Problem 2545: Sort the Students by Their Kth Score

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

There is a class with

m

students and

n

exams. You are given a

0-indexed

m x n

integer matrix

score

, where each row represents one student and

score[i][j]

denotes the score the

i

th

student got in the

j

th

exam. The matrix

score

contains

distinct

integers only.

You are also given an integer

k

. Sort the students (i.e., the rows of the matrix) by their scores in the

k

th

(

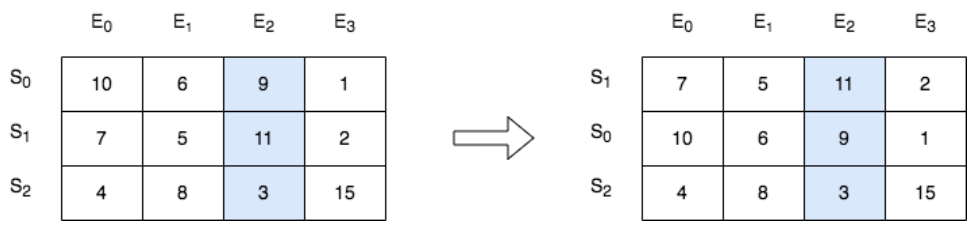
0-indexed

) exam from the highest to the lowest.

Return

the matrix after sorting it.

Example 1:



Input:

score = [[10,6,9,1],[7,5,11,2],[4,8,3,15]], k = 2

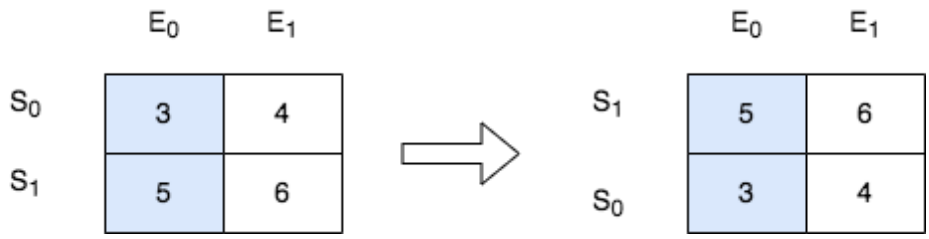
Output:

[[7,5,11,2],[10,6,9,1],[4,8,3,15]]

Explanation:

In the above diagram, S denotes the student, while E denotes the exam. - The student with index 1 scored 11 in exam 2, which is the highest score, so they got first place. - The student with index 0 scored 9 in exam 2, which is the second highest score, so they got second place. - The student with index 2 scored 3 in exam 2, which is the lowest score, so they got third place.

Example 2:



Input:

score = [[3,4],[5,6]], k = 0

Output:

[[5,6],[3,4]]

Explanation:

In the above diagram, S denotes the student, while E denotes the exam. - The student with index 1 scored 5 in exam 0, which is the highest score, so they got first place. - The student with index 0 scored 3 in exam 0, which is the lowest score, so they got second place.

Constraints:

m == score.length

n == score[i].length

1 <= m, n <= 250

1 <= score[i][j] <= 10

5

score

consists of

distinct

integers.

0 <= k < n

Code Snippets

C++:

```
class Solution {
public:
vector<vector<int>> sortTheStudents(vector<vector<int>>& score, int k) {

}
};
```

Java:

```
class Solution {
public int[][] sortTheStudents(int[][] score, int k) {

}
}
```

Python3:

```
class Solution:
def sortTheStudents(self, score: List[List[int]], k: int) -> List[List[int]]:
```

Python:

```
class Solution(object):
def sortTheStudents(self, score, k):
"""
:type score: List[List[int]]
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript:

```
/**
* @param {number[][]} score
* @param {number} k
* @return {number[][]}
*/
var sortTheStudents = function(score, k) {

};
```

TypeScript:

```
function sortTheStudents(score: number[][], k: number): number[][] {

};
```

C#:

```
public class Solution {
public int[][] SortTheStudents(int[][] score, int k) {

}
}
```

C:

```
/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** sortTheStudents(int** score, int scoreSize, int* scoreColSize, int k,
int* returnSize, int** returnColumnSizes) {

}
```

Go:

```
func sortTheStudents(score [][]int, k int) [][]int {

}
```

Kotlin:

```
class Solution {
fun sortTheStudents(score: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift:

```
class Solution {
func sortTheStudents(_ score: [[Int]], _ k: Int) -> [[Int]] {

}
}
```

Rust:

```
impl Solution {
pub fn sort_the_students(score: Vec<Vec<i32>>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby:

```
# @param {Integer[][]} score
# @param {Integer} k
# @return {Integer[][]}
def sort_the_students(score, k)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $score
* @param Integer $k
* @return Integer[][]
*/
function sortTheStudents($score, $k) {

}
}
```

Dart:

```
class Solution {
List<List<int>> sortTheStudents(List<List<int>> score, int k) {

}
}
```

Scala:

```
object Solution {
def sortTheStudents(score: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
```

```
}
```

Elixir:

```
defmodule Solution do
@spec sort_the_students(score :: [[integer]], k :: integer) :: [[integer]]
def sort_the_students(score, k) do

end
end
```

Erlang:

```
-spec sort_the_students(Score :: [[integer()]], K :: integer()) ->
[[integer()]].
sort_the_students(Score, K) ->
.
```

Racket:

```
(define/contract (sort-the-students score k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))
)
```

Solutions

C++ Solution:

```
/*
* Problem: Sort the Students by Their Kth Score
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
```

```
public:
vector<vector<int>> sortTheStudents(vector<vector<int>>& score, int k) {

}
};
```

Java Solution:

```
/**
* Problem: Sort the Students by Their Kth Score
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int[][] sortTheStudents(int[][] score, int k) {

}
}
```

Python3 Solution:

```
"""
Problem: Sort the Students by Their Kth Score
Difficulty: Medium
Tags: array, sort

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def sortTheStudents(self, score: List[List[int]], k: int) -> List[List[int]]:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def sortTheStudents(self, score, k):
"""
:type score: List[List[int]]
:type k: int
:rtype: List[List[int]]
"""
```

JavaScript Solution:

```
/**
* Problem: Sort the Students by Their Kth Score
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number[][]} score
* @param {number} k
* @return {number[][]}
*/
var sortTheStudents = function(score, k) {

};
```

TypeScript Solution:

```
/**
* Problem: Sort the Students by Their Kth Score
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
function sortTheStudents(score: number[][], k: number): number[][] {

};
```

C# Solution:

```
/*
* Problem: Sort the Students by Their Kth Score
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int[][] SortTheStudents(int[][] score, int k) {

}
}
```

C Solution:

```
/*
* Problem: Sort the Students by Their Kth Score
* Difficulty: Medium
* Tags: array, sort
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* Return an array of arrays of size *returnSize.
* The sizes of the arrays are returned as *returnColumnSizes array.
* Note: Both returned array and *columnSizes array must be malloced, assume
caller calls free().
*/
int** sortTheStudents(int** score, int scoreSize, int* scoreColSize, int k,
```

```
int* returnSize, int** returnColumnSizes) {

}
```

Go Solution:

```
// Problem: Sort the Students by Their Kth Score
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func sortTheStudents(score [][]int, k int) [][]int {

}
```

Kotlin Solution:

```
class Solution {
fun sortTheStudents(score: Array<IntArray>, k: Int): Array<IntArray> {

}
}
```

Swift Solution:

```
class Solution {
func sortTheStudents(_ score: [[Int]], _ k: Int) -> [[Int]] {

}
}
```

Rust Solution:

```
// Problem: Sort the Students by Their Kth Score
// Difficulty: Medium
// Tags: array, sort
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
```

```
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn sort_the_students(score: Vec<Vec<i32>>, k: i32) -> Vec<Vec<i32>> {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} score
# @param {Integer} k
# @return {Integer[][]}
def sort_the_students(score, k)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $score
* @param Integer $k
* @return Integer[][]
*/
function sortTheStudents($score, $k) {

}
}
```

Dart Solution:

```
class Solution {
List<List<int>> sortTheStudents(List<List<int>> score, int k) {

}
}
```

Scala Solution:

```
object Solution {
def sortTheStudents(score: Array[Array[Int]], k: Int): Array[Array[Int]] = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec sort_the_students(score :: [[integer]], k :: integer) :: [[integer]]
def sort_the_students(score, k) do

end
end
```

Erlang Solution:

```
-spec sort_the_students(Score :: [[integer()]], K :: integer()) ->
[[integer()]].
sort_the_students(Score, K) ->
.
```

Racket Solution:

```
(define/contract (sort-the-students score k)
(-> (listof (listof exact-integer?)) exact-integer? (listof (listof
exact-integer?)))
)
```