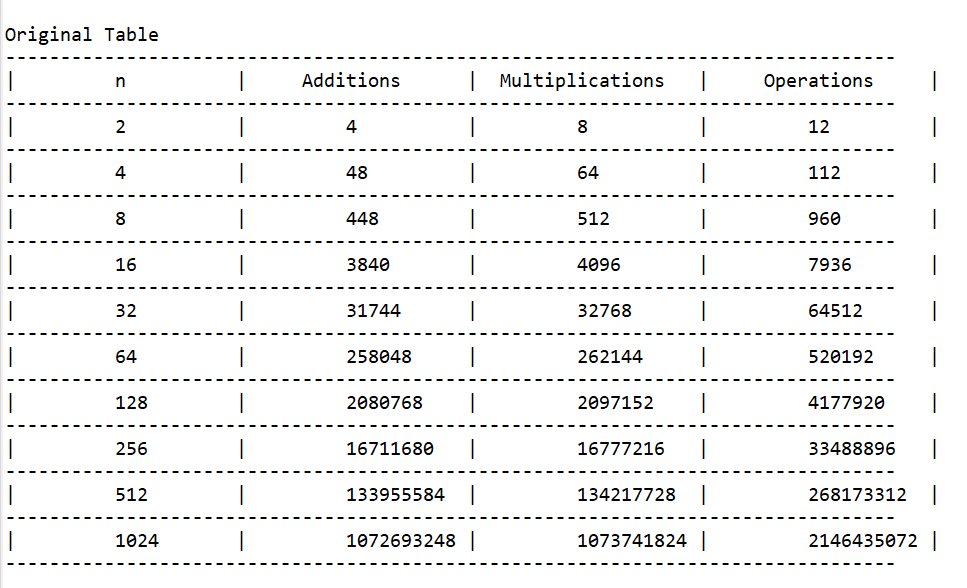
**Performance evaluation 1:** Count the exact number of operations (multiplications and additions and total)made by both implementations.

**My Outputs:**

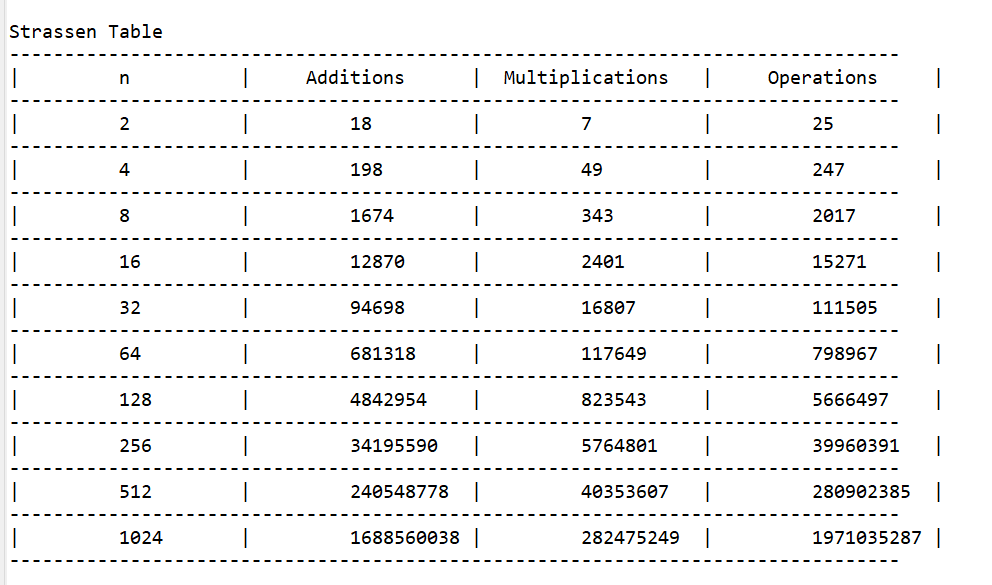
* This is the output for some small n in A \* B = C format, where A and B are input matrices and C is the output matrix.
* These outputs are same for both implementations.



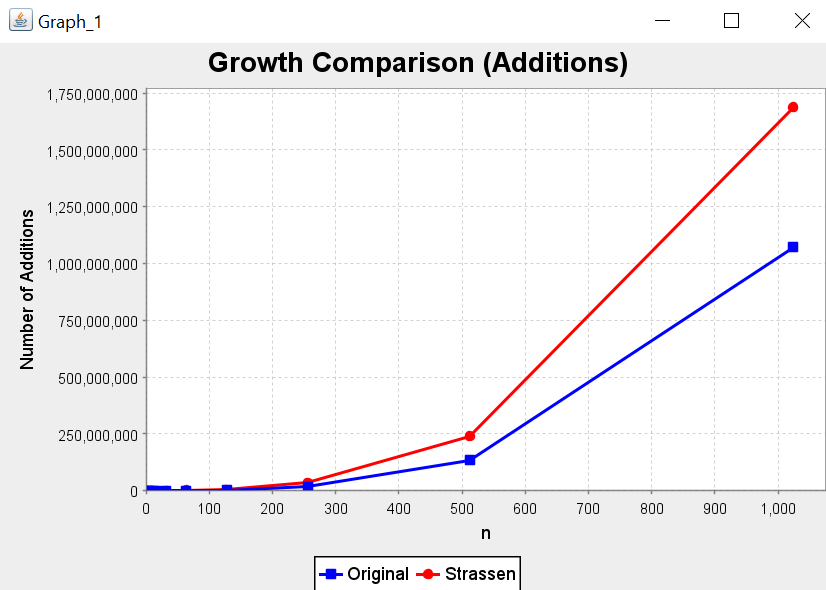
* Original (Direct) method’s count table.



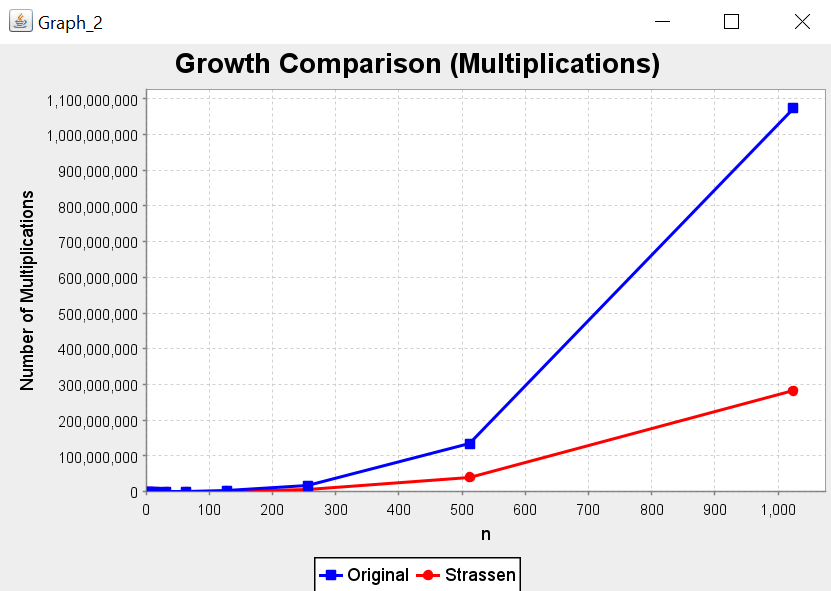
* Strassen method’s count table.



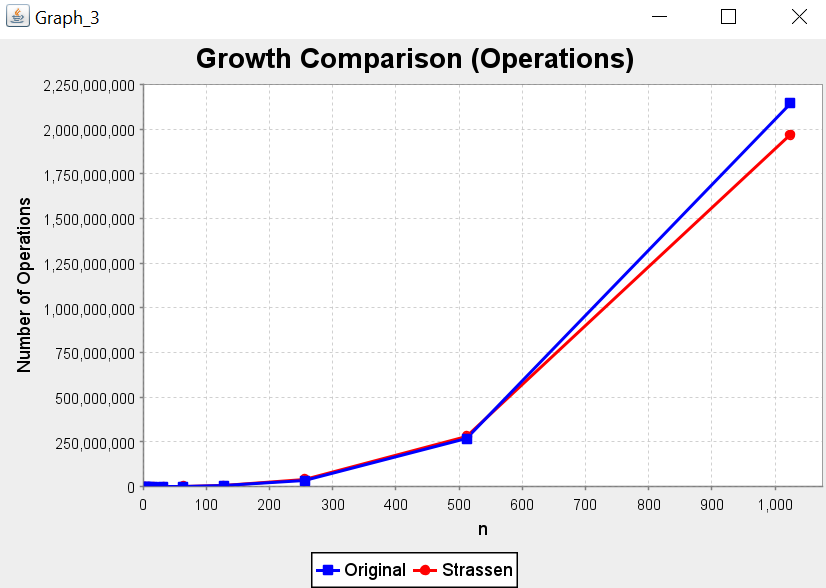
* Growth comparison on number of additions.



* Growth comparison on number of multiplications.



* Growth comparison on number of total operations.



So, from above results, we can say that

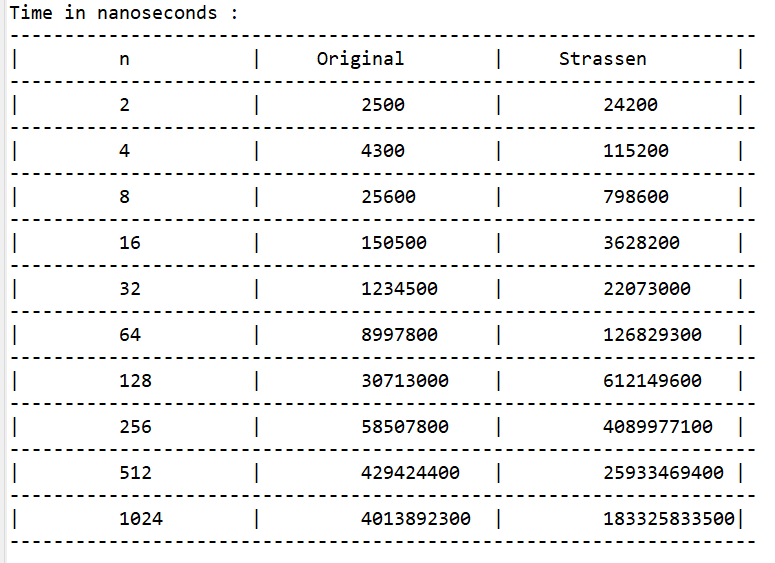
* Number of additions will remain larger for Strassen method than Original method.
* Number of multiplications will remain significantly lower for larger n in Strassen method than Original (direct) method.
* Total number of operations will remain slightly closer to each other.
* But from these tables and graphs, we can see the shift in total number of comparisons.
* Till (n=512), Strassen method does more total number of operations than Direct method. But when n becomes 1024, Direct method does a greater number of total operations.

------------------------------------------------------------------------------------------------------------------------------------------

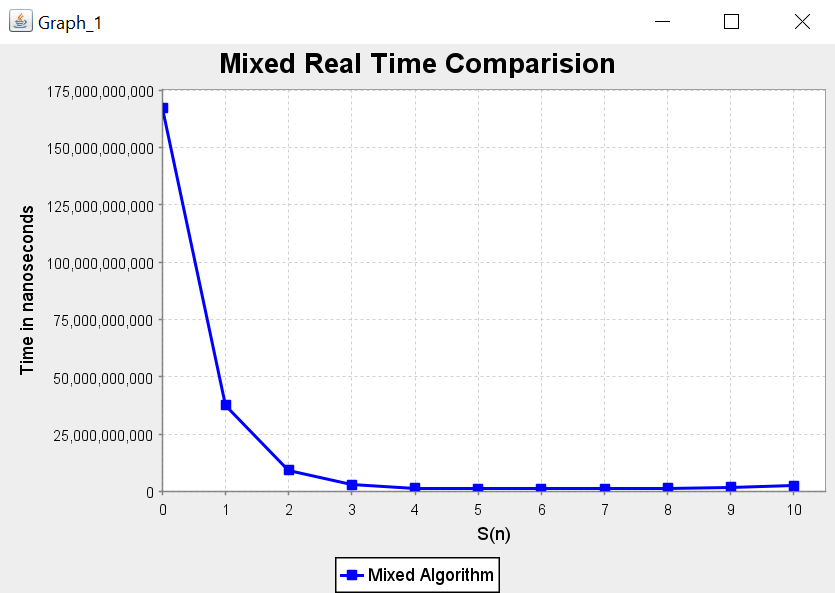
**Performance evaluation 2:** Measure the “real time” it takes both methods to multiply two all-ones matrices.

**My Outputs:**

* Time here is in nanoseconds. For larger n (n > 1024), my system takes a very long time to compute in Strassen method.
* But we know that for a very large n, Strassen method will perform better than Direct method.
* Because for small n, Strassen method has to come down to the level-1 by calling itself recursively and each iteration has many functions to do.
* But for larger n, these recursive levels will become comparatively small as compare to n and we can have a better output time in Strassen method than Direct method.
* Table for Growth comparison on real time taken by these two implementations.



* Graph on Growth comparison on real time taken by both implementations.

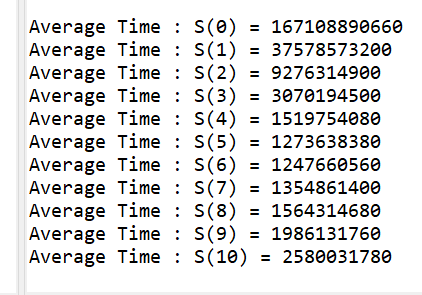


------------------------------------------------------------------------------------------------------------------------------------------

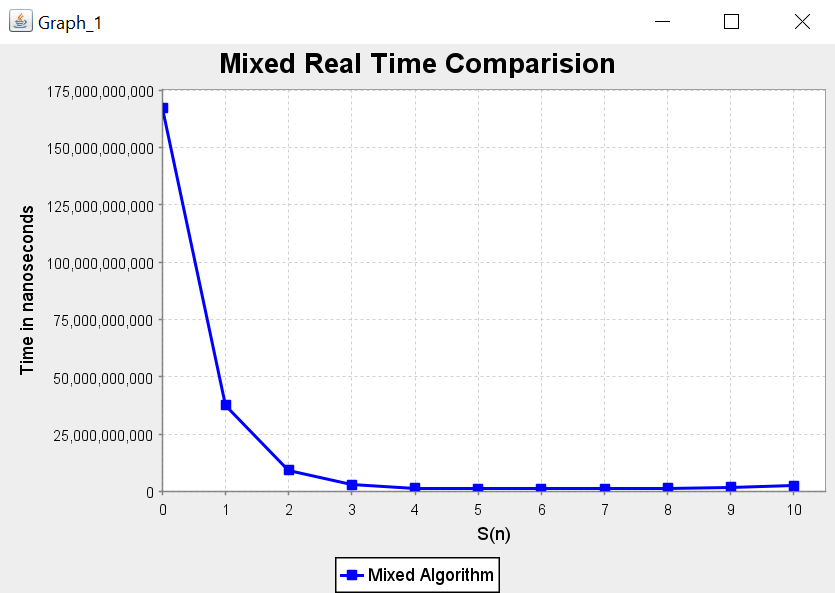
**Extra 1:** Implement a “hybrid” algorithm that stops the recursion once the sizes of the matrices reach some specific ‘n’ and then apply the direct method.

**My Outputs:**

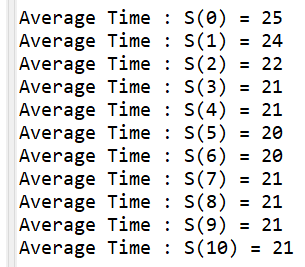
* For this implementation, I am taking (n = 1024).
* In this hybrid implementation, my program will run 11 different algorithms.
* In which, S(k) means when we reach n = 2^k while doing recursive calls, now onwards we apply Direct method on remaining small part of the matrices.
* For example, S(4) means at first, we use Strassen method on (n = 1024) and keep using Strassen method until we reach on (n = 2^4). Now onwards, we use Direct method to solve remaining part of the matrices.
* For better and clear comparison, I am running each of these 11 different algorithms for 50 times and then I am taking the average of each of them.
* Table for comparison on real-time taken by these 11 hybrid algorithms.



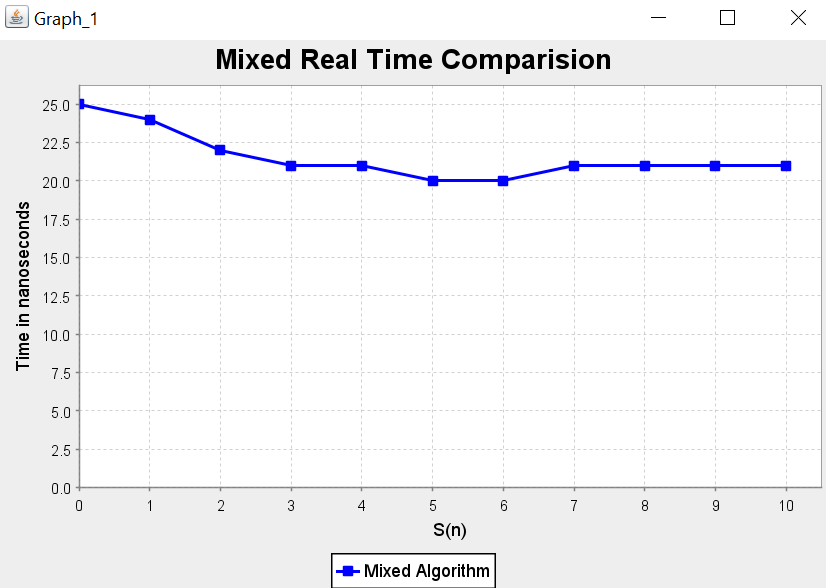
* According to the results of that, we can say that for (n = 1024), S(6) will perform best. Means if we recursively compute using Strassen method till (n = 2^7), and then from (n = 2^6) if we use Direct method, then we will have our best minimum output time.
* Graph of hybrid comparison having time in nanoseconds.



* Table of hybrid comparison having time in log base scale.



* Graph of hybrid comparison having time in log base scale.



------------------------------------------------------------------------------------------------------------------------------------------

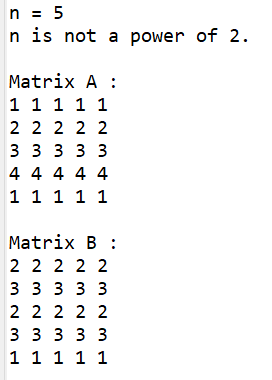
**Extra 2**: Implement the Strassen recursive algorithm for any value of n >= 1.

**My Outputs:**

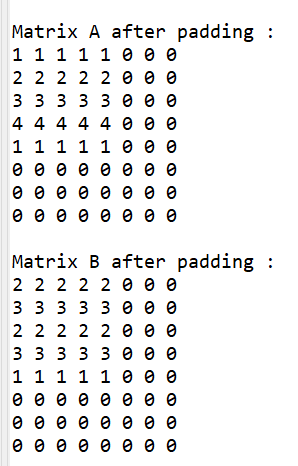
* As we know that general Strassen method is for (n = 2^k), where k >= 0 and k is an integer.
* But we can use this method for any value of n.
* We just need to pad remaining part of the matrix.
* For this, first we need to check if n is power of 2 or not.
* If n is not a power of 2, then we need to find the next power of 2 for that n.
* For example, if we have (n = 5), then at first, we know that n is not a power of 2. Now, we find the next power of 2 for (n = 5), which is 8. Now, we have input data in first 5 rows and columns.

To make this matrix from (5 \* 5) to (8 \* 8), we put zeros in 6th, 7th, and 8th rows and columns of the matrix. This is called padding a matrix.

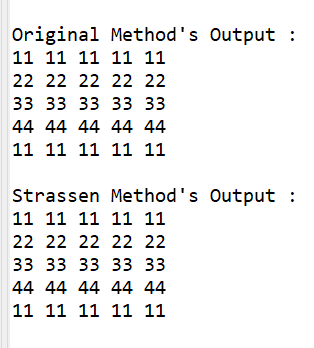
* For the demonstration, I used 3 input files for n = 5, 7, and 9.
* Below is the sample output for (n = 5).
* Matrices A and B are inputs matrices.



* Matrices A and B after padding.



* Output matrix using both Implementations remains same.



------------------------------------------------------------------------------------------------------------------------------------------