Complexity analysis- Student attendance manager

Time & Space Complexity of Operations

1. `addStudent(const Student& s)`

- **Time Complexity:**
  - Average case: **O(1)** (amortized, since `vector::emplace_back` usually adds in constant time).
  - Worst case: **O(n)** if the vector needs to resize (rare).
- **Space Complexity:**
  - **O(1)** extra (just adds one student object, vector handles resizing internally).

---

2. `findAvgAttendance()`

- **Time Complexity: O(n)** → one pass over all students to sum attendances.
- **Space Complexity: O(1)** → only a few variables (`sum` , `avg`).

---

3. `findHighest()`

- **Time Complexity: O(n)** → one pass through students to track maximum, another implicit for push_backs (still O(n) overall).
- **Space Complexity: O(k)** → where `k` is the number of students who share the highest attendance. Worst case: all students have same attendance → **O(n)**.

---

4. `findLowest()`

- **Time Complexity: O(n)** (similar to highest, single pass + conditional push_backs).
- **Space Complexity: O(k)** → `k` students with the same lowest attendance. Worst case: all students → **O(n)**.

---

5. `findZero()`

- **Time Complexity: O(n)** → scan once through all students.
- **Space Complexity: O(1)** → only a counter variable.

---

6. `findMostCommonCount()`

- **Time Complexity:**
  - Build frequency map: **O(n)** (iterate once through all students).
  - Traverse unordered_map: at most **O(u)**, where `u` is number of unique attendance values (≤ n).
  - Total: **O(n + u)** ≈ **O(n)** in worst case.
- **Space Complexity:**
  - **O(u)** for frequency map.
  - **O(m)** for storing mode values (where `m ≤ u ≤ n`).
  - Worst case: **O(n)**.

Complexity Summary Table

Method	Time Complexity	Space Complexity
`addStudent`	O(1) amortized	O(1)
`findAvgAttendance`	O(n)	O(1)
`findHighest`	O(n)	O(k) ≤ O(n)
`findLowest`	O(n)	O(k) ≤ O(n)
`findZero`	O(n)	O(1)
`findMostCommonCount`	O(n)	O(u) ≤ O(n)
