Gebze Technical University
Computer Engineering

CSE 222 - 2018 Spring

HOMEWORK 02 REPORT

AHMED SEMİH ÖZMEKİK
171044039

1 INTRODUCTION

1.1 Problem Definition

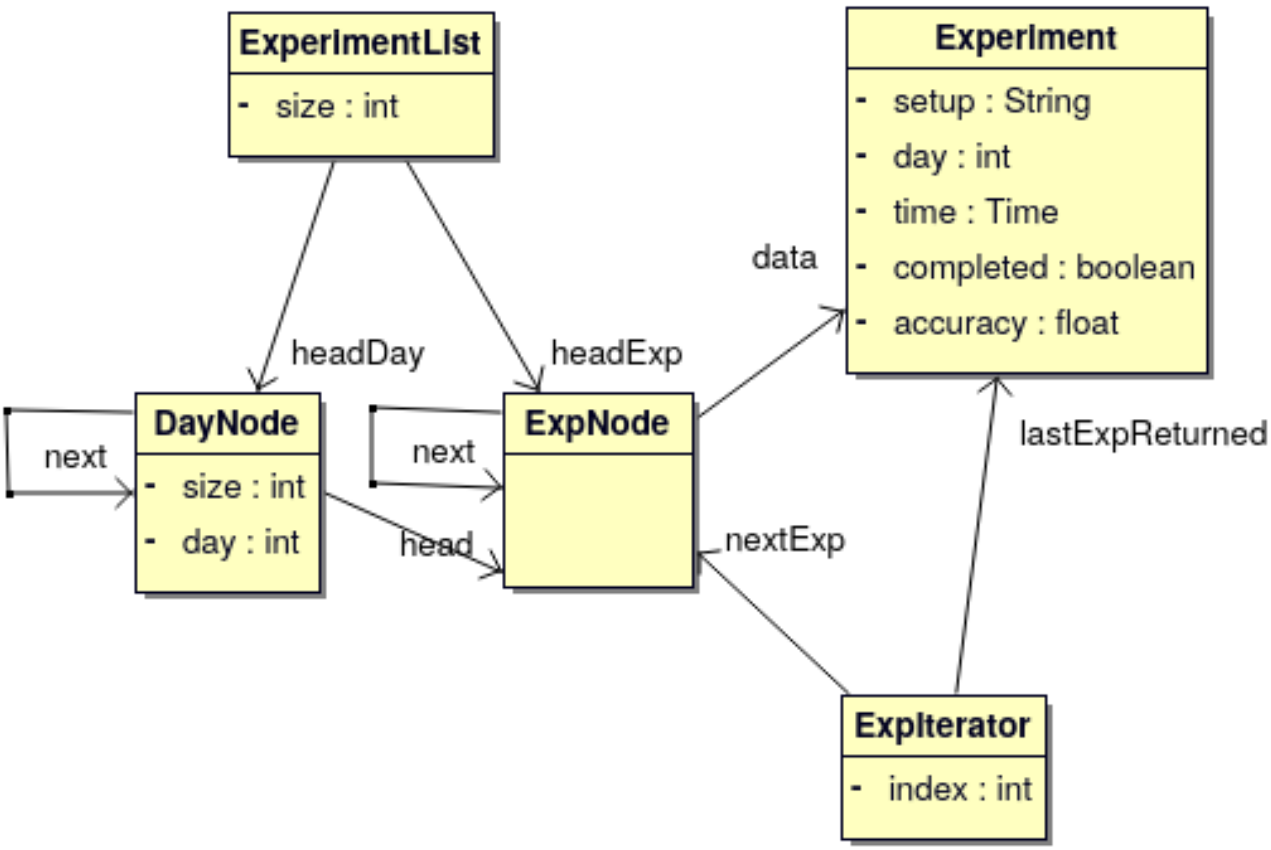
Problem is to keep track of some machine learning experiments. Those experiments have data fields about their day of start or the information about whether the experiment is completed or not. Of course, the problem is not just keep the experiments together, but to keep and track them due to their day of start information. In the absence of this requirements, problem becomes the necessity of a structure which can achieve this bookkeeping as it's finest.
In essence, our problem is to create an "Experiment List" structure satisfying such abilities mentioned above.

1.2 System Requirements

Before the requirement of a structure for to keep and track experiments, we first need an experiment structure which stores all the necessary information about the machine learning experiment itself. One must keep the experiment within this structure so that he or she can track them later on. So to say, we require Experiment Class. But of course, experiment is only the data and we need to keep them in some structure. And this data structure being a Linked-List structure. Therefore, we require an "Experiment List" which implements the basics of a single linked list to keep the experiments. And plus, in order to speed up add and remove operations we require an additional list structure which defined in the level of days.

2 METHOD

2.1 Class Diagrams



2.2 Problem Solution Approach

First of all, I have implemented and Experiment Class which has all the information about the machine learning experiment. This class carries the data portion about the experiment. And I have defined two nodes for to use in my ExperimentList structure. ExpNode was reasonable. And I added the DayNode as a bonus part to speed up the operations in the level of days. ExpNode has a data portion which is Experiment and pointer to the next ExpNode. ExperimentList stores one head pointer for to keep the ExpNode thread. For now, we can only iterate over the list within the ExpNode. To be able to achieve iterate the list in the level of days we need a DayNode which comes right in. DayNode has ExpNode head pointer in it which points the first ExpNode of that day and every DayNode has day index for indicating the number of day. And finally, DayNode has size field for to track the number of ExpNode in that day. Of course, all the experiments are connected via

ExpNodes but with the existence of DayNode we are being to able know each day's starting experiment node. ExperimentList stores one head pointer for the keep the DayNode thread in order to achieve iterating the list in the level of days.

2.3 Time Complexity of Methods

Table 1: Time Complexity of All Methods

Methods	**Time Complexity**
addExp()	$\mathcal{O}(\mathrm{n})$
getExp()	$\mathcal{O}(\mathrm{n})$
setExp()	$\mathcal{O}(\mathrm{n})$
removeExp()	$\mathcal{O}(\mathrm{n})$
listExp()	$\mathcal{O}(\mathrm{n})$
removeDay()	$\mathcal{O}(\mathrm{n})$
orderDay()	$\mathcal{O}(\mathrm{nlog}\, n)$
orderExperiments()	$\mathcal{O}(\mathrm{nlog}\, n)$
iterator()	$\mathcal{O}(1)$
Iterator.next()	$\mathcal{O}(1)$
Iterator.hasNext()	$\mathcal{O}(1)$
My Other Methods	
DayNode.add()	$\mathcal{O}(\mathrm{n})$
DayNode.addToEnd()	$\mathcal{O}(\mathrm{n})$
DayNode.lastExp()	$\mathcal{O}(\mathrm{n})$
DayNode.getExp()	$\mathcal{O}(\mathrm{n})$
DayNode.getDay()	$\mathcal{O}(\log n)$

3 RESULT

3.1 Test Cases

Case 1:

Before going any further, we must first create the Experiments and ExperimentList. And then add the experiments to the ExperimentList. This is our initialized list for now:

```
ExperimentList list = new ExperimentList();

Experiment[] expArray = new Experiment[5];

// Testing the addExp() method.
for (int i=0;i<expArray.length;++i){
    expArray[i] = new Experiment(i+1, "setup" + i, new Time(i));
    if(i%2 == 0){
        expArray[i].setCompleted(true);
        expArray[i].setAccuracy(i*i);
    }
    list.addExp(expArray[i]);
}

print("Starting to Test...");
```

We have a list now, then we can start to test. For this case, we will add some experiments which are unsorted. And they must exist in the right order after we add it.

```
print("Starting to Test...");
print("{  ExperimentList  }:");
show(list);

// Testing addExp() more.
list.addExp(new Experiment( day: 1, setup: "new0", new Time(1)));
list.addExp(new Experiment( day: 1, setup: "new1", new Time(2)));
list.addExp(new Experiment( day: 3,  setup: "new2", new Time(0)));
list.addExp(new Experiment( day: 1,  setup: "new3", new Time(0)));

print("{  ExperimentList  } + Experiments(new0, new1, new2, new3):");
show(list);
```

Case 2:

In this case, we are going to try to get the experiment for given day and index from the list:

```
// Testing getExp().
print("{  GET EXPERIMENT FROM LIST  } (Exp[day1, 3th], Exp[day5, 1st]):");
print(list.getExp( day: 1, index: 3).toString());
print(list.getExp( day: 5, index: 1).toString());
```

Case 3:
We set some experiments from the list for given day and index.

```
//Testing setExp().
print("{  SET EXPERIMENT FROM LIST  } (Exp[day3, 2nd], Exp[day1,4th])");
list.setExp( day: 3, index: 2, new Experiment( day: 3,  setup: "SET1", new Time(3)));
list.setExp( day: 1, index: 4, new Experiment( day: 1, setup: "SET2", new Time(4)));
show(list);
```

Case 4:
Before we test ordering the list, we set their accuracy not in order. And know, since we know that it's unsorted, we order day one and three.

```
//Testing orderDay().
int i=100;
for (Experiment e:list)
    e.setAccuracy(i-=11);
print("{ ExperimentList  } (Not in order)");
show(list);
list.orderDay(1);
list.orderDay(3);
print("{  ExperimentList  } (1st and 3th Days are in order)");
show(list);
```

Case 5:
In this case, we test for to order the whole list.

```
//Testing orderExperiments().
ExperimentList orderedList = list.orderExperiments();
print("{ OrderedList } (ExperimentList is in order.)");
show(orderedList);
```

Case 6:
We try to remove some experiments from the list.

```
//Testing removeExp().
list.removeExp( day: 4, index: 1);
list.removeExp( day: 1, index: 1);
list.removeExp( day: 3, index: 2);
list.removeExp( day: 3, index: 1);
print("{  ExperimentList  } - [day4, 1st], [day1, 1st], [day3, 2nd], [day3, 1st] ");

show(list);
```

Case 7:
After we complete some experiments from the list we try to list them with using proper method.

```
//Testing listExp()..
list.getExp( day: 1, index: 1).setCompleted(true);
list.getExp( day: 1, index: 2).setCompleted(true);
print("{  ExperimentList  } (Day 1 Completed Experiments:(1st, 2nd))");
list.listExp( day: 1);
```

Case 8:
In this case, we try to remove the first that from the list.

```
//Testing removeDay().
list.removeDay(1);
print("{  ExperimentList  } (Day 1 is removed)");
show(list);

print("Testing is Over...");
```

3.2 Running Results

Output Case 1:

```
[DEBUG]  Starting to Test...
[DEBUG]  {  ExperimentList  }:
[DEBUG]  Day[1]->{ Setup: setup0 }  (Accuracy: 0.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: setup2 }  (Accuracy: 4.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[4]->{ Setup: setup3 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 16.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  {  ExperimentList  } + Experiments(new0, new1, new2, new3):
[DEBUG]  Day[1]->{ Setup: setup0 }  (Accuracy: 0.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new0 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new1 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new3 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: setup2 }  (Accuracy: 4.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: new2 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[4]->{ Setup: setup3 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 16.0) (Completed: true)   (Time:02:00:00)
```

Output Case 2:

```
[DEBUG]  {  GET EXPERIMENT FROM LIST  } (Exp[day1, 3th], Exp[day5, 1st]):
[DEBUG]  Day[1]->{ Setup: new1 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 16.0) (Completed: true)   (Time:02:00:00)
```

Output Case 3:

```
[DEBUG]  {  SET EXPERIMENT FROM LIST  } (Exp[day3, 2nd], Exp[day1,4th])
[DEBUG]  Day[1]->{ Setup: setup0 }  (Accuracy: 0.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new0 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new1 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: SET2 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: setup2 }  (Accuracy: 4.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: SET1 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[4]->{ Setup: setup3 }  (Accuracy: -1.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 16.0) (Completed: true)   (Time:02:00:00)
```

Output Case 4:

```
[DEBUG]  { ExperimentList  } (Not in order)
[DEBUG]  Day[1]->{ Setup: setup0 }  (Accuracy: 89.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new0 }  (Accuracy: 78.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new1 }  (Accuracy: 67.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: SET2 }  (Accuracy: 56.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: 45.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: setup2 }  (Accuracy: 34.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: SET1 }  (Accuracy: 23.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[4]->{ Setup: setup3 }  (Accuracy: 12.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 1.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  {  ExperimentList  } (1st and 3th Days are in order)
[DEBUG]  Day[1]->{ Setup: SET2 }  (Accuracy: 56.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new1 }  (Accuracy: 67.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new0 }  (Accuracy: 78.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: setup0 }  (Accuracy: 89.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: 45.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: SET1 }  (Accuracy: 23.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: setup2 }  (Accuracy: 34.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[4]->{ Setup: setup3 }  (Accuracy: 12.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 1.0) (Completed: true)   (Time:02:00:00)
```

Output Case 5:

```
[DEBUG]  { OrderedList } (ExperimentList is in order.)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 1.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[4]->{ Setup: setup3 }  (Accuracy: 12.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: SET1 }  (Accuracy: 23.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[3]->{ Setup: setup2 }  (Accuracy: 34.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: 45.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: SET2 }  (Accuracy: 56.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new1 }  (Accuracy: 67.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new0 }  (Accuracy: 78.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: setup0 }  (Accuracy: 89.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  {  ExperimentList  } - [day4, 1st], [day1, 1st], [day3, 2nd], [day3, 1st]
```

Output Case 6:

```
[DEBUG]  {  ExperimentList  } - [day4, 1st], [day1, 1st], [day3, 2nd], [day3, 1st]
[DEBUG]  Day[1]->{ Setup: new1 }  (Accuracy: 67.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: new0 }  (Accuracy: 78.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[1]->{ Setup: setup0 }  (Accuracy: 89.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: 45.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 1.0) (Completed: true)   (Time:02:00:00)
```

Output Case 7:

```
[DEBUG]  {  ExperimentList  } (Day 1 Completed Experiments:(1st, 2nd))
Day[1]->{ Setup: new1 }  (Accuracy: 0.0) (Completed: true)   (Time:02:00:00)
Day[1]->{ Setup: new0 }  (Accuracy: 0.0) (Completed: true)   (Time:02:00:00)
Day[1]->{ Setup: setup0 }  (Accuracy: 89.0) (Completed: true)   (Time:02:00:00)
```

Output Case 8:

```
[DEBUG]  {  ExperimentList  } (Day 1 is removed)
[DEBUG]  Day[2]->{ Setup: setup1 }  (Accuracy: 45.0) (Completed: false)   (Time:02:00:00)
[DEBUG]  Day[5]->{ Setup: setup4 }  (Accuracy: 1.0) (Completed: true)   (Time:02:00:00)
[DEBUG]  Testing is Over...
```

Not: I wanted to write my calculations of time complexities in here, but trying to write equations of the calculations was almost impossible in Libre Office. Instead, I wrote them in Latex and added to this file. So, for the full report there might a small inconsistency with margins and font etc. But I had no other choice. I have given all the details about the calculations of time complexities of methods, starting from next page and continues until the end of the report.

1. addExp(Experiment): Insert experiment to the end of the day.

```
public void addExp(Experiment exp)
{
    ++size;
    if (headExp == null) {
        headExp = new ExpNode(exp);
        headDay = new DayNode(headExp);
    }
    else // DayNode handles the details.
        headDay.add(exp);
}
```

Figure 1: addExp()

$$
\begin{aligned}
&T(n) = max(T_{if}(n), T_{else}(n)) && T_{if}(n) < T_{if}(n) \\
&T(n) = T_2(n) && T_2(n) = T_{headDay.add()}(n) \\
&T(n) = \mathcal{O}(n) && (15)
\end{aligned} \tag{1}
$$

2. getExp(day,index): Get the experiment with the given day and position.

```
public Experiment getExp(int day, int index)
{
    return headDay.getDay(day).getExp(index).data;
}
```

Figure 2: getExp()

$$
\begin{aligned}
&T(n) = T_{getDay}(n) + T_{getExp()}(n) \\
&T(n) = \log n + n && (18, 19) \\
&T(n) = \mathcal{O}(n)
\end{aligned} \tag{2}
$$

3. setExp(day, index, Experiment): Set the experiment with the given day and position.

```
public void setExp(int day, int index, Experiment newExp)
{
    ExpNode exp = headDay.getDay(day).getExp(index);
    if (day != newExp.getDay())
        throw new IllegalArgumentException("Experiment's day conflicts!");
    exp.data = newExp;
}
```

Figure 3: setExp()

$$
\begin{aligned}
T(n) &= T_{getDay()}(n) + T_{getExp()}(n) + 2 \\
T(n) &= \log n + n + 2 \qquad (18, 19) \\
T(n) &= \mathcal{O}(n)
\end{aligned} \tag{3}
$$

4. removeExp(day, index): Remove the experiment specified as index from given day.

```
public Experiment removeExp(int day, int index)
{
    ExpNode temp = headDay.getDay(day).getExp(index); // Get the element we want to remove.

    if (headDay.getDay(day).size == 1) { // Day will be removed.
        if (headDay.day == day){ // Head will be changed.
            headDay = headDay.next;
            headExp = headExp.next;
        }
        else { // Find the previous DayNode and ExpNode.
            DayNode prevDay = headDay.getDay(day-1);
            ExpNode prev = prevDay.getExp(prevDay.size);

            prevDay.next = prevDay.next.next; // Remove the day.
            prev.next = prev.next.next; // Remove the experiment.
        }
    }
  else if(index == 1) { // Day won't be removed, but the head exp will change.
      if (headDay.day == day){ // Change the heads.
          headDay.head = headDay.head.next;
          headExp = headExp.next;
          --headDay.size;
      }
      else { // Find the previous DayNode and ExpNode.
          --headDay.getDay(day).size;

          DayNode prevDay = headDay.getDay(day-1);
          ExpNode prev = prevDay.getExp(prevDay.size);
          // Remove the exp.
          prevDay.lastExp().next = prevDay.head.next;
          prev.next = prev.next.next;
      }
  }
  else{ // Again, find the previous ExpNode.
      ExpNode prev = headDay.getDay(day).getExp( index: index-1);

      --headDay.getDay(day).size;
      prev.next = prev.next.next;  // Remove the experiment.
  }

  --size;
  return temp.data;
```

Figure 4: removeExp()

$$
T(n) = T_{getDay()}(n) + T_{getExp()}(n) + max(T_{if}(n), T_{elseif}(n), T_{else}(n)) \tag{4}
$$

$$
\begin{aligned}
T_{if}(n) &= T_{getDay()}(n) + max(T_{if.if}(n), T_{if.else}(n)) \quad T_{if.if}(n) < T_{if.else}(n) \\
T_{if}(n) &= T_{getDay()}(n) + T_{if.else}(n) \\
T_{if.else}(n) &= T_{getDay()}(n) + T_{getExp()}(n) + 2 \\
T_{if}(n) &= 2T_{getDay()}(n) + T_{getExp()}(n) + 2 \\
T_{if}(n) &= 2\log n + n + 2 \qquad (18, 19) \\
T_{if}(n) &= \mathcal{O}(n)
\end{aligned} \tag{5}
$$

$$
\begin{aligned}
T_{elseif}(n) &= 1 + max(T_{elseif.if}(n), T_{elseif.else}(n)) \\
T_{elseif.if}(n) &= 3 \\
T_{elseif.else}(n) &= T_{getDay()}(n) + T_{getExp()}(n) + 3 \\
T_{elseif}(n) &= 1 + T_{elseif.else}(n) \qquad T_{elseif.if}(n) < T_{elseif.else}(n) \\
T_{elseif}(n) &= 1 + T_{getDay()}(n) + T_{getExp()}(n) \\
T_{elseif}(n) &= 1 + \log n + n \qquad (18, 19) \\
T_{elseif}(n) &= \mathcal{O}(n)
\end{aligned} \tag{6}
$$

$$
\begin{aligned}
T_{else} &= 2T_{getDay()}(n) + T_{getExp()}(n) + 2 \\
T_{else} &= 2\log n + n + 2 \qquad (18, 19) T_{else} = \mathcal{O}(n)
\end{aligned} \tag{7}
$$

$$
\begin{aligned}
T_{if}(n) &= T_{else}(n) > T_{elseif}(n) \qquad (5, 6, 7) \\
T(n) &= T_{getDay()}(n) + T_{getExp()}(n) + T_{if} \qquad (4) \\
T(n) &= n \qquad (5) \\
T(n) &= \mathcal{O}(n)
\end{aligned} \tag{8}
$$

5. listExp(day): List all completed experiments in a given day.

```java
public void listExp(int day)
{
    DayNode theDay = headDay.getDay(day);
    ExpNode exp = theDay.head;

    for (int i=0;i<theDay.size;++i){
        if (exp.data.isCompleted())
            System.out.println(exp.data.toString());
        exp = exp.next;
    }

}
```

Figure 5: listExp()

$$
\begin{aligned}
T(n) &= T_{getDay}(n) + 3n \\
T(n) &= \log n + 3n \qquad (18) \\
T(n) &= \mathcal{O}(n)
\end{aligned} \tag{9}
$$

6. removeDay(day): Remove all experiments in a given day.

```
public void removeDay(int day)
{
    if (headDay.day == day){ // Head day will be removed.
        headExp = headDay.lastExp().next;
        headDay = headDay.next;
    }

    else{
        // Get the previous day and the Day.
        DayNode prev = headDay.getDay(day-1);
        DayNode theDay = headDay.getDay(day);

        // Connect nodes and delete the Day.
        prev.lastExp().next =  theDay.lastExp().next;
        prev.next = theDay.next;

    }
}
```

Figure 6: removeDay()

$$
\begin{aligned}
T(n) &= 1 + max(T_{if}(n), T_{else}(n)) \\
T_{if}(n) &= T_{lastExp()} + 1 && (10.1) \\
T_{else}(n) &= 2T_{getDay()}(n) + T_{lastExp()}(n) + 1 && (10.2) \\
T(n) &= 1 + T_{else} && T_{if}(n) < T_{else}(n)(10.1, 10.2) \qquad (10) \\
T(n) &= 2 + 2T_{getDay()}(n) + T_{lastExp()}(n) \\
T(n) &= 2 + 2\log n + n && (18, 19) \\
T(n) &= \mathcal{O}(n)
\end{aligned}
$$

7. orderDay(day): Sorts the experiments in a given day according to the accuracy, the changes will be done on the list.

```
public void orderDay(int day)
{
    DayNode theDay = headDay.getDay(day);
    Experiment[] expData = new Experiment[theDay.size];

    for (int i=0;i<expData.length;++i) // Get the experiments to array.
        expData[i] = theDay.getDay(day).getExp( index: i+1).data;

    Arrays.sort(expData);

    for (int i=0;i<expData.length;++i) // Set the experiment from array.
        setExp(day, index: i+1,expData[i]);

}
```

Figure 7: orderDay()

$$
\begin{aligned}
T(n) &= T_{getDay()}(n) + 1 + n(T_{getExp(n)}) + T_{Arrays.sort()}(n) + T_{setExp()}(n) \\
T(n) &= \log n + 1 + n + n\log n + n && (3, 18, 19) \\
T(n) &= \mathcal{O}(n\log n)
\end{aligned}
$$

(11)

8. orderExperiments(): sorts all the experiments in the list according to the accuracy, the original list should not be changed since the day list may be damage

```
public ExperimentList orderExperiments()
{
    ExperimentList orderedList = new ExperimentList();
    Experiment[] expData = new Experiment[size];


    int i=0; // Iterate the list and get the experiments.
    for (Experiment e : this)
        expData[i++] = e;

    Arrays.sort(expData);

    orderedList.headDay = null; // Since the day order is corrupted, no need.
    orderedList.size = expData.length;

    ExpNode expNode = new ExpNode(expData[0]);
    orderedList.headExp = expNode; // Connect to head.
    for (i=1;i<expData.length;++i){ // Set the experiments to connected exp nodes.
        expNode.next = new ExpNode(expData[i]);
        expNode = expNode.next;
    }

    return orderedList;
```

Figure 8: orderExperiments()

$$
\begin{aligned}
T(n) &= 3 + n + T_{Arrays.sort()}(n) + 4 + n(2) + 1 \\
T(n) &= 8 + 3n + T_{Arrays.sort()}(n) \qquad\qquad T_{Arrays.sort()} \in \mathcal{O}(n \log n) \\
T(n) &= 8 + 3n + n \log n \\
T(n) &= \mathcal{O}(n \log n)
\end{aligned}
\tag{12}
$$

9. DayNode.add(Experiment): Inserts the given experiment to the appropriate place among days.

```
private void add(Experiment exp)
{
    // First, create the Experiment Node.
    ExpNode newExp = new ExpNode(exp);

    // Case: Experiment's day is this node's day.
    if (day == exp.getDay())
        addToEnd(newExp);

    // Case: Experiment's day is another next node's day.
    // There is no another next day :  (day2-0) -> (day2-day3)
    else if(next == null)
    {
        next = new DayNode(newExp);
        lastExp().next = next.head; // connecting the experiment of different days.
    }
    // The day doesn't exist and must be created in between days.
    // (day2-0-day4) -> (day2-day3-day4)
else if(next.day > exp.getDay())
{
    DayNode newDay = new DayNode(newExp);
    newDay.next = next;
    newDay.head.next = next.head;
    next = newDay;
    lastExp().next = newDay.head;
}
else
    next.add(exp);
```

Figure 9: DayNode.add()

$$
\begin{aligned}
T(n) &= 3 + max(T_{if}(n), T_{elseif1}(n), T_{elseif2}(n), T_{else}(n)) \\
T_{if}(n) &= T_{addToEnd()}(n) \\
T_{if}(n) &= \mathcal{O}(n) \qquad (16) \\
T_{elseif1(}(n) &= 2 \\
T_{elseif2}(n) &= 5 \\
T_{else}(n) &= T_{else()}(n-1)
\end{aligned} \tag{13}
$$

$$
\begin{aligned}
&T_{else}(n) = T_{else}(n-1) \\
&\quad a = 1 \qquad b = 1 \qquad d = 0 \\
&\quad a = b^d \qquad T_{else}(n) \in \mathcal{O}(\log n) \\
&T_{else}(n) = \mathcal{O}(\log n)
\end{aligned} \tag{14}
$$

$$
\begin{aligned}
&T(n) = 3 + T_{if}(n) \quad (T_{if}(n) > T_{else}(n) > T_{elseif2}(n) > T_{elseif2}(n))(13, 14) \\
&T(n) = \mathcal{O}(n)
\end{aligned} \tag{15}
$$

10. DayNode.addToEnd():Inserts the specified experiment node end of the day node.

```
private void addToEnd(ExpNode expNode)
{
    // Get the last experiment of the day.
    ExpNode last = lastExp();

    // Connect.
    expNode.next = last.next;
    last.next = expNode;
    ++size;
}
```

Figure 10: DayNode.addToEnd()

$$\begin{aligned} T(n) &= T_{lastExp()}(n) + 3 \\ T(n) &= n + 3 \qquad (23) \\ T(n) &= \mathcal{O}(n) \end{aligned} \tag{16}$$

11. DayNode.lastExp(): Returns the last experiment of the day.

```
private ExpNode lastExp()
{
    // Shift the exp. to the end of the day.
    ExpNode temp = head;
    while(temp.next != null && temp.next.data.getDay() == day)
        temp = temp.next;
    return temp;
}
```

Figure 11: DayNode.lastExp()

$$\begin{aligned} T(n) &= 3 + n \\ T(n) &= \mathcal{O}(n) \end{aligned} \tag{17}$$

12. DayNode.getExp(): Returns the index'th experiment of the day.

```
private ExpNode getExp(int index)
{
    // Shift the exp. to the index.
    ExpNode temp = head;

    for (int i=1;i<index && temp != null;++i) // Shift the exp to experiment index.
        temp = temp.next;

    if (temp == null || temp.data.getDay() != day) // Experiment does not exist.
        throw new IndexOutOfBoundsException();

    return temp;

}
```

Figure 12: DayNode.getExp()

$$
\begin{aligned}
T(n) &= 1 + n + 2 \\
T(n) &= \mathcal{O}(n)
\end{aligned}
\tag{18}
$$

13. DayNode.getDay(): Search the given day among the days.

```
private DayNode getDay(int day)
{

    if (this.day == day)
        return this;
    else if (next == null)
        throw new IndexOutOfBoundsException();
    else
        return next.getDay(day);
}
```

Figure 13: DayNode.getDay()

$$
\begin{aligned}
&T(n) = 2 + max(T_{if}(n), T_{elseif}(n), T_{else}(n)) \\
&T(n) = T_{else}(n) && (T_{if}(n) = T_{elseif}(n) < T_{else}(n)) \\
&T(n) = T(n-1) \\
&\quad a = 1 && b = 1 && d = 0 \\
&\quad a = b^d && T(n) \in \mathcal{O}(\log n) \\
&T(n) = \mathcal{O}(\log n)
\end{aligned}
\tag{19}
$$

14. iterator(): Returns an iterator over the list.

```
public Iterator<Experiment> iterator() {
    return new ExpIterator( i: 0);
}
```

Figure 14: iterator()

$$
\begin{aligned}
T(n) &= T_{ExpIterator()}(0) \\
T(n) &= 1 \\
T(n) &= \mathcal{O}(1)
\end{aligned}
\tag{20}
$$

15. ExpIterator(): Returns the experiment iterator over the experiment list.

```
public ExpIterator(int i) {
    if (i < 0 || i > size) // Out of Boundaries.
        throw new IndexOutOfBoundsException("Invalid index: " + i);
    lastExpReturned = null;
    if (i == size) {
        index = size;
        nextExp = null;
    } else {
        nextExp = headExp;
        for (index = 0; index < i; ++index)
            nextExp = nextExp.next;
    }
}
```

Figure 15: ExpIterator()

$$
\begin{aligned}
T(n) &= 1 + T_{else} \quad (T_{else} > T_{if}) \\
T(n) &= 1 + 1 + n \\
T(n) &= \mathcal{O}(n)
\end{aligned}
\tag{21}
$$

16. ExpIterator.next(): Returns the next experiment and increments the iterator.

```
public Experiment next() {
    lastExpReturned = nextExp.data;
    nextExp = nextExp.next;
    return lastExpReturned;
}
```

Figure 16: ExpIterator.next()

$$
\begin{aligned}
T(n) &= 3 \\
T(n) &= \mathcal{O}(1)
\end{aligned}
\tag{22}
$$

17. ExpIterator.hasNext(): Informs the iterator has next element or not.

```
public boolean hasNext() { return nextExp != null; }
```

Figure 17: ExpIterator.hasNext()

$$
\begin{aligned}
T(n) &= 1 \\
T(n) &= \mathcal{O}(1)
\end{aligned}
\tag{23}
$$