자료구조

실습 보고서

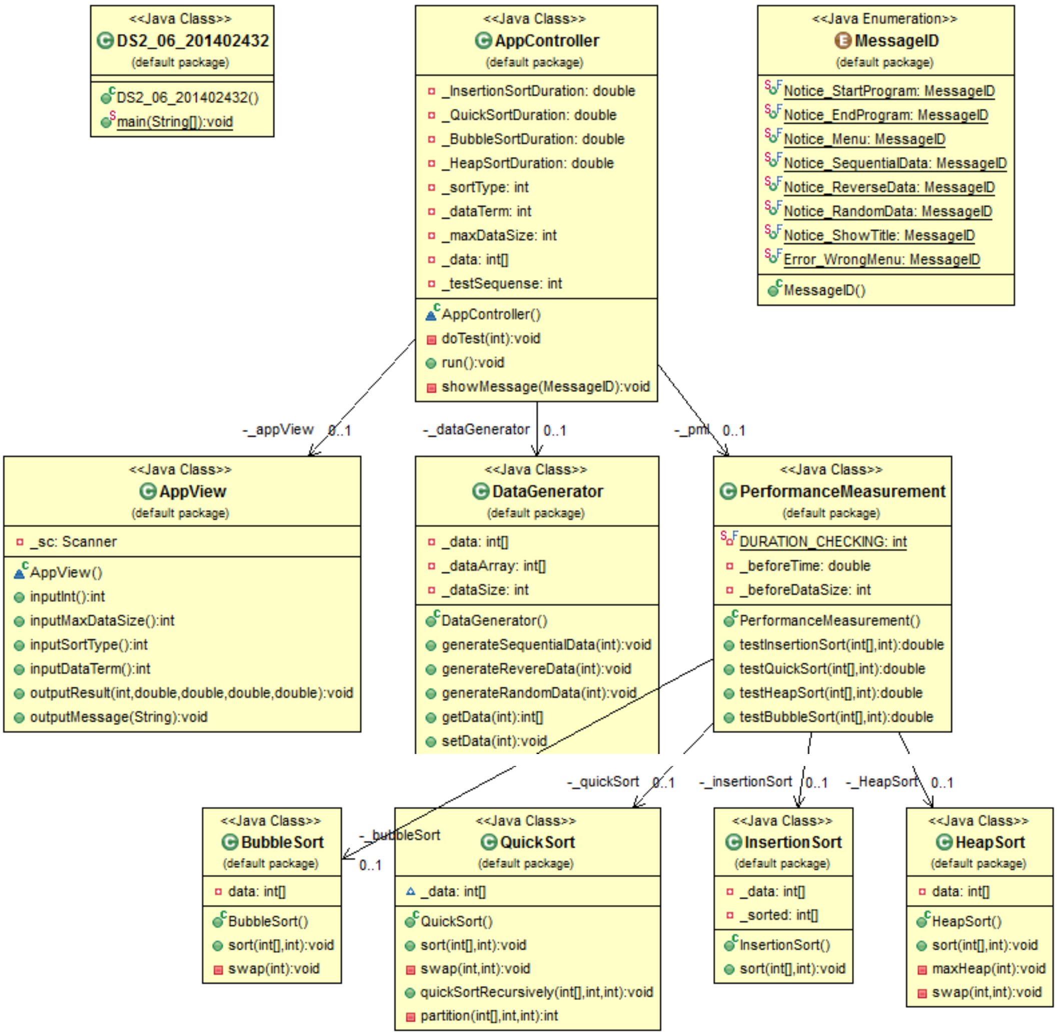
[제 06주] 정렬 – 성능비교

제출일 : 2015.11.03

201402432 / 조디모데

1.프로그램설명서

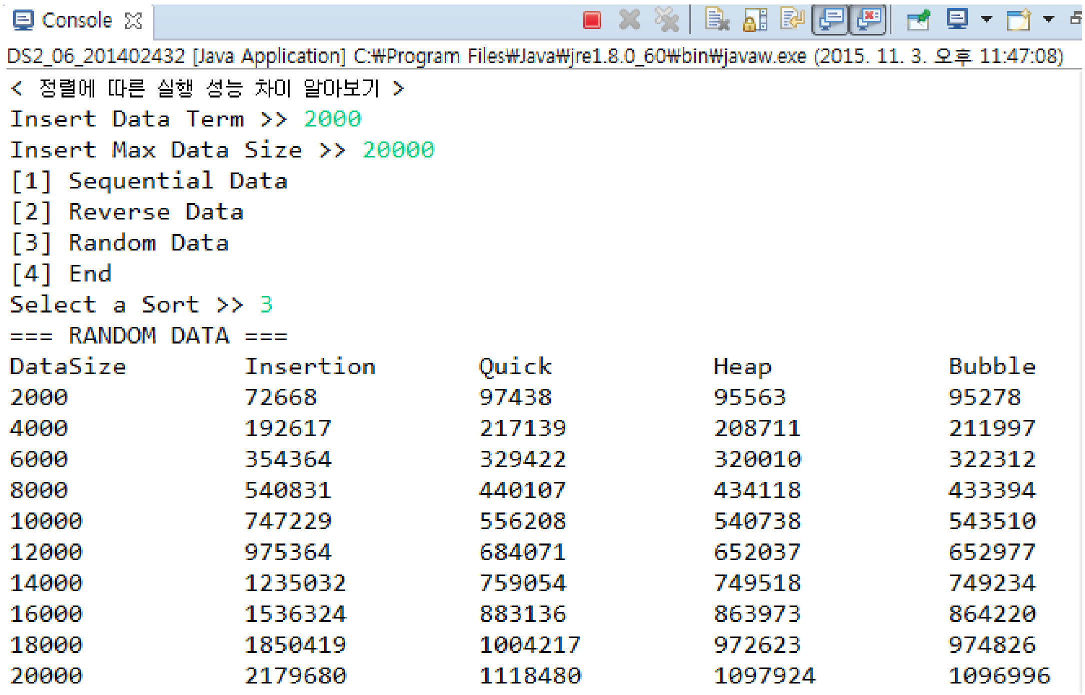
자료 구조 : Insertion, Heap, Bubble, Quick



2.실행 결과 분석

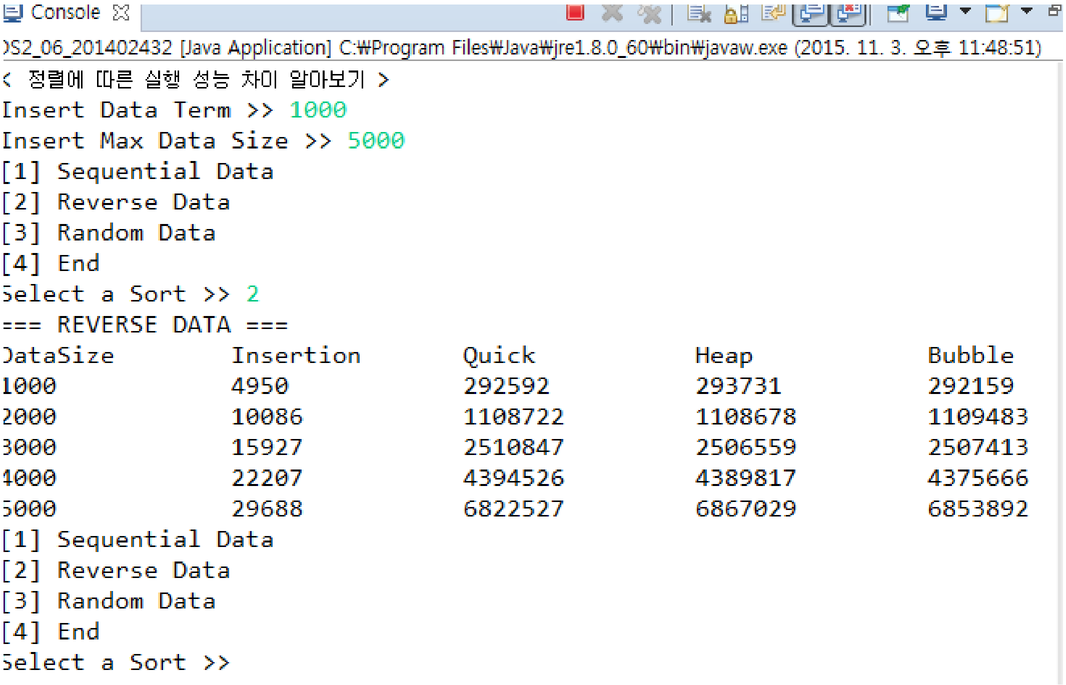
1.입력과출력

Random



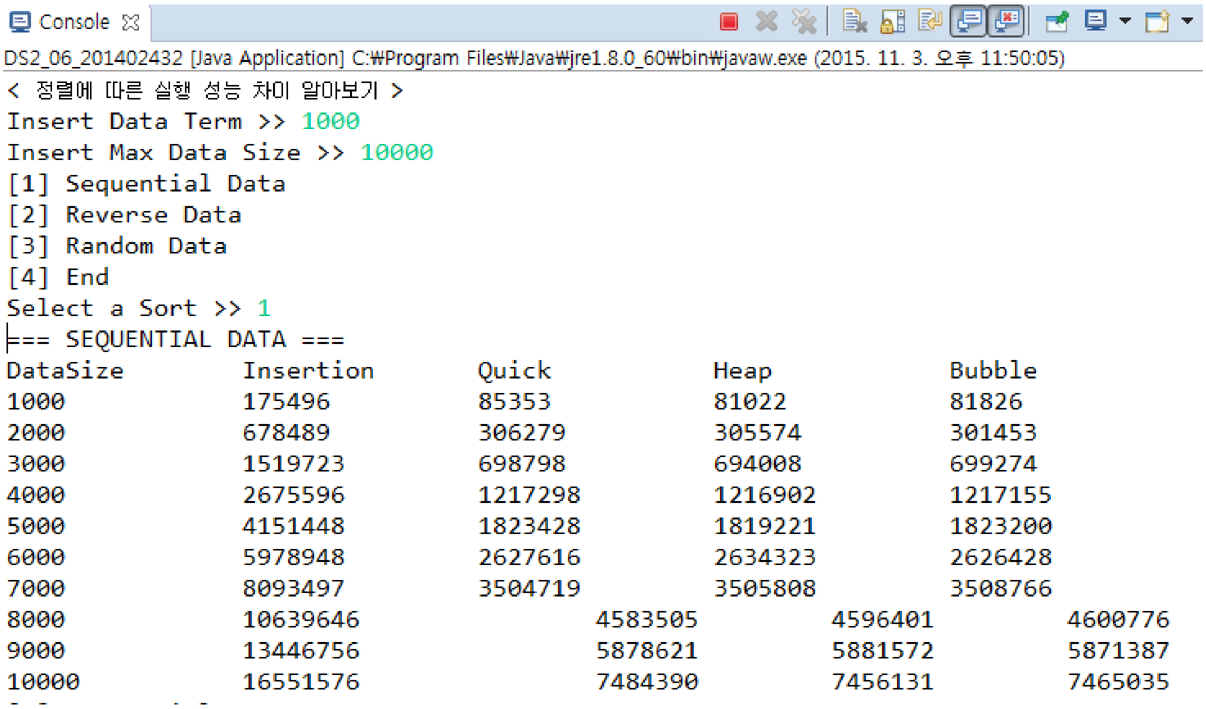
```
DS2_06_201402432 [Java Application] C:₩Program Files₩Java₩jre1.8.0_60₩bin₩javaw.exe (2015. 11. 3. 오후 11:47:08)
< 정렬에 따른 실행 성능 차이 알아보기 >
Insert Data Term >> 2000
Insert Max Data Size >> 20000
[1] Sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >> 3
=== RANDOM DATA ===
DataSize        Insertion       Quick           Heap            Bubble
2000            72668           97438           95563           95278
4000            192617          217139          208711          211997
6000            354364          329422          320010          322312
8000            540831          440107          434118          433394
10000           747229          556208          540738          543510
12000           975364          684071          652037          652977
14000           1235032         759054          749518          749234
16000           1536324         883136          863973          864220
18000           1850419         1004217         972623          974826
20000           2179680         1118480         1097924         1096996
```

Reverse



```
DS2_06_201402432 [Java Application] C:₩Program Files₩Java₩jre1.8.0_60₩bin₩javaw.exe (2015. 11. 3. 오후 11:48:51)
< 정렬에 따른 실행 성능 차이 알아보기 >
Insert Data Term >> 1000
Insert Max Data Size >> 5000
[1] Sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >> 2
=== REVERSE DATA ===
DataSize        Insertion       Quick           Heap            Bubble
1000            4950            292592          293731          292159
2000            10086           1108722         1108678         1109483
3000            15927           2510847         2506559         2507413
4000            22207           4394526         4389817         4375666
5000            29688           6822527         6867029         6853892
[1] Sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >>
```

Sequence



DS2_06_201402432 [Java Application] C:₩Program Files₩Java₩jre1.8.0_60₩bin₩javaw.exe (2015. 11. 3. 오후 11:50:05)

```
< 정렬에 따른 실행 성능 차이 알아보기 >
Insert Data Term >> 1000
Insert Max Data Size >> 10000
[1] Sequential Data
[2] Reverse Data
[3] Random Data
[4] End
Select a Sort >> 1
=== SEQUENTIAL DATA ===
DataSize        Insertion       Quick           Heap            Bubble
1000            175496          85353           81022           81826
2000            678489          306279          305574          301453
3000            1519723         698798          694008          699274
4000            2675596         1217298         1216902         1217155
5000            4151448         1823428         1819221         1823200
6000            5978948         2627616         2634323         2626428
7000            8093497         3504719         3505808         3508766
8000            10639646                4583505         4596401         4600776
9000            13446756                5878621         5881572         5871387
10000           16551576                7484390         7456131         7465035
```

3.소스 코드

<main>

```
public class DS2_06_201402432 {

      public static void main(String[] args) {

            AppController appController = new AppController() ;
            appController.run() ;


      }

}
```

<performanceMeasurement>

```
import java.util.* ;

public class PerformanceMeasurement {

      private static final int DURATION_CHECKING = 5000;
      private InsertionSort _insertionSort;
      private QuickSort _quickSort;
      private HeapSort _HeapSort;
      private BubbleSort _bubbleSort;
      private double _beforeTime;
      private int _beforeDataSize;

      public PerformanceMeasurement(){

            this._insertionSort = new InsertionSort() ;
            this._quickSort = new QuickSort() ;
            this._HeapSort = new HeapSort() ;
            this._bubbleSort = new BubbleSort() ;
            this._beforeTime = -1 ;
            this._beforeDataSize = -1 ;

      }

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

        start = System.nanoTime() ;
        this._HeapSort.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 ;

    }
```

```
}
```

<Application>

```
public class AppController {

    private AppView _appView ;
    private DataGenerator _dataGenerator ;
    private PerformanceMeasurement _pml ;
    private   double _InsertionSortDuration ;
    private   double _QuickSortDuration ;
    private   double _BubbleSortDuration ;
    private   double _HeapSortDuration ;

    private   int _sortType ;
    private   int _dataTerm ;
    private   int _maxDataSize ;
    private   int[] _data ;
    private int _testSequense ;

    AppController(){
```

```
                this._appView = new AppView() ;
                this._dataGenerator = new DataGenerator() ;
               this._pml = new PerformanceMeasurement() ;
                this._testSequense = 50 ;

        }

        private void doTest(int dataSize){

                this._InsertionSortDuration = 0 ;
                this._QuickSortDuration = 0 ;
                this._BubbleSortDuration = 0 ;
                this._HeapSortDuration = 0 ;

                this._data =
this._dataGenerator.getData(dataSize) ;

                for(int index = 0 ; index <
this._testSequense ; index++){
                        this._InsertionSortDuration +=
this._pml.testInsertionSort(this._data, dataSize);
```

```
            this._QuickSortDuration +=
this._pml.testQuickSort(this._data, dataSize);
            this._BubbleSortDuration +=
this._pml.testQuickSort(this._data, dataSize);
            this._HeapSortDuration +=
this._pml.testQuickSort(this._data, dataSize);
        }

        this._InsertionSortDuration =
this._InsertionSortDuration / this._testSequense ;
        this._QuickSortDuration =
this._QuickSortDuration / this._testSequense ;
        this._BubbleSortDuration =
this._BubbleSortDuration / this._testSequense ;
        this._HeapSortDuration =
this._HeapSortDuration / this._testSequense ;

    }

    public void run(){
```

```
    this.showMessage(MessageID.Notice_StartProgra
m);
        this._sortType = 0 ;

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


    this._dataGenerator.setData(this._maxDataSize) ;

        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_SequentialD
ata);
                        }
                        else if (this._sortType == 2){

        this._dataGenerator.generateRevereData(this._max
DataSize) ;

        this.showMessage(MessageID.Notice_ReverseData
);
                        }
                        else if (this._sortType == 3){

        this._dataGenerator.generateRandomData(this._ma
xDataSize) ;

        this.showMessage(MessageID.Notice_RandomDat
a);
```

```
            }
            else if (this._sortType == 4){
                break ;
            }
            else {

    this.showMessage(MessageID.Error_WrongMenu);
                continue ;
            }



    this.showMessage(MessageID.Notice_ShowTitle) ;

            // 메모리 생성 및 테스트의 안정성을
위하여 가장 첫 성능 측정을 미리 한번 진행한다.
            this.doTest(this._dataTerm);

            // 실제 테스트 진행
            for(int dataSize = this._dataTerm ;
dataSize<=this._maxDataSize ; dataSize +=
this._dataTerm){
```

```
                              this.doTest(dataSize);

        this._appView.outputResult(dataSize,
this._InsertionSortDuration,this._QuickSortDuration,

        this._HeapSortDuration,this._BubbleSortDuration);

                              System.out.println();
                         }
                    }

        this.showMessage(MessageID.Notice_EndProgram
) ;

        }

       private void showMessage(MessageID MessageID)
{
               switch(MessageID) {
               case Notice_StartProgram:
                         this._appView.outputMessage("< 정렬에
따른 실행 성능 차이 알아보기 >₩n");
            break;
        case Notice_EndProgram:
            this._appView.outputMessage("< 성능
측정을 종료합니다 >₩n");
            break;
        case Notice_Menu :
            this._appView.outputMessage(   "[1]
Sequential Data₩n"
                                              + "[2]
Reverse Data₩n"
                                              + "[3]
Random Data₩n"
                                              + "[4]
End₩n") ;
            break;
        case Notice_SequentialData :
            this._appView.outputMessage("===
SEQUENTIAL DATA ===₩n");
            break;
        case Notice_ReverseData :
```

```
                              this._appView.outputMessage("===
REVERSE DATA ===₩n");
                              break;
                    case Notice_RandomData :
                              this._appView.outputMessage("===
RANDOM DATA ===₩n");
                              break;
                    case Notice_ShowTitle :

          this._appView.outputMessage("DataSize₩tInsertio
n₩tQuick₩t₩tHeap₩t₩tBubble₩n");
                              break;

                    case Error_WrongMenu:

          this._appView.outputMessage("<<ERROR:
잘못된메뉴입니다.>>₩n");
                              break;


                    }
          }
```

}<AppView>

```
import java.util.Scanner;

public class AppView {

      private Scanner _sc ;

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

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

      public int inputMaxDataSize(){
            this.outputMessage("Insert Max Data Size >> ");
            return this._sc.nextInt() ;
      }

      public int inputSortType() {
            this.outputMessage("Select a Sort >> ");
            return this._sc.nextInt() ;
      }

      public int inputDataTerm() {
            this.outputMessage("Insert Data Term >> ");
            return this._sc.nextInt() ;
      }

      public void outputResult(int dataSize, double
InsertionSortDuration,
                  double QuickSortDuration, double HeapSortDuration,
                  double BubbleSortDuration) {

            String str =
dataSize+"\t\t"+(int)InsertionSortDuration+"\t\t"+(int)QuickSortDura
tion

+"\t\t"+(int)HeapSortDuration+"\t\t"+(int)BubbleSortDuration ;

            this.outputMessage(str) ;
```

```
    }

    public void outputMessage (String aMessageString) {
        System.out.print(aMessageString);
    }




}
```

<performance>

```
import java.util.* ;

public class PerformanceMeasurement {

	private static final int DURATION_CHECKING = 5000;
	private InsertionSort _insertionSort;
	private QuickSort _quickSort;
	private HeapSort _HeapSort;
	private BubbleSort _bubbleSort;
	private double _beforeTime;
	private int _beforeDataSize;

	public PerformanceMeasurement(){

		this._insertionSort = new InsertionSort() ;
		this._quickSort = new QuickSort() ;
		this._HeapSort = new HeapSort() ;
		this._bubbleSort = new BubbleSort() ;
		this._beforeTime = -1 ;
		this._beforeDataSize = -1 ;

	}

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

        start = System.nanoTime() ;
        this._HeapSort.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 ;

    }


}
```

<quick Sort>

```
public class QuickSort{
    int[] _data ;

    public void sort(int[] data, int dataSize){

        this._data = data.clone() ;

        int minLoc = 0 ;

        // 최소값을 원소 구간의 맨 끝으로 옮긴다.
        swap(minLoc, dataSize-1) ;
        // 정렬을 시작한다.
        quickSortRecursively(this._data, 0,dataSize-2) ;

    }
    private void swap(int positionA, int positionB){
        int temp = this._data[positionA] ;
        this._data[positionA] = this._data[positionB] ;
        this._data[positionB] = temp ;
    }

    public void quickSortRecursively(int[] data, int left, int
right){
        if (left < right) /* 구간의크기가2 이상이면*/{
            int mid = partition(this._data, left, right) ; //
DIVIDE
            quickSortRecursively(data, left, mid-1) ;//
CONQUER
            quickSortRecursively(data, mid+1, right) ; //
CONQUER
        }
    }

    private int partition(int data[], int left, int right){
        int pivot = left ;
        int pivotScore = data[pivot] ;
        right++ ;
        do{
            do{
                left++ ;
            }while(data[left] > pivotScore);

            do{
                right--;
            }while(data[right] < pivotScore);

            if(left<right)
                this.swap(left, right);

        }while(left<right);
        this.swap(pivot, right);
        return right ;
    }

}
}
```

<InsertionSort>

```
import java.util.Arrays;

public class InsertionSort {
    private int[] _data ;
    private int[] _sorted ;

    public InsertionSort(){

    }

    public void sort(int[] data, int dataSize) {
        int tmp ;

        this._data = Arrays.copyOf(data, dataSize) ;
        this._data[0] = -1 ;

        for(int i=1 ; i<dataSize ; i++){

            for(int j=1 ; j<i ; j++)
                if(this._data[i]<this._data[j]){
                    tmp = this._data[j] ;
                    this._data[j] = this._data[i] ;
                    this._data[i] = tmp ;
                    break ;
                }

        }

    }


} // class
```

<Heapsort>

```
public class HeapSort {

    private int[] data ;

    public HeapSort(){


    }

    public void sort(int[] data, int dataSize) {

        this.data = data.clone() ;
        for(int i=0 ; i<this.data.length ; i++)
            this.maxHeap(i) ;
```

```java
    }

    private void maxHeap(int i) {
        int l = i*2+1 ;
        int r = i*2+2 ;
        int largest ;

        if( (l <= this.data.length-1) &&
(this.data[l]>this.data[i]))
            largest = l ;
        else
            largest = i ;

        if(r <= this.data.length-1 && this.data[l] >
this.data[i])
            largest = r ;
        if(largest != i){
            swap(i, largest) ;
            maxHeap(largest) ;
        }
    }

    private void swap(int i, int largest) {
        int tmp = this.data[i] ;
        this.data[i] = this.data[largest] ;
        this.data[largest] = tmp ;
    }

}
```

<DataGenerator>

```java
import java.util.* ;
import java.util.Arrays;
public class DataGenerator {


    private int[] _data ;
    private int[] _dataArray;
    private int _dataSize ;

    public DataGenerator(){
        this._dataSize = -1 ;

    }

    public void generateSequentialData(int size) {
        this._dataArray = Arrays.copyOf(this._data,
size) ;
        Arrays.sort(this._dataArray) ;
    }

    public void generateRevereData(int size) {
        this._dataArray = Arrays.copyOf(this._data,
size) ;
        Arrays.sort(this._dataArray) ;
        int tmp[] = this._dataArray.clone() ;

        for(int i=0 ; i<size ; i++)
            this._dataArray[i] = tmp[size-i-1] ;
```

```
                this._dataArray[0] = -1 ;
        }

        public void generateRandomData(int size) {
                // 이미 랜덤으로 저장된 Data입니다.
                this._dataArray =
Arrays.copyOf(this._data,size) ;
        }

        public int[] getData(int size) {
                int[] copyArray =
Arrays.copyOf(this._dataArray, size) ;

                return copyArray ;
        }

        public void setData(int size) {
                Random r = new Random() ;
                this._dataSize = size ;
                this._data = new int[size] ;
        this._data[0] = -1 ;
        for(int i=1 ; i<size ; i++)
            this._data[i] = r.nextInt(1000)+1 ;


    }


}
```

<BubbleSort>

```
import java.util.Arrays;

public class BubbleSort {

    private int[] data ;

    public BubbleSort(){

    }

    public void sort(int[] data, int dataSize) {

        this.data = Arrays.copyOf(data, dataSize) ;

        for(int i = 0 ; i<dataSize ; i++)
            for(int j = 0 ; j<dataSize-i-1 ; j++)
                if(this.data[j]>this.data[j+1])
                    swap(j) ;
    }
    private void swap(int n){
        int tmp ;
        tmp = this.data[n] ;
        this.data[n] = this.data[n+1] ;
        this.data[n+1] = tmp ;
    }
}
```