Data Structures – practical grade

Student registry

Duration: 90 minutes · **Language:** Java or Go (your choice) · **Computer:** the computer of the lab
Allowed: the demo programs and slides of the course, your own notes · **Not allowed:** internet, artificial intelligence (AI) tools, help from other students

The task

We keep a registry of the students of a university. Every student has an **id** (`id` , integer), a **name** and a **score** (0–100). Implement the operations marked with **TODO** in the starter code **with your own data structure**!

	Java	Go
Starter code	`practical/java/StudentRegistry.java`	`practical/go/main.go`
Run	`java StudentRegistry.java`	`go run .` (in the `go` folder)
Forbidden in the solution	`java.util` collections (`ArrayList` , `LinkedList` , `HashMap` , `TreeMap` …)	`container/list` , `map` , built-in sorting

At the end the program prints **level by level** which checks pass (**OK**) and which fail (**FAIL**). Do **not modify** the checker part!

The levels

The levels must be completed **in order**: a level counts only if all previous levels are flawless.

Level 1 – pass (2): singly linked list

- `add(s)` – a new student at the **end** of the list
- `size()` – the number of students
- `find(id)` – search by id (not found: `null` / `nil`)
- `toString()` / `String()` – e.g. `[1003 Li Wei 87, 1001 Wang Fang 92]` , empty list: `[]`

Level 2 – satisfactory (3)

- `remove(id)` – deleting the student; true if there was one (take care of deleting the **first** element too!)
- `averageScore()` – the average of the scores (0 for an empty list)
- `best()` – the student with the highest score; on a tie the one earlier in the list

Level 3 – good (4)

- `addSortedByScore(s)` – insertion so that the list stays in **descending** order of score; on equal scores the new student goes **behind** the ones already in the list (stable insertion)
- `reverse()` – reversing the list **in place**, without creating new nodes

Level 4 – excellent (5): binary search tree (key: the id)

- `insert(s)` – insertion; a duplicate id is **not** inserted again
- `find(id)` – search by id
- `height()` – the height of the tree (empty tree: −1, a single node: 0)
- `inorderIds()` – the ids in inorder, e.g. `[1001, 1002, 1003]`

The check of level 4 also runs a **measurement** (20 000 searches in a list and in a search tree). Observe the result – you will be asked about it in the oral part!

Assessment

1. **Automatic check:** the level printed by the program is the **highest achievable** grade.
2. **Oral check** (about 5 minutes): you must be able to explain your own code and answer 2–3 questions (e.g. *How many steps does the `find` operation take? What happens when the first element is deleted?*).
   If the explanation is not convincing, the grade **may be lowered**.
3. The solution is **not accepted** if it does not compile, if it uses a forbidden collection, or if the checker part has been modified.

Tips

- Proceed **level by level**, and run the program after every method!
- Think of the **special cases**: empty list, one-element list, the first and the last element.
- The demo programs of weeks 9, 10, 14 and 15 help a lot.
- Save your work often!

Name: ..... **ID:** ..... **Chosen language:** Java / Go