Paged Sorting

The ability to page through data is such a ubiquitous feature in modern applications than any information retrieval-based application will likely provide some type of access to paged data. In general, the first page contains the first *n* data items, the second page contains the next *n* data items and so on. In this lab you will work with a Java interface `Pageable` in order to provide paged access to a set of objects. The pageable interface defines the following methods:

```
public interface Pageable<T> {

  public int size();           // number of elements in the collection
  public int pageSize();       // max number of elements per page
  public int pages();          // number of pages in the collection
  public T page(int i);        // returns the elements in the ith page

}
```

Part I - Native Sorting

For this part of the assignment you will write a class `JavaSortedPager` that uses `java.util.Arrays.sort()` to order the elements and make them accessible as a paged set of data items via the `Pageable` interface. Your pager class should list the elements in *increasing sort order*.

```
public class JavaSortedPager<T extends Comparable<T>> implements Pageable<T>
```

```
JavaSortedPager(T [] objects, int pageSize)    // constructor
T get(int i)                                    // get item at position i
T max()                                         // return largest item
T min()                                         // return smallest item
```

Edge Cases

The constructor should throw `java.lang.IllegalArgumentException` when `pageSize` is less than 1 or if `objects` is `null` or of length zero. `get()` should throw `java.lang.IndexOutOfBoundsException` if the value of `i` falls is outside the size of the data.

Analysis

Once your implementation is complete. Test and evaluate the run time for creating a pager object and returning the 1st page of results for increasingly large values of `N` and `pageSize`. (Use the helper function `Lab2.randomDoubles(int N)` to generate arrays with random sort keys). Provide the results of your testing and evaluation in the Run-Time Analysis section of the Analysis document. In the Run-Time Performance for Key Methods, provide the average, best and worst case performance of each method. (If there is no distinction, you can just say something like "Constant time operation" to describe the performance)

Part II - Ranked Results

For this part of the assignment consider an input array of `ScoredDocument` objects with that will ultimately be paged in descending sort order. Assume the scores are uniformly distributed.

Predictive Analysis

Before writing code, analyze the input conditions and design an algorithm for paging this data via the `Pageable` interface. Predict how your solution should perform. Note your analysis in the Predictive Analysis section on the Analysis document. Once your predictive analysis is complete, implement your algorithm as `ScoredResultsPager`. (Note: Your design should do more than simply implement one of the sorts in the book. It is ok to use them as a starting point, but a correct solution make an attempt at optimizing the sort mechanism based on the nature of the input).

```
public class ScoredResultsPager<T extends Comparable<T>> implements Pageable<T>
```

```
ScoredResultsPager(T [] objects, int pageSize)    // constructor
T get(int i)                                      // get item at position i
T max()                                           // return largest item
T min()                                           // return smallest item
```

Edge Cases

The constructor should throw `java.lang.IllegalArgumentException` when `pageSize` is less than 1 or if `objects` is `null` or of length zero. `get()` should throw `java.lang.IndexOutOfBoundsException` if the value of `i` falls is outside the size of the data.

Analysis

Once your implementation is complete. Test and evaluate the run time for creating a pager object and returning the 1st page of results for increasingly large values of `N` and `pageSize`. (Use the helper function `Lab2.generateScoredDocuments(int N)` to generate an array of `ScoredDocuments`). Provide the results of your testing and evaluation in the Run-Time Analysis section of the Analysis document. Did your implementation perform as predicted? In the Run-Time Performance for Key Methods, provide the average, best and worst case performance of each method. (If there is no distinction, you can just say something like "Constant time operation" to describe the performance).

What To Submit

The Analysis Google Doc, `ScoredResultsPager.java`, `JavaSortedPager.java` and any other supporing source files.

You may not use any additional libraries/imports beyond Arrays, Iterator, Comparable, StdRandom and the exceptions noted above. If you have any doubts, just ask me.