1.3 Question 2

I wrote a program to calculate how many unique students I have taught over the years. `first_name last_name sid` and populate a `vector` of `Student*` pointers. To ca students the following was applied:

```
int unique_students = 0, comparisons = 0;
for (unsigned i = 0;i < students.size();++i) {
    bool already_counted = false;
    for (unsigned j = 0;j < i;++j) {
        ++comparisons;
        if (v.at(i) == v.at(j)) {
            already_counted = true;
        }
    }
    if (!already_counted) { ++unique_students; }
}
```

At the end of the nested for loop, the variable `unique_students` contains the numbe

I already know that this method is suboptimal, and have provided a result of the numbe solution. I am, however, at a loss of how to optimize this solution. That is where you cor

Runtime analysis

First, annotate the existing method (in the provided `main.cpp` template) with the run-ti
overall run-time and space complexity in a comment at the top of the outermost loop. *D*
you use in your Big-oh notation.

Optimization

Implement the `IntSet` class (header provided) and use this data structure to reduce th
necessary. The `IntSet` is a **set** of integers; a **set** is a data structure that stores unique
order. Details of the IntSet:

- Implemented as a hash table
- Uses *open addressing* with *linear probing*
- When inserting an element causes the **size** of the set to increase beyond the `capa`
  `rehash` and increase the capacity to `capacity * 3`.
- When inserting a duplicate element, do nothing.
- When erasing an element that does not exist, do nothing.
- The function count will return a 1 if the key passed in exists in the set and a 0 othe
- You **may** implement additional **private** helper functions as you see necessary

Public interface:

- IntSet(): Default constructor sets capacity to 1
- explicit IntSet(int n): Set the capacity to n
- IntSet(const IntSet &): Copy constructor
- void insert(int key): If the `key` exists in the set, do nothing, otherwise, insert it.
- void erase(int key): If the `key` does not exist in the set, do nothing, otherwise, eras
- void rehash(size_t): Rehash to the new size passed in.
- int comparisons(): Returns the number of camparisons from the last operation pe
- size_t size() const: Returns the size
- size_t capacity() const: Returns the capacity
- bool empty() const: Returns true if the set is empty, false otherwise.
- int count(int key): Returns 1 if the key passed in exists in the set and 0 otherwise.

Private functions:

- Used for proper hashing. you should implement all of 'probe', 'hash', 'hash*map', an*

Counting Comparisons

You will need to implement a comparison count in your **set** to show that it is faster than

counting comparisons, any `if-else if-else` statement counts as **1 comparison**. Tl

```
if (x < 5) { ... }
else if (x < 10) { ... }
else { ... }
```

will count as only **1 comparison**. The `int comparison()` function will return the num
the previous function call. The following operations need to track comparisons:

- `insert`
- `erase`
- `count`
- `rehash`

Don't forget, if one of the above functions calls another of the above functions, you neec
in that action (pay careful attention to **insert** and **rehash**).

Count unique students

Once you have implemented the `IntSet` class, you will use it to calculate the number c
taught. In your new optimized algorithm, annotate each line of code (in the `main.cpp` c
annotation to the top with it's run-time and space complexity.

Don't change any code above the `/* Insert Solution Here */ comment. You
naive solution and your optimized solution. The final output wh
the`smallInput.txt` file should be:

```
Unique students: 10
    Comparisons made: 45
Unique students: 10
    Comparisons made: 38
```

LAB ACTIVITY | 1.3.1: Question 2

Submission Instructions

Downloadable files

`main.cpp` , `Student.h` , `Student.cpp` , `IntSet.h` , `smallDuplicates.txt` , and `smallInput.txt`

Compile command