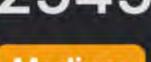
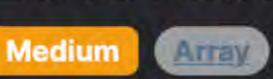
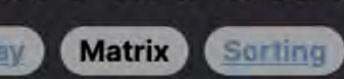
2545. Sort the Students by Their Kth Score





Problem Description



In this problem, we have a class containing m students and n exams. We are presented with a matrix score of size m x n, where score[i][j] represents the score of the ith student in the jth exam. It's important to note that the scores are distinct, meaning no two students have received the exact same score in an exam.

Leetcode Link

We are also given an integer k, which refers to a specific exam. The task is to sort the students based on their scores in this kth exam. The sorting should be done in descending order - the student with the highest score on the kth exam should come first, and so on. What we aim to achieve is a new matrix with the same rows as the original score matrix, but ordered such that when looking at column k, the scores are in decreasing order.

Ultimately, we need to return the matrix once it has been sorted by these criteria.

Intuition

The solution to this problem is relatively straightforward due to Python's powerful sorting functions. We can use the built-in sorted function to sort the rows (students) based on their score in the kth exam (k is a 0-indexed exam number).

To specify the sorting criteria, we provide a lambda function to the key argument of sorted, which tells it to use the score at index k of each row as the key for sorting. Since we want to sort the scores in descending order, we negate the scores by using -x[k]. This means that higher scores will be treated as "smaller" by the sorting algorithm because of the negative sign, thereby placing them at the front of the sorted list.

higher priority in the sort order, hence sorting the array in descending order based on the kth index. So, the solution, in essence, is simply a one-liner that returns a new matrix score sorted by the score of the kth exam for each

This approach leverages Python's negative value ordering where it treats smaller values (negative values in this case) as having

student, in descending order.

Solution Approach

The implementation is concise because it relies on Python's sorting algorithm, TimSort, an optimized hybrid sorting algorithm derived from merge sort and insertion sort. Here is a step-by-step breakdown:

scores across all exams. 2. The key parameter of the sorted function takes a lambda function. This lambda function lambda x: -x[k] is used to determine

the sorting order. The key parameter defines the basis on which the elements of score are sorted. In this instance, the lambda

1. The sorted function is called on the matrix score. In this case, score is a list of lists, with each inner list representing a student's

- returns the score of the kth exam (0-indexed) but negated, so high scores become lower numbers, and because Python sort is ascending by default, this negative transformation ensures that we get a descending order. 3. The inner workings of the sorted function then automatically handle the comparison of these keys (i.e., the negated kth exam
- scores) for each row in the matrix and arrange them in ascending order (which, due to the negation, corresponds to descending order of the actual scores).
- 5. The sorted list of lists, i.e., the sorted score matrix, is then returned as output. This new matrix has rows ordered by descending

4. As Python lists are 0-indexed, k refers directly to the correct column in the list of lists without requiring any adjustment.

scores of the kth exam, which is exactly what the problem statement asked for. Thus, the algorithm makes efficient use of Python's built-in sorting facilities and does not require any additional complex data

structures or patterns. It's a simple, elegant solution that takes advantage of the way Python's sort works with keys and negation to achieve the desired descending order. The full implementation in Python, as provided in the original solution, is as follows:

def sortTheStudents(self, score: List[List[int]], k: int) -> List[List[int]]: return sorted(score, key=lambda x: -x[k])

Let's consider a small example to better understand the solution approach described above.

Example Walkthrough

1 class Solution:

Suppose we have the following matrix score where m=3 (students) and n=4 (exams), and each cell score[i][j] represents the score

of student i in exam j: 1 score = [60, 85, 75, 90], # Student 1

```
[85, 85, 85, 88] # Student 3
We are also given k=2, which refers to the third exam (remember, indices start at 0).
```

[88, 92, 67, 75], # Student 2

1 Exam 3 scores: [75, 67, 85]

We need to sort these scores in descending order and then rearrange the score matrix accordingly.

Our goal is to sort the students based on their scores in the kth exam. So we need to look at the scores in exam 3:

Now, using the solution approach:

x[2].

1 score = [

[60, 85, 75, 90],

print(sorted_score)

1. We call the sorted function on the score matrix. 2. The key parameter is provided with a lambda function lambda x: -x[k]. Since k=2, the lambda function becomes lambda x: -

3. The sorted function applies this key (sorting criteria) to each student's list of exam scores. 4. Negative values are created for the scores of the third exam, which results in the following transformed scores: [-75, -67,

[88, 92, 67, 75] # Student 2

- -85]. 5. These negative values are sorted in ascending order automatically by the sorted function, leading to the following order (while
- preserving the original scores): [-85, -75, -67]. 6. This order corresponds to sorting the original scores of the third exam in descending order: [85, 75, 67].

7. Finally, the score matrix is reordered to match this sorting, giving us the following result:

1 sorted_score = [[85, 85, 85, 88], # Student 3 [60, 85, 75, 90], # Student 1

```
Here, if you look at the third column, which corresponds to exam 3, you'll see the scores are sorted in descending order: 85, 75, 67.
The code for this in Python, using the provided solution approach, would look like this:
```

[88, 92, 67, 75], [85, 85, 85, 88] 6 k = 2

```
The output of this code snippet will be the sorted matrix:
  [[85, 85, 85, 88],
    [60, 85, 75, 90],
    [88, 92, 67, 75]]
As you can see, the students are now sorted by their scores in the third exam, in descending order, as required.
```

sorted_score = sorted(score, key=lambda x: -x[k])

Sort the students by the k-th score in descending order. # The lambda function extracts the k-th element from each sub-list for comparison. # The minus sign '-' is used to sort in descending order. sorted_scores = sorted(scores, key=lambda student_scores: -student_scores[k])

def sortTheStudents(self, scores: List[List[int]], k: int) -> List[List[int]]:

```
Java Solution
```

Python Solution

class Solution:

9

10

from typing import List

return sorted_scores

```
import java.util.Arrays; // Import Arrays class for sorting functionality
   class Solution {
       /**
        * Sorts the students based on the scores in a specific column.
        * @param scores A 2D array representing the scores of the students. Each row is a student, and each column is a score for a diff
        * @param k The index of the score based on which the sorting should be performed.
        * @return The sorted 2D array of students' scores.
10
11
12
       public int[][] sortTheStudents(int[][] scores, int k) {
           // Sort the 2D array 'scores' using a custom comparator
13
           // that compares the elements in column 'k' in descending order.
14
           Arrays.sort(scores, (a, b) -> b[k] - a[k]);
15
           return scores; // Return the sorted array.
16
17
```

C++ Solution

18 }

19

```
1 #include <vector> // Include the necessary header for using vectors
2 #include <algorithm> // Include the algorithm header for the sort function
   class Solution {
  public:
       // Function to sort the students based on their scores at index 'k'
       // score: A 2D vector containing students and their scores
       // k: The index of the score to sort by
       // Returns: The sorted 2D vector of students
 9
       vector<vector<int>>> sortTheStudents(vector<vector<int>>& scores, int k) {
10
11
12
           // Use the standard sort function to reorder the 'scores' vector
13
           // Sort is done in descending order based on the value at index 'k'
           sort(scores.begin(), scores.end(), [&](const auto& a, const auto& b) {
14
15
                return a[k] > b[k]; // Lambda function to compare scores at index 'k'
           });
16
17
           return scores; // Return the sorted vector
19
20 };
21
```

Typescript Solution

```
* Sorts a matrix of student scores based on a particular score index.
    * @param {number[][]} scores - A matrix where each row represents a student's scores.
    * @param {number} scoreIndex - The index of the score to sort the students by.
    * @returns {number[][]} - The sorted matrix with students ordered by the score at the given index.
6
    */
   function sortTheStudents(scores: number[][], scoreIndex: number): number[][] {
       // Sorts the array of scores in descending order based on the score at scoreIndex within each student's array.
       return scores.sort((studentA, studentB) => studentB[scoreIndex] - studentA[scoreIndex]);
10 }
11
```

Time and Space Complexity

the built-in sorted function in Python uses an algorithm called Timsort, which has this time complexity on average. The space complexity of the provided code is O(n), due to the space required to hold the new sorted list that is returned by the

sorted function. The original list is not modified in place; instead, a new list is created to hold the sorted result.

The time complexity of the provided sorting algorithm is $O(n \log n)$, where n is the number of items in the score list. This is because