

Mock Test > sammetamanogna82@gmail.com

Full Name: Sammeta Manogna Email: sammetamanogna82@gmail.com Test Name: **Mock Test** Taken On: 15 Aug 2025 20:49:36 IST 22 min 59 sec/ 40 min Time Taken: Invited by: Ankush 14 Aug 2025 22:54:13 IST Invited on: Skills Score: Tags Score: Algorithms 195/195 Constructive Algorithms 90/90 Core CS 195/195 Easy 105/105 Greedy Algorithms 90/90 90/90 Medium Problem Solving 195/195 105/105 Search Sorting 105/105 problem-solving 195/195



scored in **Mock Test** in 22 min 59 sec on 15 Aug 2025 20:49:36 IST

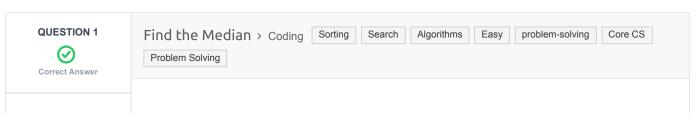
Recruiter/Team Comments:

No Comments.

Plagiarism flagged

We have marked questions with suspected plagiarism below. Please review it in detail here -





The median of a list of numbers is essentially its middle element after sorting. The same number of elements occur after it as before. Given a list of numbers with an odd number of elements, find the median?

Example

$$arr = [5, 3, 1, 2, 4]$$

The sorted array arr'=[1,2,3,4,5]. The middle element and the median is 3.

Function Description

Complete the findMedian function in the editor below.

findMedian has the following parameter(s):

• int arr[n]: an unsorted array of integers

Returns

• *int:* the median of the array

Input Format

The first line contains the integer n, the size of arr.

The second line contains n space-separated integers arr[i]

Constraints

- $1 \le n \le 1000001$
- **n** is odd
- $-10000 \le arr[i] \le 10000$

Sample Input 0

```
7
0 1 2 4 6 5 3
```

Sample Output 0

3

Explanation 0

The sorted arr = [0, 1, 2, 3, 4, 5, 6]. It's middle element is at arr[3] = 3.

CANDIDATE ANSWER

Language used: C

```
/*
    * Complete the 'findMedian' function below.

*    * The function is expected to return an INTEGER.

* The function accepts INTEGER_ARRAY arr as parameter.

*/
int compare(const void *a, const void *b) {
    return (*(int *)a - *(int *)b);

}
int findMedian(int arr_count, int* arr) {
    qsort(arr, arr_count, sizeof(int), compare);
    return arr[arr_count / 2];

}
```

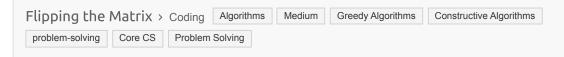






Needs Review

Score 90



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, $m{n}$.
- Each of the next 2n lines contains 2n space-separated integers matrix[i][j] in row i of the matrix.

Constraints

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

Sample Input

Sample Output

```
414
```

Explanation

Start out with the following 2n imes 2n matrix:

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

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

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

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

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

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

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

CANDIDATE ANSWER

Language used: C

```
1  /*
2  * Complete the 'flippingMatrix' function below.
3  *
4  * The function is expected to return an INTEGER.
5  * The function accepts 2D_INTEGER_ARRAY matrix as parameter.
6  */
7  int flippingMatrix(int matrix_rows, int matrix_columns, int** matrix) {
9   int n = matrix_rows / 2;
10  int total = 0;
```

```
for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
             int a = matrix[i][j];
14
              int b = matrix[i][matrix_columns - j - 1];
             int c = matrix[matrix_rows - i - 1][j];
             int d = matrix[matrix_rows - i - 1][matrix_columns - j - 1];
             int max = a;
              if (b > max) max = b;
             if (c > max) max = c;
             if (d > max) max = d;
             total += max;
         }
     }
      return total;
25 }
```

TESTCASEDIFFICULTYTYPESTATUSSCORETIME TAKENMEMORY USEDTestcase 1EasySample case✓ Success00.0086 sec7 KBTestcase 2EasyHidden case✓ Success150.0302 sec12 KBTestcase 3EasyHidden case✓ Success150.0563 sec15.1 KBTestcase 4EasyHidden case✓ Success150.037 sec11 KBTestcase 5EasyHidden case✓ Success150.0275 sec13 KBTestcase 6EasyHidden case✓ Success150.0365 sec14 KBTestcase 7EasyHidden case✓ Success150.0364 sec14.8 KBTestcase 8EasySample case✓ Success00.0088 sec7.13 KB							
Testcase 2 Easy Hidden case Success 15 0.0302 sec 12 KB Testcase 3 Easy Hidden case Success 15 0.0563 sec 15.1 KB Testcase 4 Easy Hidden case Success 15 0.037 sec 11 KB Testcase 5 Easy Hidden case Success 15 0.0275 sec 13 KB Testcase 6 Easy Hidden case Success 15 0.0365 sec 14 KB Testcase 7 Easy Hidden case Success 15 0.0364 sec 14.8 KB	TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 3 Easy Hidden case Success 15 0.0563 sec 15.1 KB Testcase 4 Easy Hidden case Success 15 0.037 sec 11 KB Testcase 5 Easy Hidden case Success 15 0.0275 sec 13 KB Testcase 6 Easy Hidden case Success 15 0.0365 sec 14 KB Testcase 7 Easy Hidden case Success 15 0.0364 sec 14.8 KB	Testcase 1	Easy	Sample case	Success	0	0.0086 sec	7 KB
Testcase 4 Easy Hidden case Success 15 0.037 sec 11 KB Testcase 5 Easy Hidden case Success 15 0.0275 sec 13 KB Testcase 6 Easy Hidden case Success 15 0.0365 sec 14 KB Testcase 7 Easy Hidden case Success 15 0.0364 sec 14.8 KB	Testcase 2	Easy	Hidden case	Success	15	0.0302 sec	12 KB
Testcase 5 Easy Hidden case Success 15 0.0275 sec 13 KB Testcase 6 Easy Hidden case Success 15 0.0365 sec 14 KB Testcase 7 Easy Hidden case Success 15 0.0364 sec 14.8 KB	Testcase 3	Easy	Hidden case	Success	15	0.0563 sec	15.1 KB
Testcase 6 Easy Hidden case	Testcase 4	Easy	Hidden case	Success	15	0.037 sec	11 KB
Testcase 7 Easy Hidden case Success 15 0.0364 sec 14.8 KB	Testcase 5	Easy	Hidden case	Success	15	0.0275 sec	13 KB
	Testcase 6	Easy	Hidden case	Success	15	0.0365 sec	14 KB
Testcase 8 Easy Sample case ✓ Success 0 0.0088 sec 7.13 KB	Testcase 7	Easy	Hidden case	Success	15	0.0364 sec	14.8 KB
	Testcase 8	Easy	Sample case	Success	0	0.0088 sec	7.13 KB

No Comments

PDF generated at: 15 Aug 2025 15:44:09 UTC