Periodically, a company processes the retirement of some of its employees. In this question, you will write methods to help the company determine whether an employee is eligible to retire and to process the retirement of employees who wish to retire. You will also analyze the runtime performance of one of the methods that you write.

The Java interface `Employee` is implemented by objects that represent employees. The interface is declared as follows.

```java
public interface Employee
{
    public int getAge(); // returns age of employee
    public int getYearsOnJob(); // returns number of years on job
    public double getSalary(); // returns salary in dollars
    public int getID(); // returns unique employee ID number
}
```

The `Company` class is declared as follows.

```java
import java.util.ArrayList;

public class Company
{
    // minimum age, years on job, and salary needed to retire
    private final static int RETIRE_AGE   = 65;
    private final static int RETIRE_YEARS = 30;
    private final static double RETIRE_SALARY = 10000.0;

    private ArrayList myEmployees; // list of employees
    private double myTotalSalary;   // total salary of all employees

    // ... constructor and other methods not shown

    // precondition: claimants is sorted in ascending order by
    // employee ID, contains no duplicates, and each
    // element of claimants is in myEmployees

    // postcondition: returns true if emp is eligible to retire;
    // otherwise, returns false

    private boolean employeeIsEligible(Employee emp)
    {
        // you will write this function
    }
}
```
public void processRetirements(Employee[] claimants) {
    // you will write this function
}

Two class invariants must be maintained by Company objects:

- The instance variable myEmployees is sorted by employee ID.
- The instance variable myTotalSalary is the total of all employee salaries.

The Company constructor will establish these invariants as true initially. Each Company method must ensure that the invariants remain true after the method's execution.

Part A

An employee is eligible for retirement if (s)he meets at least two of the following requirements.

1. The employee is at least RETIRE_AGE years old.
2. The employee has worked for at least RETIRE_YEARS years.
3. The employee's salary is at least RETIRE_SALARY.

Write the private Company method employeeIsEligible, which is described as follows. Method employeeIsEligible returns a boolean value that indicates whether the employee represented by parameter emp is eligible for retirement, using the rules above.

Complete method employeeIsEligible below.

private boolean employeeIsEligible(Employee emp) {
    // postcondition: returns true if emp is eligible to retire;
    // otherwise, returns false
    }
**Part B**

( *Assume all import statements you need are made. You do not need to write import statements*).

Write the *Company* method `processRetirements`, which is described as follows. Method `processRetirements` has one parameter, `claimants` representing all employees that wish to retire. Assume `claimants` is sorted in ascending order by ID number, contains no duplicates, and that all elements in `claimants` are also in private instance variable `myEmployees`. Method `processRetirements` removes from `ArrayList` `myEmployees` only those employees listed in `claimants` that are eligible for retirement and maintains the two class invariants described above: the `ArrayList` is maintained in order by employee ID and `myTotalSalary` is the total of all salaries of the remaining employees.

In writing `processRetirements`, you may call method `employeeIsEligible`, specified in part (a). Assume that `employeeIsEligible` works as specified, regardless of what you wrote in part (a).

Complete method `processRetirements` below.

```java
public void processRetirements(Employee[] claimants) {

  // precondition: claimants is sorted in ascending order by
  // employee ID, contains no duplicates, and each
  // element of claimants is in myEmployees
  // postcondition: all retirement-eligible employees in claimants have been
  // removed from myEmployees; myEmployees remains
  // sorted by employee ID; myTotalSalary has been
  // updated to maintain invariant that it represents
  // total of all employee salaries

}
```

**Part C**

Assume that $N$ is the number of employees in the company. Give the best Big-Oh expression (in terms of $N$) for the worst-case running time for your implementation of the function `processRetirements`. Justify your answer with reference to the code you wrote in part (b). You will NOT receive full credit if you do not provide a justification.

---

*Owen L. Astrachan*
Last modified: Mon May 19 14:03:52 EDT 2003