

### Mock Test > subiyamehek2003@gmail.com

Full Name: Subiya Mehek Email: subiyamehek2003@gmail.com Test Name: **Mock Test** Taken On: 9 Aug 2025 18:02:00 IST Time Taken: 2 min 34 sec/ 40 min Invited by: Ankush 9 Aug 2025 18:01:46 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



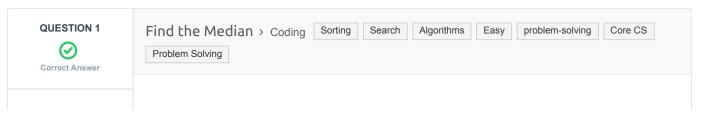
# **Recruiter/Team Comments:**

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

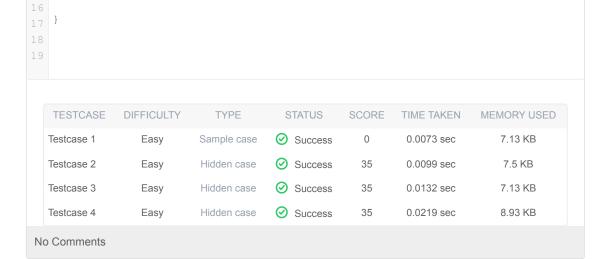
### **Explanation 0**

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

#### **CANDIDATE ANSWER**

### Language used: C

```
1
2  /*
3  * Complete the 'findMedian' function below.
4  *
5  * The function is expected to return an INTEGER.
6  * The function accepts INTEGER_ARRAY arr as parameter.
7  */
8  int compare (const void* a, const void* b) {
9    return (*(int*)a - *(int*)b);
10  }
11
12
13  int findMedian(int arr_count, int* arr) {
14    qsort(arr, arr_count, sizeof(int), compare);
15    return arr[arr_count / 2];
```









#### **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  $2 \times 2$  and we want to maximize the top left quadrant, a  $1 \times 1$  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**.
- ullet Each of the next  $m{2n}$  lines contains  $m{2n}$  space-separated integers  $m{matrix}[i][j]$  in row  $m{i}$  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: C

```
1
2 /*
3 * Complete the 'flippingMatrix' function below.
4 *
```

```
\,^\star The function is expected to return an INTEGER.
   * The function accepts 2D INTEGER ARRAY matrix as parameter.
8
9 int flippingMatrix(int matrix_rows, int matrix_columns, int** matrix) {
     int n = matrix_rows / 2;
      int sum = 0;
      for (int i = 0; i < n; i++) {
         for (int j = 0; j < n; j++) {
               int a = matrix[i][j];
               int b = matrix[i][matrix_columns - 1 - j];
              int c = matrix[matrix rows - 1 - i][j];
              int d = matrix[matrix_rows - 1 - i][matrix_columns - 1 - j];
              int max val = a;
              if (b > max_val) max_val = b;
             if (c > max_val) max_val = c;
              if (d > max_val) max_val = d;
              sum += max_val;
          }
      }
       return sum;
28 }
```

Testcase 1 Easy Sample case  Success 0 0.0082 sec 7.25 KE Testcase 2 Easy Hidden case  Success 15 0.0231 sec 12.3 KE Testcase 3 Easy Hidden case  Success 15 0.0369 sec 15.5 KE Testcase 4 Easy Hidden case  Success 15 0.0192 sec 11.1 KE Testcase 5 Easy Hidden case  Success 15 0.0389 sec 12.9 KE							
Testcase 2 Easy Hidden case    Success 15 0.0231 sec 12.3 KE  Testcase 3 Easy Hidden case    Success 15 0.0369 sec 15.5 KE  Testcase 4 Easy Hidden case    Success 15 0.0192 sec 11.1 KE  Testcase 5 Easy Hidden case    Success 15 0.0389 sec 12.9 KE	TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 3 Easy Hidden case ⊘ Success 15 0.0369 sec 15.5 KE  Testcase 4 Easy Hidden case ⊘ Success 15 0.0192 sec 11.1 KE  Testcase 5 Easy Hidden case ⊘ Success 15 0.0389 sec 12.9 KE	estcase 1	Easy	Sample case	Success	0	0.0082 sec	7.25 KB
Testcase 4 Easy Hidden case ⊘ Success 15 0.0192 sec 11.1 KE Testcase 5 Easy Hidden case ⊘ Success 15 0.0389 sec 12.9 KE	estcase 2	Easy	Hidden case	Success	15	0.0231 sec	12.3 KB
Testcase 5 Easy Hidden case   Success 15 0.0389 sec 12.9 KE	estcase 3	Easy	Hidden case	Success	15	0.0369 sec	15.5 KB
	estcase 4	Easy	Hidden case	Success	15	0.0192 sec	11.1 KB
Testcase 6 Easy Hidden case ⊘ Success 15 0.0336 sec 14.1 KE	estcase 5	Easy	Hidden case	Success	15	0.0389 sec	12.9 KB
	estcase 6	Easy	Hidden case	Success	15	0.0336 sec	14.1 KB
Testcase 7 Easy Hidden case ⊘ Success 15 0.0319 sec 14.8 KE	estcase 7	Easy	Hidden case	Success	15	0.0319 sec	14.8 KB
Testcase 8 Easy Sample case ⊘ Success 0 0.0092 sec 7.38 KE	estcase 8	Easy	Sample case	Success	0	0.0092 sec	7.38 KB

No Comments

PDF generated at: 9 Aug 2025 12:36:23 UTC