

Mock Test > brundan73@gmail.com

Full Name: Mutyala Brundan Reddy Email: brundan73@gmail.com Test Name: **Mock Test** Taken On: 13 Aug 2025 11:38:23 IST Time Taken: 7 min 5 sec/ 40 min Invited by: Ankush 13 Aug 2025 11:38:03 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

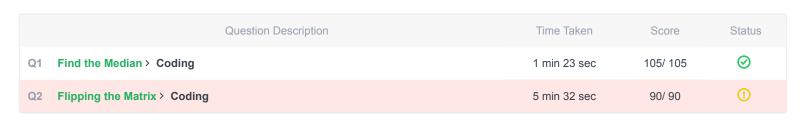


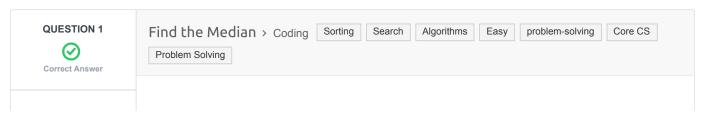
Recruiter/Team Comments:

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: Java 8

```
class Result {

/*

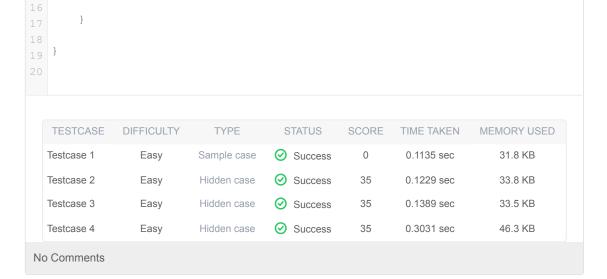
* Complete the 'findMedian' function below.

* The function is expected to return an INTEGER.

* The function accepts INTEGER_ARRAY arr as parameter.

*/

public static int findMedian(List<Integer> arr) {
  int median = 0;
  Collections.sort(arr);
  median = arr.size()/2;
  return arr.get(median);
}
```





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

$$matrix = \left[[1,2], [3,4] \right]$$

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

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

Sample Input

Sample Output

414

Explanation

Start out with the following $2n \times 2n$ 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 $n \times n$ submatrix in the upper-left quadrant:

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

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

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

CANDIDATE ANSWER

Language used: Java 8

```
1
2 class Result {
3
```

```
4
       * Complete the 'flippingMatrix' function below.
       * The function is expected to return an INTEGER.
       * The function accepts 2D_INTEGER_ARRAY matrix as parameter.
8
9
      public static int flippingMatrix(List<List<Integer>> matrix) {
      int max = 0;
     int n = matrix.size()/2;
14
     for(int i=0; i< n; i++) {
          for(int j=0;j< n;j++){
              int a = matrix.get(i).get(j);
              int b = matrix.get(i).get(2*n-1-j);
              int c = matrix.get(2*n-1-i).get(j);
              int d = matrix.get(2*n-1-i).get(2*n-1-j);
              max = max + Math.max(Math.max(a, b), Math.max(c, d));
          }
     }
     return max;
24
26 }
```

TESTCASEDIFFICULTYTYPESTATUSSCORETIME TAKENMEMORY USEDTestcase 1EasySample case✓ Success00.1104 sec31.9 KBTestcase 2EasyHidden case✓ Success150.3648 sec72.2 KBTestcase 3EasyHidden case✓ Success150.5795 sec91.2 KBTestcase 4EasyHidden case✓ Success150.4425 sec64.5 KBTestcase 5EasyHidden case✓ Success150.3528 sec76 KBTestcase 6EasyHidden case✓ Success150.426 sec85.3 KBTestcase 7EasyHidden case✓ Success150.3829 sec88.4 KBTestcase 8EasySample case✓ Success00.1234 sec32 KB							
Testcase 2 Easy Hidden case Success 15 0.3648 sec 72.2 KB Testcase 3 Easy Hidden case Success 15 0.5795 sec 91.2 KB Testcase 4 Easy Hidden case Success 15 0.4425 sec 64.5 KB Testcase 5 Easy Hidden case Success 15 0.3528 sec 76 KB Testcase 6 Easy Hidden case Success 15 0.426 sec 85.3 KB Testcase 7 Easy Hidden case Success 15 0.3829 sec 88.4 KB	TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 3 Easy Hidden case Success 15 0.5795 sec 91.2 KB Testcase 4 Easy Hidden case Success 15 0.4425 sec 64.5 KB Testcase 5 Easy Hidden case Success 15 0.3528 sec 76 KB Testcase 6 Easy Hidden case Success 15 0.426 sec 85.3 KB Testcase 7 Easy Hidden case Success 15 0.3829 sec 88.4 KB	Testcase 1	Easy	Sample case	Success	0	0.1104 sec	31.9 KB
Testcase 4 Easy Hidden case Success 15 0.4425 sec 64.5 KB Testcase 5 Easy Hidden case Success 15 0.3528 sec 76 KB Testcase 6 Easy Hidden case Success 15 0.426 sec 85.3 KB Testcase 7 Easy Hidden case Success 15 0.3829 sec 88.4 KB	Testcase 2	Easy	Hidden case	Success	15	0.3648 sec	72.2 KB
Testcase 5 Easy Hidden case Success 15 0.3528 sec 76 KB Testcase 6 Easy Hidden case Success 15 0.426 sec 85.3 KB Testcase 7 Easy Hidden case Success 15 0.3829 sec 88.4 KB	Testcase 3	Easy	Hidden case	Success	15	0.5795 sec	91.2 KB
Testcase 6 Easy Hidden case Success 15 0.426 sec 85.3 KB Testcase 7 Easy Hidden case Success 15 0.3829 sec 88.4 KB	Testcase 4	Easy	Hidden case	Success	15	0.4425 sec	64.5 KB
Testcase 7 Easy Hidden case Success 15 0.3829 sec 88.4 KB	Testcase 5	Easy	Hidden case	Success	15	0.3528 sec	76 KB
	Testcase 6	Easy	Hidden case	Success	15	0.426 sec	85.3 KB
Testcase 8 Easy Sample case ⊘ Success 0 0.1234 sec 32 KB	Testcase 7	Easy	Hidden case	Success	15	0.3829 sec	88.4 KB
	Testcase 8	Easy	Sample case	Success	0	0.1234 sec	32 KB

No Comments

PDF generated at: 13 Aug 2025 06:17:39 UTC