Introduction of ITR HW2

The goal of this assignment is to simulate how students can be assigned to courses based on some predefined conditions.

You will need to select or design data structures and algorithms based on everything you learned in this class to provide an optimal solution.

M104020014 周紘樟

Data Structure and Algorithm

ArrayList data structure was implemented to store the data of the students.

The reason is that the size of ArrayList is easy to resize.

Also, the sort() method in Collections helps me easily, by overriding compare method to help sort the students by their year and id.

```
private static Course[] simulate(Student[] students, Course[] courses) {
  ArrayList<Student> sortList = new ArrayList<>(); //ArrayList for students with preference to sort
  ArrayList<Student> blankList = new ArrayList<>();//ArrayList for students with no preference

  for(Student student : students){  //for loop go through sutdents' data
    if(student.preference.length==0) {  //if student doesn't have any preference
      blankList.add(student);           //add into blankList
      continue;                         //continue the loop looking for next one
    }else{
      sortList.add(student);            //otherwise we add students with preference into sortList
    }
  }
```

Data Structure and Algorithm

As for the algorithm, which is a MergeSort.

By overriding the compare method, I can manipulate the function to compare according to the requirements.

```
Collections.sort(sortList, new Comparator<Student>(){
public int compare(Student s1, Student s2)
  { // We sort the sortlist depends on two conditions,
    //if the years are the same then we compare student id (AESC order)
    if(Integer.compare(s2.year, s1.year) == 0) return Integer.compare(s1.id, s2.id);
    else return Integer.compare(s2.year, s1.year); //else we compare thier years(DESC order)
  }
});
Collections.sort(blankList, new Comparator<Student>(){
  public int compare(Student s1, Student s2)
    { // We sort the blankList depends on two conditions,
      //if the years are the same then we compare student id (AESC order)
      if(Integer.compare(s2.year, s1.year) == 0) return Integer.compare(s1.id, s2.id);
      else return Integer.compare(s2.year, s1.year); //else we compare thier years(DESC)
    }
  });

sortList.addAll(blankList); //Finally, We concatenate two sorted arraylists
                            //which students with no preferences stay at the end of the arrayList
```

Time complexity approximately : O(nlogn)

Reference: https://www.geeksforgeeks.org/how-to-override-compareto-method-in-java/

How it works ?...

1. Check whether students have preferneces for the courses or not.
2. We separate them into two ArrayLists, one includes preferences, and the other doesn't.
3. Sort two ArrayList by year and id independently, then we concatenate them together as a new ArrayList.
4. Loop through the sorted ArrayList and check each students' preference once at a time.
5. If the first-preferred course is available, we add that student into the course candidates.Then we record the the action which takes 1 candidate spot of the course.
6. If not, we check the student's next-preferred course and reassure there is still available spots, and so on.
7. Repeat 5. and 6. for the rest of the students with preferences.
8. By the time there are students with preferneces still get no chances to pick a course, we loop through the courses to find the rest of the spots in courses for students to add.
9. Finally, those with no preferences, we loop through the courses to find the rest of the spots in courses for students to add.

Time and Space Complexity Analysis

Time Complexity for the code

Collections.sort() for sorting the students by their year and id :

$O(nlogn)$, where n is the size of the students

※ *The sorting algorithm is a modified mergesort (in which the merge is omitted if the highest element in the low sublist is less than the lowest element in the high sublist).*

This algorithm offers guaranteed n log(n) performance. – Java Documentation

Preference simulation process due to nested for loop :

$O(n^2)$, where n is the size of the students

Time Complexity for the code

ArrayLists to store students' year and id :

$O(n)$, where n is the size of the students