Programming Assignment 1 (Due: 2 Mar 2021, 11.59pm)

For this programming assignment, you will not have to submit any written answers (except for the challenge). Instead, you will have to submit a program written in Java or C++11 on CodeCrunch at `https://codecrunch.comp.nus.edu.sg/`. The portal will stop accepting submissions after 2 Mar 2021 2359h so please start your assignment **early**.

Templates will be provided for all the problems. These templates provide a starting point for your implementation. It is highly recommended (but not required) to use all the templates given to you. **Do not change the file name or the class name of the template, or else your code may be marked as incorrect**. You have to submit your own work. **Posting the question or solution in public repositories are not allowed also counts as a form of plagiarism.** Any form of plagiarism is subject to disciplinary action.

You will not be graded for programming style. However, you may be asked to explain your code if the marker cannot understand it. Marks will be deducted if there are bugs in your code or if your algorithm does not meet the time complexity stated in the question. You are strongly encouraged to design your own test cases to test your code.

If you need any clarification on any of the questions, do post on the LumiNUS forum so that everyone can see your responses.

For queries related to task 1, contact Rasnayaka Mudiyanselage Sanka Nuwan Bandara Rasnayaka (e0147061@u.nus.edu). For queries related to task 2, contact Ling Yan Hao (e0174827@u.nus.edu).

Note: Passing all the test cases on CodeCrunch does not guarantee that you will get full credit for the assignment. We may add additional test cases while grading your solutions.

1 Task 1 (2 marks)

1.1 Statement

To multiply two n by n matrices, Strassen's matrix multiplication runs in $\mathcal{O}(n^{\log_2 7})$ time, which is faster than the naive algorithm which runs in $\mathcal{O}(n^3)$ time.

However, asymptotic analysis does not take into account the constant factors. For small matrices, Strassen's algorithm can run significantly slower than the naive approach.

As such, one approach is to switch to naive matrix multiplication for small matrices and use Strassen's algorithm for large matrices. In particular, when we perform the recursive step, we check for the size of the matrix. When the matrix is sufficiently small, we switch to naive matrix multiplication instead of recursively doing Strassen's algorithm.

(Side remark: this paradigm of switching to an asymptotically slower algorithm for small inputs is a common theme. For example, $\mathcal{O}(n^2)$ sorting algorithms like insertion sort may perform faster than $\mathcal{O}(n \log n)$ algorithms like quick sort and merge sort for small inputs. As such, it is common to use insertion sort for small arrays and quick sort for large arrays. Another example of this paradigm comes from parallel programming where the overhead of spawning additional threads makes it more efficient to run sequential algorithms for small inputs.)

You are given two matrices A and B. Your task is to compute the matrix product AB using Strassen's algorithm with the above optimization. For this task, we recommend switching to naive matrix multiplication for matrices of size 64×64 or smaller, while using recursive algorithm for larger matrices.

1.2 Input format

The first line of the input consists of a single integer n, indicating that the matrix are $n \times n$.

After which n lines follow which represents the entries of A. In particular, the i-th line of these n lines will contain $A_{i,1}, A_{i,2}, \ldots, A_{i,n}$.

Following which, another n lines follow, indicating the entries of B.

1.3 Output format

Output n lines, each consisting of n integers, representing the matrix product AB.

1.4 Constraints

- $1 \leq n \leq 1024$
- n is a power of 2
- All entries of the matrix A and B are between -100 and 100 inclusive. Note that this does not mean that the correct output AB will have entries between -100 and 100.

1.5 Scoring

Your solution should terminate in 20 seconds for C++ or 30 seconds for Java.

Note that your solution will still be manually graded and should properly implement Strassen's algorithm. Solutions which pass the sample test case using other techniques will not be given credit.

1.6 Samples

Sample input

```
2
1 4
3 2
7 5
9 8
```

Sample output

```
43 37
39 31
```

2 Task 2 (6 marks + 1 mark challenge)

2.1 Statement

You have a rectangular-shaped cake of size r by c. On this cake, there are n rectangular-shaped toppings. The sides of these rectangles are parallel to the sides of the cake.

You are allowed to cut the cake into several pieces according to the following rules:

1. Every cut must be either a vertical line or a horizontal line. No diagonal cuts are allowed.
2. Every cut must increase the number of pieces by 1. Partial cuts (refer to Figure 2) are not allowed.
3. The cuts cannot break up the toppings (i.e. each topping should remain as a single piece). However, cuts along the boundary of the toppings are allowed.
4. At the end of all the cuts, every piece should have at least 1 topping.

Find the maximum number of pieces you can cut the cake into which follows these rules.

2.2 Input format

The first line of the input contains 3 space-separated integers r, c, n. After which, there are n lines. Each of these n lines contains 4 space-separated integers x_1, y_1, x_2, y_2 ($0 \leq x_1 < x_2 \leq r, 0 \leq y_1 < y_2 \leq c$) , indicating that there is a single topping on the rectangular region $x_1 < x < x_2, y_1 < y < y_2$.

2.3 Output format

Output a single integer- the maximum number of pieces you can cut the cake into.

2.4 Constraints

- $1 \leq n \leq 5000$
- $1 \leq r, c \leq 10^9$
- The region occupied by different toppings are disjoint (i.e. do not overlap).

2.5 Scoring

Full credit (6 marks) will be given for solutions which run in $\mathcal{O}(n^2 \log n)$ time. Solutions which run in $\mathcal{O}(n^3)$ time will be given a partial credit of 4 marks. Solutions in C++ should terminate in 1 second and solutions in Java should terminate in 2 seconds on CodeCrunch.

2.5.1 Challenge (1 mark bonus)

Our best solution runs in $\mathcal{O}(n \log n)$ time. Students who can find this solution will be awarded 1 mark extra credit. To score this 1 mark, submit a written description of your algorithm together with a time complexity analysis and email it to `e0174827@u.nus.edu` with the subject 'CS3230 PA1 Bonus' before the deadline. There will be no test cases provided for the challenge.

2.6 Samples

Refer to figures 5 and 6 for explanation for the samples.

Sample input 1

```
3 3 4
0 0 2 1
2 0 3 2
1 2 3 3
0 1 1 3
```

Sample output 1

```
1
```

Sample input 2

```
3 3 4
0 0 2 1
2 0 3 1
1 2 3 3
0 1 1 3
```

Sample output 2

```
4
```

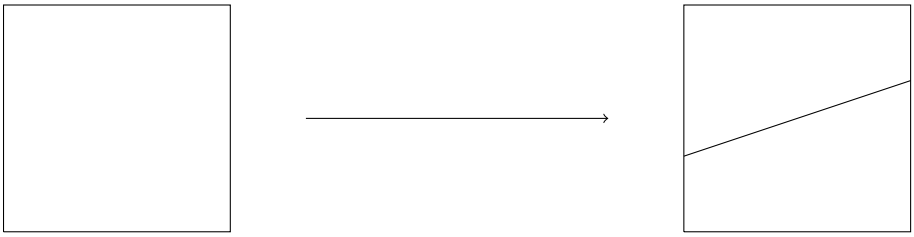

Figure 1: This cut is not allowed as the line is not parallel to the sides of the cake.

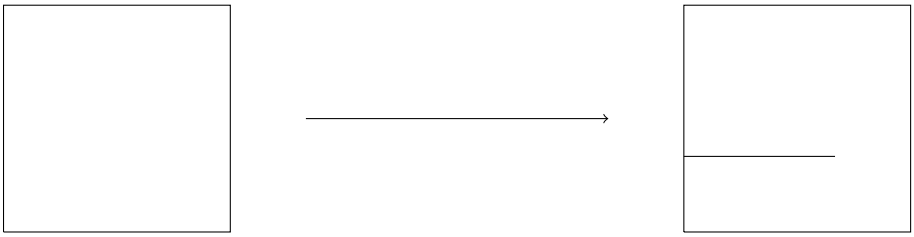

Figure 2: This is a partial cut and thus it is not allowed.

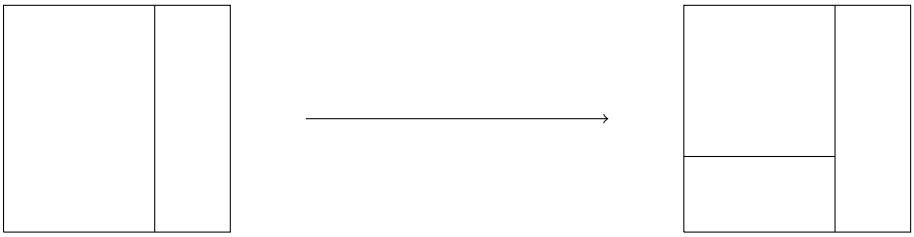

Figure 3: Although this cut does not go through the whole cake, it is allowed as it increases the number of pieces.

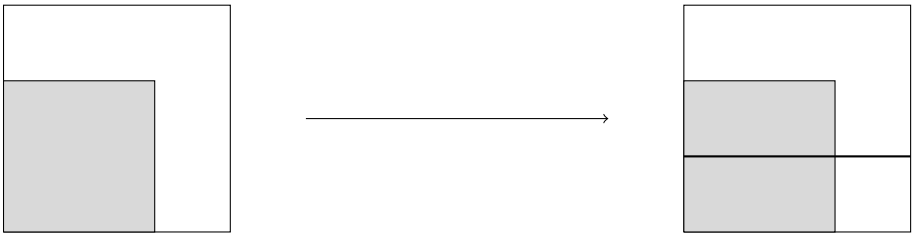

Figure 4: This is a disallowed cut as it goes cuts the topping (the shaded region) into two parts.

4	3	3
4		2
1	1	2

Figure 5: Sample input 1: We are unable to make any cuts which satisfies the rules. Therefore, the answer is 1.

4	3	3
4		
1	1	2

Figure 6: Sample input 2: By making the horizontal cut before the vertical cuts, it is possible to make 4 cuts such that the 4 toppings are in different parts.