AP® Computer Science AB
2003 Sample Student Responses

The data member `empList` is sorted in ascending order by employee ID. The total of all salaries is maintained in the data member `salaryBudget`.

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

1. The employee is at least `retireAge` years old.
2. The employee has worked for at least `retireYears`.
3. The employee's salary is at least `retireSalary`.

Write the `Company` member function `EmployeeIsEligible`, which is described as follows. `EmployeeIsEligible` returns true if `Employee emp` is eligible for retirement, using the rules described above.

Complete function `EmployeeIsEligible` below.

```
bool Company::EmployeeIsEligible(const Employee & emp) const
// postcondition: returns true if emp is eligible to retire;
//                otherwise, returns false
{
    if(emp.Age() >= retireAge)
    {   if(emp.YearsOnJob()>=retireYears ||
            emp.Salary() >= retireSalary)
            return true;
    }
    else if (emp.YearsOnJob() >= retireYears &&
            emp.Salary()>= retireSalary)
            return true;

    return false;
}
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

Complete function `ProcessRetirements` below.

```
void Company::ProcessRetirements(const apvector<Employee> & claimants)
// precondition:  claimants is sorted in ascending order
//                by employee ID with no duplicates;
//                all claimants are in empList
// postcondition: all eligible employees in claimants have been
//                removed from empList; empList has been resized to
//                reflect retirements;
//                empList remains sorted by employee ID;
//                salaryBudget has been updated to reflect remaining
//                employees
{   int i,j,k;
    int count=0;
    for (i=0; i<claimants.length(); ++i)
    {   if (EmployeeIsEligible(claimants[i])==true)
        {   count++;
            for(j=0; j<empList.length(); ++j)
            {
                if(claimants[i].Id()== empList[j].Id())
                {   salaryBudget -= empList[j].Salary();
                    for(k=j+1; k<empList.Length(); ++k)
                        empList[k-1]=empList[k];
                }   break;
            }
        }
    }
    empList.resize(empList.length() - count);
}
```

Part (c) begins on page 8.

GO ON TO THE NEXT PAGE.

(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. $O(N^2)$

In the worst case scenario, all employees submit claims for retirement and are all eligible.

Claimants.length() is N, so the for loop through the list of claimants is N iterations, when the correct Id is found using the j for loop, each employee has gone another N iterations. The moving up of the higher Id employees does not increase the iterations because the find Id goes from index 0 to j where $0 \le j < N$ and the rearrange forloop(k) goes from j+1 to empList.length(). With the break statement, the latter part of the j for loop is not executed. Summing these two partial loops gives a iteration of N. So with one loop of N inside another loop of N, the Big O notation is $O(N^2)$.

GO ON TO THE NEXT PAGE.

The data member `empList` is sorted in ascending order by employee ID. The total of all salaries is maintained in the data member `salaryBudget`.

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

1. The employee is at least `retireAge` years old.
2. The employee has worked for at least `retireYears`.
3. The employee's salary is at least `retireSalary`.

Write the `Company` member function `EmployeeIsEligible`, which is described as follows. `EmployeeIsEligible` returns true if `Employee emp` is eligible for retirement, using the rules described above.

Complete function `EmployeeIsEligible` below.

```
bool Company::EmployeeIsEligible(const Employee & emp) const
// postcondition: returns true if emp is eligible to retire;
//                otherwise, returns false
    int elig = 0;
    if (emp.Age() >= retireAge)
        elig++;
    if (emp.YearsOnJob() >= retireYears)
        elig++;
    if (emp.Salary() >= retireSalary)
        elig++;
    if (elig >= 2)
        return true;
    else
        return False;
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

Complete function `ProcessRetirements` below.

```
void Company::ProcessRetirements(const apvector<Employee> & claimants)
// precondition:  claimants is sorted in ascending order
//                by employee ID with no duplicates;
//                all claimants are in empList
// postcondition: all eligible employees in claimants have been
//                removed from empList; empList has been resized to
//                reflect retirements;
//                empList remains sorted by employee ID;
//                salaryBudget has been updated to reflect remaining
//                employees
```

```
for (int i = 0; i < claimants.length(); i++)
   if (EmployeeIsEligible(claimants[i])) {
      salaryBudget -= claimants[i].Salary();
      For (int x = i; x < claimants.length()-1; x++)
         claimants[x] = claimants[x+1];
      claimants.resize(claimants.length()-1);
      i--;
   }
```

Part (c) begins on page 8.

GO ON TO THE NEXT PAGE.

(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.

$O(N^2)$

the two nested for loops loop through the dada, causing it to be $O(N^2)$. If every employee was eligable to retire, every employee would need to be run through the nested for loops.

GO ON TO THE NEXT PAGE.

AB1 C

The data member `empList` is sorted in ascending order by employee ID. The total of all salaries is maintained in the data member `salaryBudget`.

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

1. The employee is at least `retireAge` years old.
2. The employee has worked for at least `retireYears`.
3. The employee's salary is at least `retireSalary`.

Write the `Company` member function `EmployeeIsEligible`, which is described as follows. `EmployeeIsEligible` returns true if `Employee emp` is eligible for retirement, using the rules described above.

Complete function `EmployeeIsEligible` below.

```
bool Company::EmployeeIsEligible(const Employee & emp) const
// postcondition: returns true if emp is eligible to retire;
//                otherwise, returns false
{

  if((emp.Age() == retireAge) && (emp.Salary() == retire Salary))
  {
    return true;
  }
  else if((emp.Age() = retireAge) &&(emp.YearsOnJob() = = retireYears))
  {
    return true;
  }
  else if((emp.Salary() == retireSalary) && (emp.YearsOnJob() == retireYears))
  {
    return true;
  }
  else
    return false;

}
```

Part (b) begins on page 6.

GO ON TO THE NEXT PAGE.

Complete function `ProcessRetirements` below.

```
void Company::ProcessRetirements(const apvector<Employee> & claimants)
// precondition:  claimants is sorted in ascending order
//                by employee ID with no duplicates;
//                all claimants are in empList
// postcondition: all eligible employees in claimants have been
//                removed from empList; empList has been resized to
//                reflect retirements;
//                empList remains sorted by employee ID;
//                salaryBudget has been updated to reflect remaining
//                employees
{
  int count=0;
  for(int x=0; x<claimantslength(); x++)
  {
    if( EmployeeIsEligible( claimants[x]))
    {
      salaryBudget -= claimants[x].Salary;
      for(int y=x; y<claimants.length()-1; y++)
      {
        emplist[y-count]= empList[y-count+1];
      }
      count++;
    }
  }
  count = empList.length() - count;
  empList.resize(count);
}
```

Part (c) begins on page 8.

GO ON TO THE NEXT PAGE.

(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.

The worst-case running time for the implementation of the function ProcessRetirements is O(N log N), because if the worst-case scenario is all retirements in empList, The function has to check each element, then if the employee is eligible, it has to step through the vector from that employee to the end, shifting everything left one, So in easier terms, it goes through the vector to search (N steps). Then if it finds a proper person to retire, it has to shift that part of the vector, every time it finds a person (log N steps). So this is why it takes O(N log N) steps.

GO ON TO THE NEXT PAGE.