DSA

Name ..... ID .....

You are given the results of a series of football (soccer) matches between 8 teams, identified by team IDs from 0 to 7. Each match is represented as a 4-element integer array:

`{teamA_id, teamB_id, teamA_goals, teamB_goals}`

Write `static int[][] computeResults(List<int []> matches)`

You are to process these results and generate a **league table** for all participating teams. Each row in the league table should contain the following fields:

`{team_id, number_of_matches_played, goals_for, goals_against, points}`

Where:

`number_of_matches_played` is the total number of matches the team participated in.

`goals_for` is the total number of goals the team scored.

`goals_against` is the total number of goals the team conceded.

`points` is awarded as:

3 points for a win 1 point for a draw 0 points for a loss

You must **sort the final table** by:

- **Points** (descending)
- **Goal difference** (goals_for - goals_against) (descending)
- **Goals scored** (goals_for) (descending)

(next page)

DSA

Name ..... ID .....

```
static void demo_1() {
  Map<Integer, String> teamNames = Map.of(
            1, "phy", 2, "chem", 3, "bio",
            4, "math", 5, "stat", 6, "com", 7, "kdai"
        );
  List<int[]> results = Arrays.asList(
        new int[]{1,2,1,2}, new int[]{1,3,2,0}, new int[]{1,4,0,0},
                    new int[]{1,5,0,1}, new int[]{1,6,1,2}, new int[]{1,7,2,2},
                            new int[]{2,3,3,2}, new int[]{2,4,0,1},
                    new int[]{2,5,3,3}, new int[]{2,6,3,0}, new int[]{2,7,2,0},
                                                new int[]{3,4,1,0},
                    new int[]{3,5,1,0}, new int[]{3,6,2,3}, new int[]{3,7,0,0},

                    new int[]{4,5,3,1}, new int[]{4,6,0,0}, new int[]{4,7,1,2},

                                        new int[]{5,6,0,0}, new int[]{5,7,1,0},
                                                            new int[]{6,7,1,0}
        );
  int [][] table = computeTable(results);

  displayTable(table, teamNames);
}
static void displayTable(int [][] table, Map<Integer,String> teamNames) {
    System.out.println("Team  MP  GF  GA  Pts");
    for (int[] row : table) {
        String name = teamNames.getOrDefault(row[0], "Team" + row[0]);
        System.out.printf("%-5s %3d %3d %3d %4d\n",
                name, row[1], row[2], row[3], row[4]);
    }
}
static int[][] computeTable(List<int[]> matches) {
  int maxTeamId = 7; // { n*(n-1) } /  = 21 matches -> n*n - n - 42 = 0 -> n = 7;
  int[][] stats = new int[maxTeamId][5];  // [teamId, MP, GF, GA, Pts]
  for (int i = 0; i < maxTeamId; i++)
     stats[i][0] = i + 1;

  /* your code – keep in mind that team_id j is at row (j - 1)*/

  return stats
}
```

(next page)

DSA

Name ..... ID .....

Java syntax for sorting rows in an array is

```
Arrays.sort(stats, (a, b) -> {
    return Integer.compare(a[0], b[0]);
    // sort by team_id
});
```

Expected Result

Team	MP	GF	GA	Pts
chem	6	13	7	13
com	6	6	6	11
math	6	5	4	8
stat	6	6	7	8
bio	6	6	8	7
phy	6	6	7	5
kdai	6	4	7	5

Classic style using simple 2D int arrays is straightforward and efficient. Functional style programming, like using Java Streams with a TeamStat class, better reflects the power of abstraction and can simplify code. However, this advantage doesn't extend well when working directly with primitive 2D arrays.

Submit: Lab0_LeagueTable_XXYYYY.java which completes the
`int[][] computeTable(List<int[]> matches)`

Due: TBA