Full Name:	Hideki Ikeda
Email:	hidekiai+hackerrank@gmail.com
Test Name:	**Mock Test**
Taken On:	11 Sep 2023 02:06:18 IST
Time Taken:	23 min 51 sec/ 24 min
Linkedin:	https://www.linkedin.com/in/hidekiai/
Invited by:	Ankush
Invited on:	11 Sep 2023 02:06:07 IST
Skills Score:	
Tags Score:	Algorithms 0/90 Constructive Algorithms 0/90 Core CS 0/90 Greedy Algorithms 0/90 Medium 0/90 Problem Solving 0/90 problem-solving 0/90

0%

0/90

scored in **Mock Test** in 23 min 51 sec on 11 Sep 2023 02:06:18 IST

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	Flipping the Matrix > **Coding**	24 min 13 sec	0/ 90	⊗

QUESTION 1

⊗ Wrong Answer

Score 0

Flipping the Matrix > Coding

Algorithms | Medium | Greedy Algorithms | Constructive Algorithms | problem-solving | Core CS | Problem Solving

QUESTION DESCRIPTION

Sean invented a game involving a $2n \times 2n$ matrix where each cell of the matrix contains an integer. He can reverse any of its rows or columns any number of times. The goal of the game is to maximize the sum of the elements in the $n \times n$ submatrix located in the upper-left quadrant of the matrix.

Given the initial configurations for q matrices, help Sean reverse the rows and columns of each matrix in the best possible way so that the sum of the elements in the matrix's upper-left quadrant is maximal.

Example

$matrix = [[1, 2], [3, 4]]$

```
1 2
3 4
```

It is 2×2 and we want to maximize the top left quadrant, a 1×1 matrix. Reverse row 1:

```
1 2
4 3
```

And now reverse column 0:

```
4 2
1 3
```

The maximal sum is 4.

Function Description

Complete the *flippingMatrix* function in the editor below.

flippingMatrix has the following parameters:
- *int matrix[2n][2n]:* a 2-dimensional array of integers

Returns
- *int:* the maximum sum possible.

Input Format

The first line contains an integer q, the number of queries.

The next q sets of lines are in the following format:

- The first line of each query contains an integer, n.
- Each of the next $2n$ lines contains $2n$ space-separated integers $matrix[i][j]$ in row i of the matrix.

Constraints

- $1 \leq q \leq 16$
- $1 \leq n \leq 128$
- $0 \leq matrix[i][j] \leq 4096$, where $0 \leq i, j < 2n$.

Sample Input

```
STDIN           Function
-----           --------
1               q = 1
2               n = 2
112 42 83 119   matrix = [[112, 42, 83, 119], [56, 125, 56, 49], \
56 125 56 49              [15, 78, 101, 43], [62, 98, 114, 108]]
15 78 101 43
62 98 114 108
```

Sample Output

```
414
```

Explanation

Start out with the following $2n \times 2n$ matrix:

$$matrix = \begin{bmatrix} 112 & 42 & 83 & 119 \\ 56 & 125 & 56 & 49 \\ 15 & 78 & 101 & 43 \\ 62 & 98 & 114 & 108 \end{bmatrix}$$

Perform the following operations to maximize the sum of the $n \times n$ submatrix in the upper-left quadrant:

2. Reverse column 2 ($[83, 56, 101, 114] \rightarrow [114, 101, 56, 83]$), resulting in the matrix:

$$matrix = \begin{bmatrix} 112 & 42 & 114 & 119 \\ 56 & 125 & 101 & 49 \\ 15 & 78 & 56 & 43 \\ 62 & 98 & 83 & 108 \end{bmatrix}$$

3. Reverse row 0 ($[112, 42, 114, 119] \rightarrow [119, 114, 42, 112]$), resulting in the matrix:

$$matrix = \begin{bmatrix} 119 & 114 & 42 & 112 \\ 56 & 125 & 101 & 49 \\ 15 & 78 & 56 & 43 \\ 62 & 98 & 83 & 108 \end{bmatrix}$$

The sum of values in the $n \times n$ submatrix in the upper-left quadrant is $119 + 114 + 56 + 125 = 414$.

CANDIDATE ANSWER

The candidate did not manually submit any code. The last compiled version has been auto-submitted and the score you see below is for the auto-submitted version.

Language used: **C++14**

```
/*
 * Complete the 'flippingMatrix' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts 2D_INTEGER_ARRAY matrix as parameter.
 */


int flippingMatrix(vector<vector<int>> matrix) {
  auto dump_matrix = [](const vector<vector<int>> &matrix) {
      cout << "Matix dim: " << matrix.size() << "x" << matrix[0].size() <<
endl;
    for (auto row = 0; row < matrix.size(); ++row) {
      for (auto col = 0; col < matrix[0].size(); ++col) {
        cout << matrix[row][col] << " ";
      }
      cout << endl;
    }
  };

  // function to sum each quadrant for evaluations (returns tuple<long, long,
  // long, long>, input is matrix)
  auto sum_quadrants = [](const vector<vector<int>> &matrix) {
    const auto height = matrix.size();
    const auto mid_height = height / 2;
    const auto width = matrix[0].size();
    const auto mid_width = width / 2;
    long q1 = 0, q2 = 0, q3 = 0, q4 = 0;
    for (auto row = 0; row < mid_height; ++row) {
      for (auto col = 0; col < mid_width; ++col) {
        q1 += matrix[row][col];
```

```
        q2 += matrix[row][col + mid_width];
        q3 += matrix[row + mid_height][col];
        q4 += matrix[row + mid_height][col + mid_width];
      }
    }
    cout << "Q1:" << q1 << " Q2:" << q2 << endl << "Q3:" << q3 << " Q4:" <<
q4 << endl << endl;

    return make_tuple(q1, q2, q3, q4);
  };
  // function to flip Q2 to Q1 modifying matrix inplace
  auto flip_q2_to_q1 = [](vector<vector<int>> &matrix) {
    const auto height = matrix.size();
    const auto mid_height = height / 2;
    const auto width = matrix[0].size();
    const auto mid_width = width / 2;
    for (auto row = 0; row < mid_height; ++row) {
      auto sumq1 = 0L;
      auto sumq2 = 0L;
      for (auto col = 0; col < mid_width; ++col) {
        sumq1 += matrix[row][col];
        sumq2 += matrix[row][col + mid_width];
      }
      // if sumq2 > sumq1, flip row
      if (sumq2 > sumq1) {
        // for horizontal flip, all we need to do is reverse the row
        reverse(matrix[row].begin(), matrix[row].end());
      }
    }
  };
  // function to flip Q4 to Q3 modifying matrix inplace
  auto flip_q4_to_q3 = [](vector<vector<int>> &matrix) {
    const auto height = matrix.size();
    const auto mid_height = height / 2;
    const auto width = matrix[0].size();
    const auto mid_width = width / 2;
    for (auto row = 0; row < mid_height; ++row) {
      auto sumq3 = 0L;
      auto sumq4 = 0L;
      for (auto col = 0; col < mid_width; ++col) {
        sumq3 += matrix[row + mid_height][col];
        sumq4 += matrix[row + mid_height][col + mid_width];
      }
      // if sumq3 > sumq4, flip row
      if (sumq3 > sumq4) {
        // for horizontal flip, all we need to do is reverse the row
        reverse(matrix[row + mid_height].begin(),
                matrix[row + mid_height].end());
      }
    }
  };
  // function to flp (vertically) Q4 to Q2 modifying matrix inplace
  auto flip_q4_to_q2 = [](vector<vector<int>> &matrix) {
    const auto height = matrix.size();
    const auto mid_height = height / 2;
    const auto width = matrix[0].size();
    const auto mid_width = width / 2;
    for (auto col = 0; col < mid_width; ++col) {
      auto sumq2 = 0L;
      auto sumq4 = 0L;
      for (auto row = 0; row < mid_height; ++row) {
        sumq2 += matrix[row][col + mid_width];
        sumq4 += matrix[row + mid_height][col + mid_width];
```

```
      }
      // if sumq4 > sumq2, flip column
      if (sumq4 > sumq2) {
        // for vertical flip, we need to swap the values in the column
        // and the easiest way is to start at each ends and meet in the
middle
        for (auto row = 0; row < mid_height; ++row) {
          swap(matrix[row][col + mid_width],
               matrix[(height - 1) - row][col + mid_width]);
        }
      }
    }
  };

  auto q1_is_biggest = [](tuple<long, long, long, long> &quadrants) {
    auto q1 = get<0>(quadrants);
    auto q2 = get<1>(quadrants);
    auto q3 = get<2>(quadrants);
    auto q4 = get<3>(quadrants);
    return q1 > q2
        && q1 > q3
        && q1 > q4
        && q2 > q4
        && q4 > q3;
  };

dump_matrix(matrix);
  auto quadrants = sum_quadrants(matrix);
  auto q1 = get<0>(quadrants);
  auto q2 = get<1>(quadrants);
  auto q3 = get<2>(quadrants);
  auto q4 = get<3>(quadrants);
  while (q1_is_biggest(quadrants) == false) {
    if (q2 > q1) {
      flip_q2_to_q1(matrix);
    }
    quadrants = sum_quadrants(matrix);
    q1 = get<0>(quadrants);
    q2 = get<1>(quadrants);
    q3 = get<2>(quadrants);
    q4 = get<3>(quadrants);
    if (q3 > q4) {
      flip_q4_to_q3(matrix);
dump_matrix(matrix);
    }
    quadrants = sum_quadrants(matrix);
    q1 = get<0>(quadrants);
    q2 = get<1>(quadrants);
    q3 = get<2>(quadrants);
    q4 = get<3>(quadrants);
    if (q4 > q2) {
      flip_q4_to_q2(matrix);
dump_matrix(matrix);
    }

    quadrants = sum_quadrants(matrix);
    q1 = get<0>(quadrants);
    q2 = get<1>(quadrants);
    q3 = get<2>(quadrants);
    q4 = get<3>(quadrants);
  }
  // finally, return the sum of the upper-left quadrant
  return q1;
```

```
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 1	Easy	Sample case	Success	0	0.0518 sec	9 KB
Testcase 2	Easy	Hidden case	Wrong Answer	0	0.2236 sec	9.14 KB
Testcase 3	Easy	Hidden case	Wrong Answer	0	0.3381 sec	9.32 KB
Testcase 4	Easy	Hidden case	Wrong Answer	0	0.1628 sec	8.83 KB
Testcase 5	Easy	Hidden case	Wrong Answer	0	0.224 sec	9.45 KB
Testcase 6	Easy	Hidden case	Wrong Answer	0	0.272 sec	9.27 KB
Testcase 7	Easy	Hidden case	Wrong Answer	0	0.2765 sec	9.31 KB
Testcase 8	Easy	Sample case	Success	0	0.0648 sec	8.8 KB

No Comments