자료구조 실습 보고서
[제 11주] 정렬 성능비교

제출일 : 2015 - 05 - 16

201402395 이승희

1. 프로그램 설명서
   1) 주요 알고리즘 / 자료구조 / 기타
      : insertion Sort, Quick Sort, Bubble Sort, Selection Sort에 대하여 성능을 측정한다. 각각의 Sorting 알고리즘에 대하여 Sequential, Reverse, Random Data의 경우를 측정한다. 각각의 측정에 대하여 Data의 size를 달리하여 측정한다.
   2) 함수 설명서
      (1) AppController
         - private void doTest(int dataSize) // 각 정렬을 테스트하고 Duration을 얻어낸다.
         - public void run() // 실질적인 main함수 역할.
      (2) AppView
         - public int inputInt() // 정수를 입력받는다.
         - public int inputMaxDataSize() // inputInt()를 이용해 MaxDataSize를 입력받는다.
         - public int inputDataTerm() // inputInt()를 이용해 inputDataTerm을 입력받는다.
         - public int inputSortType() // inputInt()를 이용해 inputSortType을 입력받는다.
         - public void outputResult(int dataSize, double insertionSortDuration, double quickSortDuration. Duble selectionSortDuration, double bubbleSortDuration) // 결과 출력 함수로, 각 정렬들의 Duration을 출력한다.
      (3) DataGenerator
         - public DataGenerator() // this._dataArray, this._dataSize를 초기화
         - public void generateSequentialData(int size) // 첫번째부터 size 까지 순차적으로 dataArray에 값을 넣는다.
         - public void generateReverseData(int size) // size부터 첫번째까지 역순으로 dataArray에 값을 넣는다.
         - public void generateRandomData(int size) // 난수 발생 함수를 이용하여 dataArray에 랜덤 값을 넣는다.
         - public int[] getData(int size) // 복사할 size만큼의 배열 copyArray를 생성하여 생성된 data배열을 size 만큼 복사한 후, 복사된 copyArray를 리턴한다.
      (4) PerformanceMeasurement
         - public PerformanceMeasurement() // insertionSort, quicksort, selectionSort, bubbleSort, beforTime, beforeDataSize를 초기화한다.
         - public double testInsertionSort(int[] data, int dataSize) // 삽입정렬의 성능을 테스트하여 insertTime을 리턴한다.
         - public double testQuickSort(int[] data, int dataSize) // 퀵정렬의 성능을 테스트하여 insertTime을 리턴한다.
         - public double testSelectionSort(int[] data, int dataSize) // 선택정렬의 성능을 테스트하여 insertTime을 리턴한다.
         - public double testBubbleSort(int[] data, int dataSize) // 버블정렬의 성능을 테스트하여 inserTime을 리턴한다.

(5) InsertionSort
- public InsertionSort() // insertionSort 생성자
- public void sort(int[] data, int size) // 배열에 데이터를 삽입하며 정렬

(6) SelectionSort
- public SelectionSort() // SelectionSort 생성자
- public void sort(int[] data, int size) // 작은 값을 앞으로 보낸다. 기준 위치는 맨 처음으로 설정하고 기준 위치에 맞는 값을 검색한 뒤 자리에 있는 값과 교환한다. 기준 위치를 그 다음으로 이동하고 다시 맞는 값을 검색하여 그 자리의 자료와 교환한다.

(7) QuickSort
- public QuickSort() // QuickSort생성자
- public void sort(int[] data, int size) // pivor과 divide를 통하여 정렬. 리스트 가운데 하나의 원소를 정해서 pivot으로 결정한 후 pivot을 기준으로 pivot앞에는 작은 모든 원소들이 오고 pivot의 뒤에는 큰 모든 원소들이 오도록 리스트를 둘로 나눈다. 분할된 작은 리스트에서 위의 과정을 반복.

(8) BubbleSort
- public BubbleSort() // BubbleSort 생성자
- public void sort(int[] data, int size) // 인접한 레코드의 키를 비교해서 그 결과 순서화되어 있지 않으면 교환하여 정렬. 주어진 자료들을 첫 번째 원소로부터 인접 항목끼리 비교 정렬, 큰 값을 계속해서 뒤로 보낸다.

3) 종합 설명서
: insertion, quick, selection, bubble Sort알고리즘을 이해하고 이를 구현한다. 구현한 Sorting 알고리즘의 성능을 측정하는 프로그램을 작성한다.

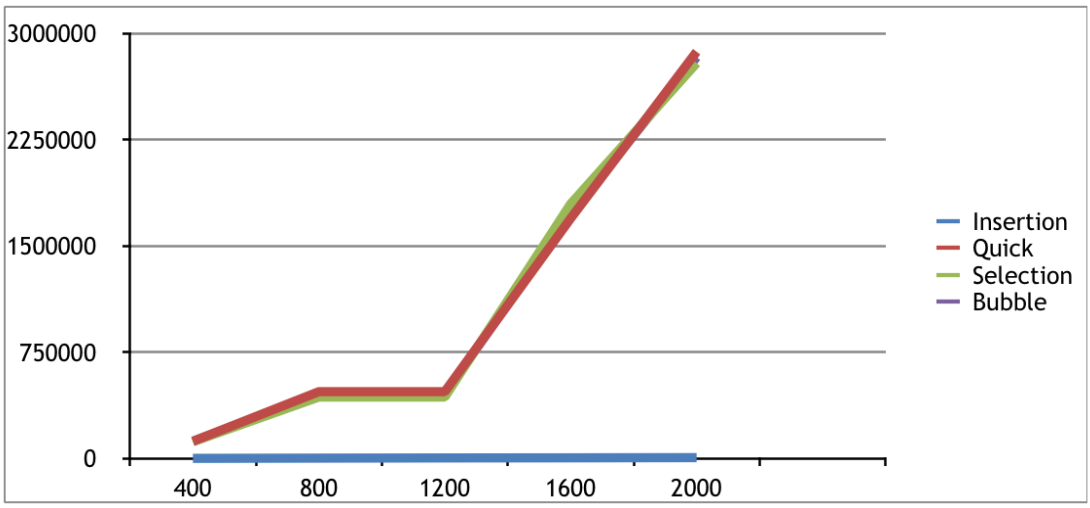
2. 프로그램 장단점 분석
: 4가지의 정렬의 성능을 비교하여 경우에 따라 어떤 정렬이 효율적인지 알 수 있다.

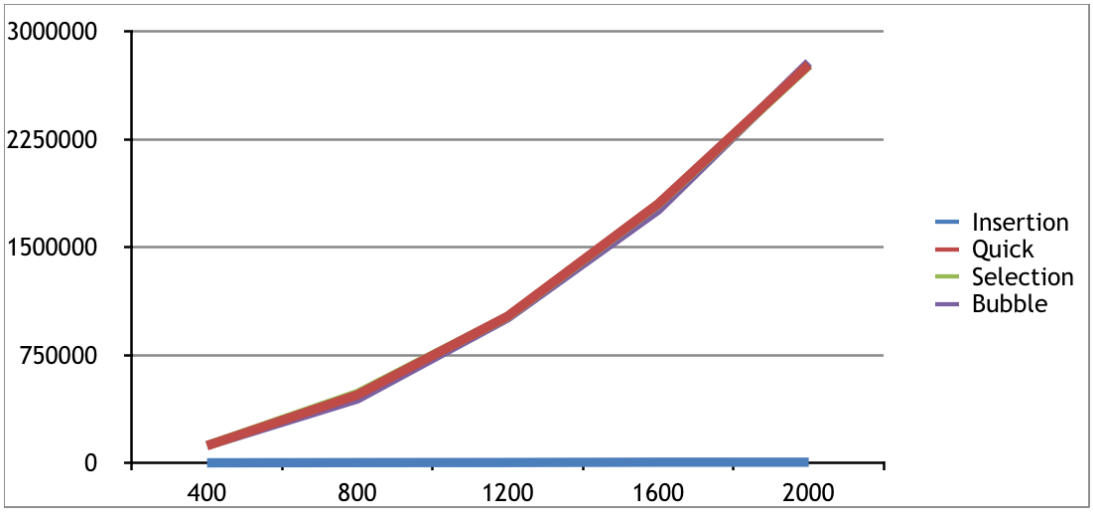
3. 실행 결과 분석
1) 입력과 출력

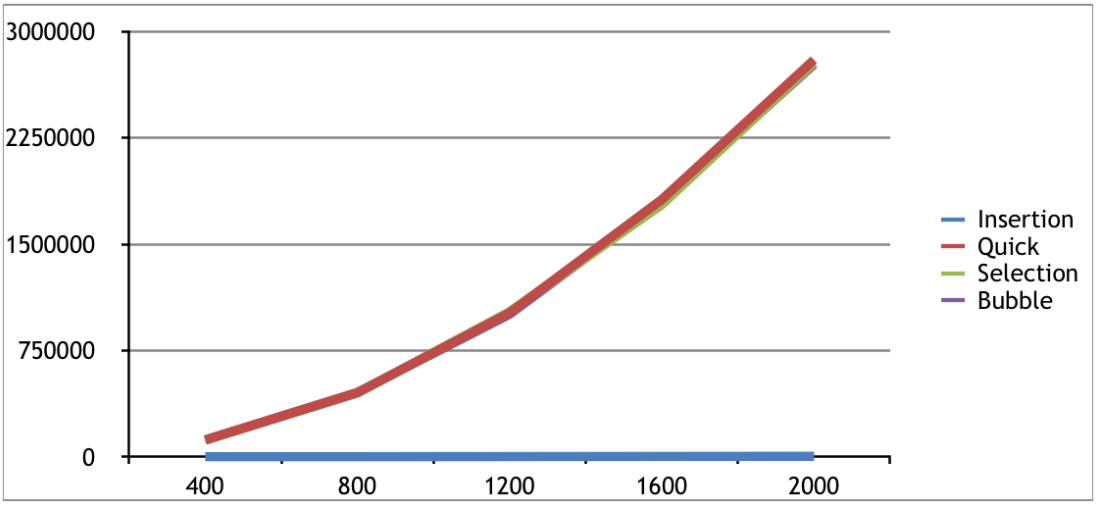
```
< 정렬에 따른 실행 성능 차이 알아보기 >
Insert Max Data Size >> 2000
Select a Sort >> 400
[1] sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >> 1
=== SEQUENTIAL DATA ===
DataSize        Insertion       Quick           Selection       Bubble
400             1203            121576          118562          121462
800             2788            472055          434094          455704
1200            4165            980370          973785          978687
1600            4911            1695724         1798531         1727563
2000            6414            2873004         2792535         2834604
[1] sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >> 2
=== REVERSE DATA ===
DataSize        Insertion       Quick           Selection       Bubble
400             960             119791          120707          121458
800             2436            473754          482989          445543
1200            3112            1021639         1018744         1009888
1600            5182            1802758         1799774         1760279
2000            5022            2769694         2764103         2785162
[1] sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >> 3
=== RANDOM DATA ===
DataSize        Insertion       Quick           Selection       Bubble
400             1852            119000          118324          121111
800             1903            452824          450760          455416
1200            2811            1015945         1024621         1005374
1600            3712            1814913         1777850         1798577
2000            4673            2799137         2777179         2765899

[1] sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >> 4
< 성능 측정을 종료합니다 >
```

2) 결과 분석

: Max Data Size는 2000, Data Term을 400으로 지정한 후 sequential Data, Reverse Data, Random Data로 정렬했을 때의 각각의 성능을 엑셀을 이용해 분석한 결과가 다음과 같다.







이론대로 나와야 할 데이터는 아니라 여러번 시도하고 코드도 바꾸어 보았지만 결과는 비슷했다. 무엇이 문제인지는 잘 모르겠다.

4. 소스코드

(1) AppController

```
public class AppController {
    private AppView _appView;
    private int[] _data;
    private DataGenerator _dataGenerator;
    private PerformanceMeasurement _pmMeasurement;
    private double _InsertionSortDuration;
    private double _QuickSortDuration;
    private double _bubbleSortDuration;
    private double _SelectionSortDuration;
    private int _sortType;

    private int _maxDataSize;
    private int _dataTerm;

    public AppController() {
        this._appView = new AppView();
        this._dataGenerator = new DataGenerator();
        this._pmMeasurement = new PerformanceMeasurement();
    }

    private void doTest(int dataSize) {
        this._InsertionSortDuration = 0;
        this._QuickSortDuration = 0;
        this._SelectionSortDuration = 0;
        this._bubbleSortDuration = 0;

        for (int i = 0; i < this._maxDataSize; i++) {
            this._data = this._dataGenerator.getData(dataSize);
            this._InsertionSortDuration += this._pmMeasurement
                    .testInsertionSort(this._data, dataSize);
        }
        this._InsertionSortDuration = this._InsertionSortDuration
                / this._maxDataSize;

        for (int i = 0; i < this._maxDataSize; i++) {
            this._data = this._dataGenerator.getData(dataSize);
            this._QuickSortDuration += this._pmMeasurement.testQuickSort(_data,
                    dataSize);


        }
        this._QuickSortDuration = this._QuickSortDuration / this._maxDataSize;

        for (int i = 0; i < this._maxDataSize; i++) {
            this._data = this._dataGenerator.getData(dataSize);
            this._bubbleSortDuration += this._pmMeasurement.testQuickSort(
                    _data, dataSize);

        }
        this._bubbleSortDuration = this._bubbleSortDuration / this._maxDataSize;

        for (int i = 0; i < this._maxDataSize; i++) {
            this._data = this._dataGenerator.getData(dataSize);
            this._SelectionSortDuration += this._pmMeasurement.testQuickSort(
                    _data, dataSize);

        }
        this._SelectionSortDuration = this._SelectionSortDuration
                / this._maxDataSize;
    }

    public void run() {
        this.showMessage(MessageiD.Notice_StartProgram);
        this._sortType = 0;

        this._maxDataSize = this._appView.inputMaxDataSize();
        this._dataTerm = this._appView.inputDataTerm();
```

```
        while (this._sortType != 4) {
            this.showMessage(MessageiD.Notice_Menu);
            this._sortType = this._appView.inputSortType();
            if (this._sortType == 1) {
                this._dataGenerator.generateSequentialData(this._maxDataSize);
                this.showMessage(MessageiD.Notice_SequentialData);
            } else if (this._sortType == 2) {
                this._dataGenerator.generateSequentialData(this._maxDataSize);
                this.showMessage(MessageiD.Notice_ReverseData);
            } else if (this._sortType == 3) {
                this._dataGenerator.generateSequentialData(this._maxDataSize);
                this.showMessage(MessageiD.Notice_RandomData);
            } else if (this._sortType == 4) {
                break;
            } else {
                this.showMessage(MessageiD.Error_WrongMenu);
                continue;
            }
            this.showMessage(MessageiD.Notice_ShowTitle);

            this.doTest(this._dataTerm);

            for (int dataSize = this._dataTerm; dataSize <= this._maxDataSize;
                    dataSize += this._dataTerm) {
                this.doTest(dataSize);
                this._appView.outputResult(dataSize, _InsertionSortDuration,
                        _QuickSortDuration, _SelectionSortDuration,
                        _bubbleSortDuration);
            }
        }
        this.showMessage(MessageiD.Notice_EndProgram);
    }



    private void showMessage(MessageiD aMessageID) {
        switch (aMessageID) {
        case Notice_StartProgram:
            System.out.println("< 정렬에 따른 실행 성능 차이 알아보기 >");
            break;
        case Notice_Menu:
            System.out.println("[1] sequential Data");
            System.out.println("[2] Reverse Data");
            System.out.println("[3] Random Data");
            System.out.println("[4] End");
            break;
        case Notice_SequentialData:
            System.out.println("=== SEQUENTIAL DATA ===");
            break;
        case Notice_ReverseData:
            System.out.println("=== REVERSE DATA ===");
            break;
        case Notice_RandomData:
            System.out.println("=== RANDOM DATA ===");
            break;
        case Notice_ShowTitle:
            System.out
                    .println("DataSize\tInsertion\tQuick\t\tSelection\t\tBubble");
            break;
        case Notice_EndProgram:
            System.out.println("< 성능 측정을 종료합니다 >");
            break;
        default:
            System.out.println("ERROR");

        }

    }

}
```

(2) AppView

```
import java.util.Scanner;

public class AppView {
    private Scanner _scanner;

    public AppView() {
        this._scanner = new Scanner(System.in);
    }

    public int inputInt() {
        return this._scanner.nextInt();
    }

    public int inputMaxDataSize() {
        System.out.print("Insert Max Data Size >> ");
        return this.inputInt();
    }

    public int inputDataTerm() {
        System.out.print("Insert Data Term >> ");
        return this.inputInt();
    }

    public int inputSortType() {
        System.out.print("Select a Sort >> ");
        return this.inputInt();
    }

    public void outputResult(int dataSize, double insertionSortDuration,
            double quickSortDuration, double selectionSortDuration,
            double bubbleSortDuration) {
        System.out
                .println(dataSize + "\t\t" + (int) insertionSortDuration + "\t\t"
                        + (int) quickSortDuration + "\t\t"
                        + (int) selectionSortDuration + "\t\t\t"
                        + (int) bubbleSortDuration);

    }
}
```

(3) BubbleSort

```
public class BubbleSort {
    public BubbleSort() {

    }

    public void sort(int[] data, int dataSize) {
        for (int i = 1; i < dataSize; i++)
            for (int j = 0; j < i; j++)
                if (data[j] > data[j + 1]) {
                    int temp = data[j];
                    data[j] = data[j + 1];
                    data[j + 1] = temp;
                }

    }

}
```

(4) DataGenerator

```
public class DataGenerator {
    private int[] _dataArray;
    private int _dataSize;

    public DataGenerator() {
        this._dataSize = 0;
        this._dataArray = new int[this._dataSize];
    }

    public void generateSequentialData(int size) {
        this._dataArray = new int[size];
        this._dataArray[0] = -1;
        this._dataSize = size;
        for (int i = 1; i < size; i++)
            this._dataArray[i] = i;
    }

    public void generateReverseData(int size) {
        this._dataArray = new int[size];
        this._dataArray[0] = -1;
        this._dataSize = size;
        for (int i = 1; i < size; i++)
            this._dataArray[i] = size - i;
    }

    public void generateRandomData(int size) {
        this._dataArray = new int[size];
        this._dataArray[0] = -1;
        this._dataSize = size;
        for (int i = 1; i < size; i++)
            this._dataArray[i] = (int) (Math.random() * this._dataSize);
    }


    public int[] getData(int size) {
        int[] copyArray = new int[size];
        for (int i = 0; i < size; i++)
            copyArray[i] = this._dataArray[i];

        return copyArray;
    }
}
```

(5) DS1_11_201402395_이승희

```
public class DS1_11_201402395_이승희 {

    public static void main(String[] args) {
        AppController appController = new AppController();
        appController.run();

    }

}
```

(6) InsertinSort

```
public class InsertionSort {

    public InsertionSort() {

    }
```

```
    public void sort(int[] data, int dataSize) {
        for (int i = 1; i < dataSize; i++) {
            int temp = data[i];
            int j = i - 1;
            while (j >= 0 && data[j] > temp) {
                data[j + 1] = data[j];
                j--;
            }
            data[j + 1] = temp;
        }

    }

}
```

(7) MessageID

```
public enum MessageID {
    Notice_StartProgram,
    Notice_EndProgram,
    Notice_Menu,
    Notice_SequentialData,
    Notice_ReverseData,
    Notice_RandomData,
    Notice_ShowTitle,
    // MessageIDs for Errors:
    Error_WrongMenu,
}
```

(8) PerformanceMeasurement

```
public class PerformanceMeasurement {
    private static final int DURATION_CHECKING = 5000;
    private InsertionSort _insertionSort;
    private QuickSort _quickSort;
    private SelectionSort _selectionSort;
    private BubbleSort _bubbleSort;
    private double _beforeTime;
    private int _beforeDataSize;

    public PerformanceMeasurement() {
        this._insertionSort = new InsertionSort();
        this._quickSort = new QuickSort();
        this._selectionSort = new SelectionSort();
        this._bubbleSort = new BubbleSort();

        this._beforeTime = 0;
        this._beforeDataSize = 0;
    }

    public double testInsertionSort(int[] data, int dataSize) {
        double insertTime = 0;
        long start, end;

        start = System.nanoTime();
        this._insertionSort.sort(data, dataSize);
        end = System.nanoTime();
        insertTime = (double) (end - start);

        return insertTime;
    }
```

```
    public double testQuickSort(int[] data, int dataSize) {
        double insertTime = 0;
        long start, end;

        start = System.nanoTime();
        this._quickSort.sort(data, dataSize);
        end = System.nanoTime();
        insertTime = (double) (end - start);

        return insertTime;
    }

    public double testSelectionSort(int[] data, int dataSize) {
        double insertTime = 0;
        long start, end;

        start = System.nanoTime();
        this._selectionSort.sort(data, dataSize);
        end = System.nanoTime();
        insertTime = (double) (end - start);

        return insertTime;
    }

    public double testBubbleSort(int[] data, int dataSize) {
        double insertTime = 0;
        long start, end;

        start = System.nanoTime();
        this._bubbleSort.sort(data, dataSize);
        end = System.nanoTime();
        insertTime = (double) (end - start);

        return insertTime;
    }

}
```

(9) QuickSort

```
public class QuickSort {
    public QuickSort() {

    }

    public void sort(int[] data, int dataSize) {
        if (dataSize > 1) {
            int maxLoc = 0;
            for (int i = 1; i < dataSize; i++) {
                if (data[i] > data[maxLoc])
                    maxLoc = i;
            }
            swap(data, maxLoc, dataSize - 1);
            quickSortRecursively(data, 0, dataSize - 2);
        }

    }

    private void quickSortRecursively(int[] data, int left, int right) {
        int pivot, up, down;
        if (left < right) {
            pivot = left;
            up = left;
            down = right + 1;
            do {
                do {
                    up++;
                } while (data[pivot] > data[up]);
                do {
                    down--;
                } while (data[pivot] < data[down]);
                if (up < down) {
                    swap(data, up, down);
                }
            } while (up < down);
            swap(data, pivot, down);
            int mid = down;
            quickSortRecursively(data, left, mid - 1);
            quickSortRecursively(data, mid + 1, right);

        }
    }
```

```
    private void swap(int[] data, int a, int b) {
        int temp = data[a];
        data[a] = data[b];
        data[b] = temp;

    }

}
```

(10) SelectionSort

```
public class SelectionSort {
    public SelectionSort() {

    }

    public void sort(int[] data, int size) {
        int i = 0;
        while (i != size) {
            int min = data[i];
            for (int j = size - 1; j < size; j++)
                if (min > data[j])
                    min = data[j];
            sort(data, i);
            i++;
        }
    }

}
```