자료구조 실습 보고서

[제 10주]

제출일: 19.05.22
학번/이름: 201603867/조성환

1. 프로그램 설명서

1) 주요 알고리즘/ 자료구조/ 기타

정렬 알고리즘의 결과를 검증.

2) 함수 설명서

.main()
appController.run() 함수를 통해 프로그램을 실행한다.

AppController

.run()
프로그램의 시작을 알리는 출력문과 함께 정렬 결과의 검증을 나타내는 출력을 함.
this.validateWith ascending, descending, random orderList() 함수를 차례로 실행시킨 후 마지막으로 끝내는 출력문과 함께 마침.

.validateWithAscendingOrderList()
앞서 서술한 ascending, descending, random 세 가지 함수가 모두 같은 순서로 진행함.
setListOrder(), setList()를 통해 해당 List의 종류와 길이를 정한 뒤 showFirstPartOfDataList()함수를 통해 앞 5개의 요소를 나열함.
마지막으로 validateSortsAndShowResult()함수를 통해 검증을 한 뒤 결과를 출력함.

setListOrder()
ListOrder enum class에서 정의한 order를 맞춰줌. 오름차, 내림차, 무작위가 있으며, ListOrder의 종류에 따라 List가 정해짐. ListOrder.~로 정해줌.

setList()
list를 생성함.
DataGenerator.~List(개수)를 통해 리스트를 생성함. ~는 3가지 중 한 종류이며, 개수는 해당 개수의 크기로 list를 만듬.

showFirstPartOfDataList()
이 함수를 실행함을 출력함.
for문을 통해 FIRST_PART_SIZE만큼 돌림. AppView.output()을 통해 list의 I번째를 출력함
ouputLine을 통해 줄바꿈을 함

validateSortsAndShowResult()
validateSort()를 통해 해당 정렬을 성공했는지 확인함.
성공 여부를 출력함.

validateSort()
Integer 배열에 copyList()의 결과를 넣음
주어진 sort의 종류에 따라 .sort()를 진행함.

asort와 list를 validationMessage()에 넣음.

copyList()
Integer배열을 입력한 리스트의 길이만큼 생성한 뒤 for문을 통해 복사를 함.
해당 배열을 리턴함.

showValidationMessage()
Appview.output()에서 listOrder().orderName(). aSort.getClass().getSimpleName()을 출력함.
.orderName()은 오름차, 내림차, 무작위 중 하나를 의미하며,
getClass.getSimpleName은 class의 의미를 가져옴.
여기서 가져오는 class의 이름은 insertion과 quicksort 중 하나이므로, 둘 중에 어떤 정렬을 했는지를 의미함.
만약 sortedListIsValid(aList)가 true면 통과를 했으므로 올바릅니다를 출력,
false면 통과를 못했으므로 틀렸습니다를 출력함.

sortedListValid()
for을 aList의 길이만큼 돌림. 만약 compareTo를 통해 I번째가 I+1번째 보다 크다면 return false를 함. 통과를 하면 true를 반환함.

list를 선언 this.list를 통해 참조

sortby score에서
maxLoc를 선언
size만큼 돌리면서 최대 위치를 찾음
maxloc위치에 있는놈을 마지막으로 보냄.(swap)
이후 나머지를 reculsively 함
reculsive 내에서
left가 right보다 커질 때 까지
mid를 partition을 통해 구함
partition의 pivot은 left
toRight은 left
toLeft는 right+1

ListOrder
enum 클래스로 Ascending, Descending, Random 만듬.
Order_Names 배열에 각 오름차순 내림차순 무작위를 넣음
orderName()
ListOrder.ODER_NAMES[this.ordinal()]를 통해 한글 이름을 가져옴

Sort

sort()
list를 정렬함. 6주차 과제를 응용해서 작성함.
만약 list의 사이즈가 유효하지 않다면 false를 리턴함
size만큼 for문을 돌림. for문을 통해 가장 큰 요소를 가진 위치를 가져옴.
가져오면 맨 마지막에 가장 큰 요소를 넣고, 0번째부터 뒤에서2번째 위치까지를 quicksortreculsively에 집어넣음.

swap()
E를 하나 만든 후 aList의 I번째를 넣은 뒤 I번째에 j를 넣고, j번째에 E를 집어넣어 서로 바꿔줌.

quickSortRecursively()
left가 right보다 작으면 mid를 partition()을 통해 계산한 후 left에서 mid-1까지와 mid+1에서 right까지 quickRecursively()를 통해 재귀적으로 정렬을 해줌.

partition()
pivot을 설정한 뒤 toRight에 left, toLeft에 right+1을 넣어 do while을 돌림.

Data Generator

ascendingList()
list를 만든 후 size만큼 for문을 돌림. I번째에 I를 차례로 집어넣음
list를 반환함

descendingList()
list를 만든 후 size만큼 for문을 돌림. I번째에 size-1을 넣은 후 차례로 - -를 통해 역순으로 집어넣음
list를 반환함

randomList()
list를 만든 후 size만큼 for문을 돌림. I번째에 I를 차례로 집어넣음. 이후 Random객체를 생성한 뒤. size만큼 for문을 만들어 size만큼 random을 돌림. temp에 I번째 리스트를 가져옴. I번째에 r번째 값을 집어넣은 후 r번째에 temp를 집어넣음.
list를 반환함

InsertionSort

sort()
본 함수를 통해 정렬을 시작함.
size가 유효한지 검사한 후 틀리면 false를 리턴함
유효하면 minLoc를 선언한 뒤 1번째부터size의 크기만큼 for문을 돌림

해당 위치의 요소가 minLoc위치의 요소보다 작으면 minLoc를 해당 위치로 바꿈
for문이 끝난 후 swap()을 통해 minLoc의 요소를 0번째로 보냄
for문을 2번째부터 size의 크기 만큼 돌림
aList가 E의 배열임. E를 하나 선언한 후 해당 위치의 요소를 넣음
insertionLoc를 I-1로 초기화함. insertionLoc위치에 있는 요소와 I에 위치한 요소를 비교해 insertionLoc에 위치한 요소가 더 크면 insertionLoc+1위치에 insertionLoc의 요소를 넣고 insertionLoc를 1빼줌. 이런식으로 계속 전에있는 요소가 현재 삽입하려는 요소보다 크다면 뒤로 보내줘서 공간을 만듬. 마지막으로 while문을 빠져나가면 insertionLoc는 insertedElement가 들어가야할 위치 전의 숫자임. +1위치에 insertedElement를 넣어줌.
이런식을 hahems 요소를 정렬해준 뒤 true를 리턴함.

swap()
E를 만들어 I위치의 요소를 담음. I위치에 j위치 요소를 넣고 j위치에 E에 담은 I요소를 넣음

QuickSort

sort()
size가 유효한지 검사한 후 틀리면 false를 리턴함
유효하면 maxLoc를 선언한 뒤 for문을 1부터 size만큼 돌림
aList의 I번째와 maxLoc의 위치와 비교한 후 I가 더 크면 maxLoc를 I로 초기화함
끝나면 swap을 통해 maxLoc를 맨 마지막에 보냄
quickSortRecursively()에 0번째와 뒤에서 두 번째까지를 넣은 후 재귀적으로 정렬이 되도록 함. 마지막으로 true를 반환함

swap()
앞서 설명한 내용

quickSortRecursively()
left가 right보다 작으면 mid를 partition을 통해 구함.
quickSortRecursively에 left에서 mid-1, mid+1에서 right까지를 2개 해서 넣음
이제 재귀적으로 1개단위까지 쪼개진 뒤 다시 모두 정렬이 됨.

partition()
pivot을 기준으로 양쪽을 나눔
pivot을 left로 초기화함. toRight는 left, toLeft는 right로 가야할 위치를 정함
do while을 통해 toRight에 위치한 요소가 클때까지 toRight을 ++, toLeft에 위치한 요소가 pivot보다 작을 때 까지 toLeft를ㅡ함 만약 toRight이 toLeft보다 작으면 toRight와 toLeft를 swap()함 이는 toRight가 toLeft보다 클때까지 반복함.
마지막으로 left를 toLeft와 swap한 뒤 toLeft를 반환함.

pivot()
left를 리턴함

3) 종합 설명서

결국 dataGenerator에서 각 순서에 맞게 list를 생성함. 이후 sort를 통해 insertion, quick으로 정렬을 한 후 validateSort를 통해 정렬이 제대로 됐는지를 확인. 정렬이 제대로 됐다면 성공을 출력하는 형식으로 프로그램이 진행한다.

insertionSort는 I번째서부터 쭉내려가면서 현재 위치보다 큰 요소가있는지를 계속 확인 후 적당한 위치에 집어넣음

quickSort는 계속 partition을 통해 쪼개면서 재귀적으로 정렬을 함.

성능 비교는 없고 성공 여부만 알림

2. 프로그램 장단점/ 특이점 분석

insertion은 순서대로 진행하기 때문에 상황에 따라 효율적이겠지만 개수가 많아지고 값이 무작위로 입력되면 비효율적임. 예를들어 개수가 5개이고 순서대로 입력되어있으면 insertion이 더 빠르겠지만 100개만 넘어가도 앞에서 진행하는 것보다 재귀적으로 partition을 통해 나눠가면서 정렬을 하는 것이 더 빠를 수 있기 때문에 상황에따라 어떤 정렬을 사용할지 잘 결정하는 것이 중요함을 느낌.

저번 과제처럼 time을 사용해 시간을 비교해도 괜찮지 않았을까 하는 아쉬움이 있음.

단점 testSize가 너무 커서 그런지 가끔 stackOverflow가 발생함. 이는 해결하지 못했음. 만약 stackoverflow가 발생하면 다시 실행하면 됨.

3. 실행 결과 분석

1) 입력과 출력

입력 x

```
<<< 정렬 알고리즘의 정렬 결과를 검증하는 프로그램을 시작합니다 >>>

> 정렬 결과의 검증:

[오름차순 리스트] 의 앞 부분: 0 1 2 3 4
[오름차순 리스트]를 [InsertionSort] 한 결과는 올바릅니다.
[오름차순 리스트]를 [QuickSort] 한 결과는 올바릅니다.

[내림차순 리스트] 의 앞 부분: 9999 9998 9997 9996 9995
[내림차순 리스트]를 [InsertionSort] 한 결과는 올바릅니다.
[내림차순 리스트]를 [QuickSort] 한 결과는 올바릅니다.

[무작위 리스트] 의 앞 부분: 8873 5420 5960 7696 7987
[무작위 리스트]를 [InsertionSort] 한 결과는 올바릅니다.
[무작위 리스트]를 [QuickSort] 한 결과는 올바릅니다.

<<< 정렬 알고리즘의 정렬 결과를 검증하는 프로그램을 종료합니다 >>>
```

2) 결과 분석

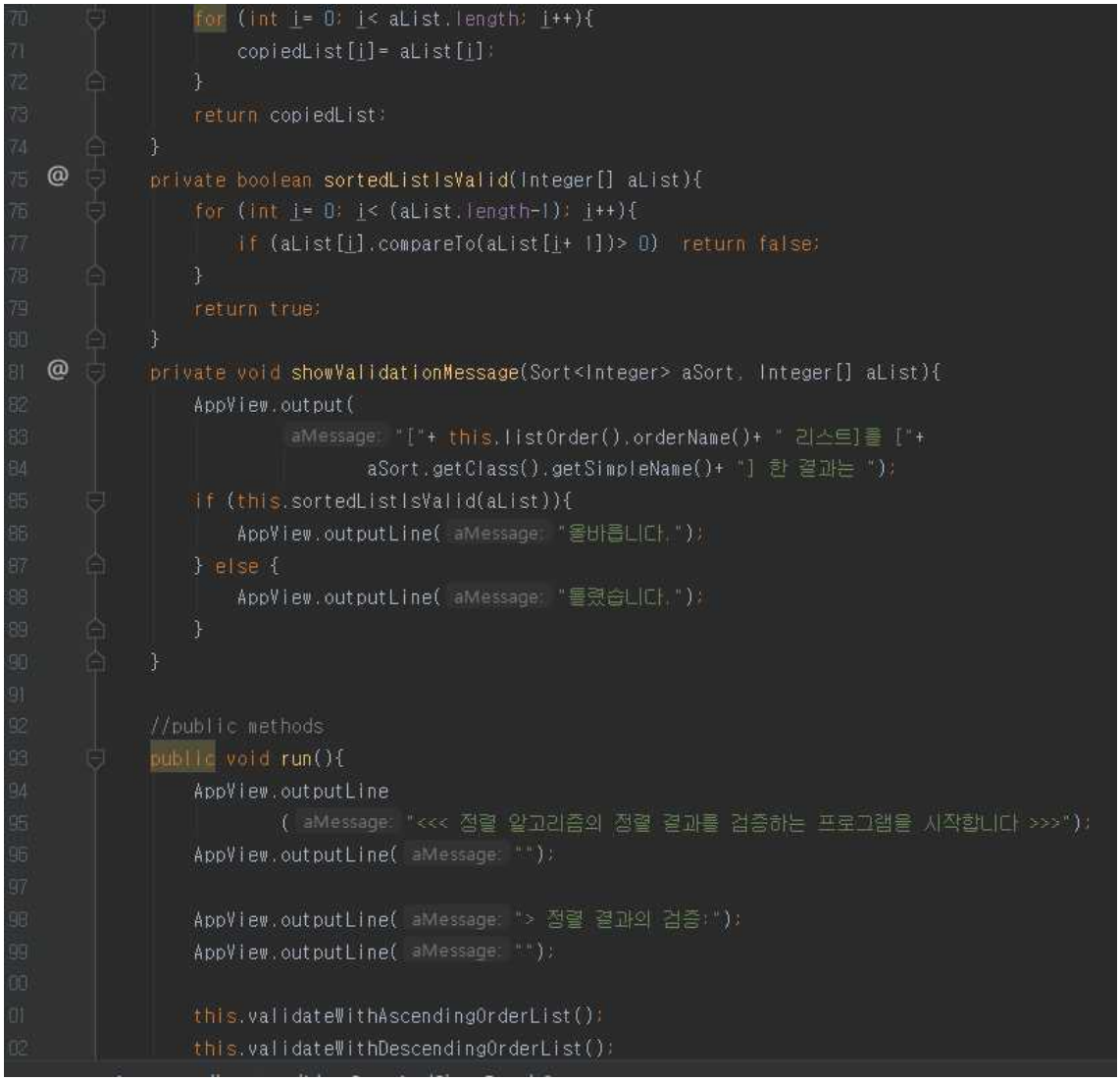
정상적으로 실행이됨.

4. 소스 코드

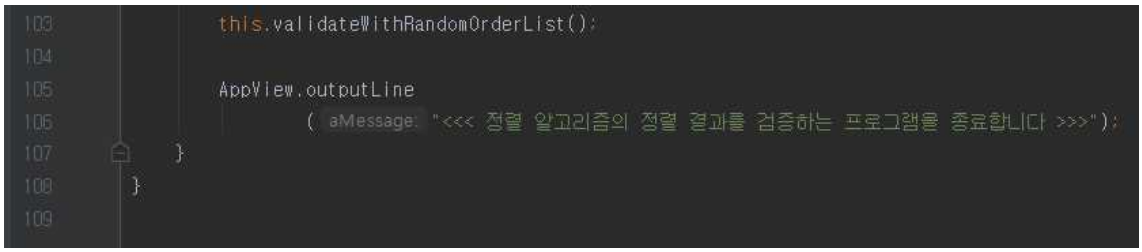
AppController

```
package controller;

import model.DataGenerator;
import model.InsertionSort;
import model.QuickSort;
import model.Sort;
import view.AppView;

public class Appcontroller {
    //constants
    private static final int TEST_SIZE= 10000;
    private static final int FIRST_PART_SIZE= 5;
    private static final InsertionSort<Integer> INSERTION_SORT= new InsertionSort<~>();
    private static final QuickSort<Integer> QUICK_SORT= new QuickSort<~>();

    //private var
    private Integer[] _list;
    private ListOrder _listOrder;

    //getter setter
    public Integer[] list() { return _list; }
    public void setList(Integer[] newList) { this._list = newList; }

    public ListOrder listOrder() { return _listOrder; }
    public void setListOrder(ListOrder newListOrder) { this._listOrder = newListOrder; }

    //constructor
    public Appcontroller(){}

    //private methods
    private void validateWithAscendingOrderList(){
        this.setListOrder(ListOrder.Ascending);
        this.setList(DataGenerator.ascendingList(Appcontroller.TEST_SIZE));
```

```
40          this.showFisrtPartOfDataList();
41          this.validateSortsAndShowResult();
42      }
43      private void validateWithRandomOrderList(){
44          this.setListOrder(ListOrder.Random);
45          this.setList(DataGenerator.randomList(Appcontroller.TEST_SIZE));
46          this.showFisrtPartOfDataList();
47          this.validateSortsAndShowResult();
48      }
49      private void showFisrtPartOfDataList(){
50          AppView.output
51                  ( aMessage: "["+ this.listOrder().orderName()+ " 리스트] 의 앞 부분: ");
52          //작성
53          for (int i= 0; i< Appcontroller.FIRST_PART_SIZE; i++){
54              AppView.output( aMessage: this.list()[i]+ " ");
55          }
56          AppView.outputLine( aMessage: "");
57      }
58      private void validateSortsAndShowResult(){
59          this.validateSort(Appcontroller.INSERTION_SORT);
60          this.validateSort(Appcontroller.QUICK_SORT);
61          AppView.outputLine( aMessage: "");
62      }
63  @  private void validateSort(Sort<Integer> aSort){
64          Integer[] list= this.copyList(this.list());
65          aSort.sort(list, list.length);
66          this.showValidationMessage(aSort, list);
67      }
68  @  private Integer[] copyList(Integer[] aList){
69          Integer[] copiedList= new Integer[aList.length];
```



```
        for (int i= 0; i< aList.length; i++){
            copiedList[i]= aList[i];
        }
        return copiedList;
    }
    private boolean sortedListIsValid(Integer[] aList){
        for (int i= 0; i< (aList.length-1); i++){
            if (aList[i].compareTo(aList[i+ 1])> 0)  return false;
        }
        return true;
    }
    private void showValidationMessage(Sort<Integer> aSort, Integer[] aList){
        AppView.output(
                aMessage: "["+ this.listOrder().orderName()+ " 리스트]를 ["+
                        aSort.getClass().getSimpleName()+ "] 한 결과는 ");
        if (this.sortedListIsValid(aList)){
            AppView.outputLine( aMessage: "올바릅니다.");
        } else {
            AppView.outputLine( aMessage: "틀렸습니다.");
        }
    }

    //public methods
    public void run(){
        AppView.outputLine
                ( aMessage: "<<< 정렬 알고리즘의 정렬 결과를 검증하는 프로그램을 시작합니다 >>>");
        AppView.outputLine( aMessage: "");

        AppView.outputLine( aMessage: "> 정렬 결과의 검증:");
        AppView.outputLine( aMessage: "");

        this.validateWithAscendingOrderList();
        this.validateWithDescendingOrderList();
```



```
        this.validateWithRandomOrderList();

        AppView.outputLine
                ( aMessage: "<<< 정렬 알고리즘의 정렬 결과를 검증하는 프로그램을 종료합니다 >>>");
    }
}
```

DataGenerator

```java
package model;

import java.util.Random;

public final class DataGenerator {
    private DataGenerator(){}

    public static Integer[] ascendingList(int aSize){
        Integer[] list= null;
        if (aSize> 0 ){
            list= new Integer[aSize];
            for (int i= 0; i< aSize; i++){
                list[i]= i;
            }
        }
        return list;
    }
    public static Integer[] descendingList(int aSize){
        Integer[] list= null;
        if (aSize> 0 ){
            list= new Integer[aSize];
            int insert= aSize-1;
            for (int i= 0; i< aSize; i++){
                list[i]= insert;
                insert--;
            }
        }
        return list;
    }
    public static Integer[] randomList(int aSize){
        Integer[] list= null;
        if (aSize> 0){
            list= new Integer[aSize];
```

```
34              for (int i= 0; i< aSize; i++){
35                  list[i]= i;
36              }
37
38              Random random= new Random();
39              for (int i= 0; i< aSize; i++){
40                  int r= random.nextInt(aSize);
41                  Integer temp= list[i];
42                  list[i]= list[r];
43                  list[r]= temp;
44              }
45          }
46          return list;
47
48      }
49  }
50
```

ListOrder

```
1   package controller;
2
3   public enum ListOrder {
4       //데이터리스트를 구분하기 위해 사용. 오름차, 내림차, 무작위 순
5       Ascending, Descending, Random;
6
7       public static final String[] ORDER_NAMES= {"오름차순", "내림차순", "무작위"};
8
9   @   public String orderName() { return ListOrder.ORDER_NAMES[this.ordinal()]; }
12
13  }
14
```

Sort

```java
package model;

public abstract class Sort<E extends  Comparable<E>> {
    //protected method
    protected void swap(E[] aList, int i, int j){
        E tempElement= aList[i];
        aList[i]= aList[j];
        aList[j]= tempElement;
    }

    //Constructor
    protected Sort(){}

    //public Method
    public abstract boolean sort(E[] aList, int aSize);

}
```

QuickSort

```java
package model;

public class QuickSort<E extends  Comparable<E>> extends Sort<E> {
    //protected method
    protected void swap(E[] aList, int i, int j){
        E tempElement= aList[i];
        aList[i]= aList[j];
        aList[j]= tempElement;
    }
    //Constructor
    public QuickSort(){}
    //private Methods
    private int pivot(E[] aList, int left, int right){ return left;}
    private int partition(E[] aList, int left, int right){
        int pivot= this.pivot(aList, left, right);
        int toRight= left;
        int toLeft= right+1;
        do {
            do {toRight++;} while (aList[toRight].compareTo(aList[pivot])< 0);
            do {toLeft--;} while (aList[toLeft].compareTo(aList[pivot])> 0);
            if (toRight< toLeft){
                this.swap(aList, toRight, toLeft);
            }
        } while (toRight< toLeft);
        this.swap(aList, left, toLeft);
        return toLeft;
    }
    private void quickSortRecursively(E[] aList, int left, int right){
        if (left< right){
            int mid= this.partition(aList, left, right);
            this.quickSortRecursively(aList, left,  right: mid-1);
            this.quickSortRecursively(aList,  left: mid+ 1, right);
        }
```

```
    }

    //public Method
    @Override
    public boolean sort(E[] aList, int aSize){
        if ((aSize< 1) || (aSize> aList.length)) return false;
        int maxLoc= 0;
        for (int i= 1; i< aSize; i++){
            if (aList[i].compareTo(aList[maxLoc])> 0) maxLoc= i;
        }
        this.swap(aList,maxLoc,  j: aSize- 1);
        this.quickSortRecursively(aList,  left: 0,  right: aSize- 2);
        return true;
    }

}
```

Insertion

```java
package model;

public class InsertionSort<E extends  Comparable<E>> extends Sort<E> {
    //protected method
    protected void swap(E[] aList, int i, int j){
        E tempElement= aList[i];
        aList[i]= aList[j];
        aList[j]= tempElement;
    }

    //Constructor
    public InsertionSort(){}
    //public Method
    public boolean sort(E[] aList, int aSize){
        if ((aSize< 1) || (aSize> aList.length)) return false;
        int minLoc= 0;
        for (int i= 1; i< aSize; i++){
            if (aList[i].compareTo(aList[minLoc])< 0) minLoc= i;
        }
        this.swap(aList,  0, minLoc);
        for (int i= 2; i< aSize; i++){
            E insertedElement= aList[i];
            int insertionLoc= i- 1;
            while (aList[insertionLoc].compareTo(insertedElement)> 0){
                aList[insertionLoc+ 1]= aList[insertionLoc];
                insertionLoc--;
            }
            aList[insertionLoc+ 1]= insertedElement;
        }
        return true;
    }
}

```