Intermediate Operations you should know

- **`stream() // Not an intermediate operation`**
    - Must first call `.stream()` to convert a collection into stream
- **`sorted()`**
    - Sorts the stream
    - Can pass in a `Comparator` (i.e a lambda) to defy natural ordering of the Stream
- **`map(Function f)`**
    - *Applies* the function on each of the elements of the stream
- **`filter(Predicate p)`**
    - *Tests* each element to determine if it should remain in the stream

Terminal Operations you should know

- `reduce(BinaryOperator accumulator)`
  - *Applies* the accumulator and combines elements in the stream, two at a time
  - ~~Returns an Optional~~ ==> `orElse(T other)`
- `collect(Collector c)`
  - Turns the stream into a collection
  - `Collectors.toList()`, `Collectors.toSet()`, `Collectors.groupingBy(Function f)`
- `forEach(Consumer action)`

For the following problems, use the classes defined below. You may use only one semicolon per method.

```
public class Student {
    public int getLabSection() { ... }
    public int getAge() {...}
    public Group getGroup {...}
}

public class Group {
    public int size() {...}
    public Group merge(Group other) { ... }
}
```

Q1 uniqueSections

Collect all the unique lab sections of a list of students. There should not be duplicates.

```
public Set<Integer> uniqueSections(List<Student> lst) {







}
```

Q2 grpMap

Each TA submits a list of their students. Write a method that will map each lab section to the students in that section whose group is larger than size 4. Assume there is always at least one student in each section whose group size is larger than 4.

```
public Map<Integer, List<Student>> grpMap (List<List<Student>> lst) {




}
```

Q3 over20

Each TA submits a list of their students. Write a method that will get the number of students who are over age 20.

```
public int over20 (List<List<Student>> lst) {



}
```

Q4 groupMerge

Each TA submits a list of the groups in their section. Write a method that will merge all the groups in a section into one supergroup and return that supergroup, if it is the case that there are less than 4 groups in a section. If there are no sections with less than 4 groups in it, then return null

```
public Group groupMerge(List<List<Group>> lst) {




}
```