Assignment 6

Radityo Eko Prasojo, Werner Nutt,
Rafael Penaloza, Guohui Xiao

Sorting and Heaps

1. Person Sorter

In this assignment we are applying sorting algorithms to objects instead of numbers. In the `DSA_A6` zip file, you find templates for the three classess related to this assignment, that is, the classes

1. `Person`,
2. `PersonSorter`, and
3. `PersonSorterTest`, which contains the JUnit tests.

Your task is to complete the class `PersonSorter`. The `PersonSorter` class provides methods to ascendingly sort an array of instances of `Person` by a field contained in the class `Person`. We provide you with two constants for two sorting modes: `BY_LAST_NAME` and `BY_DATE_OF_BIRTH`, each of which corresponds to a field of the class `Person`. The method `sortBy` then takes an array of `Person` and a specified sorting mode as inputs and returns the sorted array.
For the class `PersonSorter`, your task is as follows.

1. Implement the two sorting methods mentioned above. Use a different sorting algorithm for each method.
2. Implement as well a method `sortByLastNameAndDateOfBirth`. This method is to sort an array of `Person` object by the last name and the date of birth in lexicographic order, which means the following: Consider p_1 and p_2 to be two elements of the input array. If p_1 and p_2 have different last names, then arrange them by their last name. Otherwise, if they have the same last name, then arrange them by their date of birth. Use also a different sorting algorithm for this method.

Finally, test your sorting algorithm using the JUnit tests provided in the class `PersonSorterTest`.

(10 Points)

2. Sorting algorithm comparison

In this exercise you are asked to:

- Implement in Java the *Quicksort* algorithm for arrays of integers.
- Implement in Java the *Hybrid Quicksort* algorithm for arrays of integers, a variant of Quicksort that depends on an additional parameter k. The algorithm works as follows:
  - if the size of the portion of the array to be sorted is greater than k, it recursively calls the Hybrid Quicksort method with the same k;
  - if the size of the portion of the array to be sorted is less than or equals to k, it sorts that portion by using the Insertion Sort algorithm.
- Empirically compare these two sorting algorithms for different values of k ($k = \{10, 20, 50, 100, \ldots\}$) and different array sizes n ($n = \{20, 100, 1000, 10000\}$). Find out for which values of k and for which array sizes Hybrid Quicksort outperforms the standard Quicksort.

Provide:

1. the code that you developed for your experiments;
2. an analysis for which value of k and for which array size n the standard Quicksort algorithm starts to outperform the hybrid one;
3. a discussion and possible explanation for the obtained running times.

Background: Many implementations of recursive sorting algorithms, like Quicksort or Mergesort, actually implement such a hybrid version.

Hint: Write Java code to make the tries and take the measurements, rather than doing it manually with pencil and paper.

(20 Points)

3. Priority Queues

In this coursework we realize so-called priority queues using heaps. Priority queues are a an abstract data-type that allows one to insert new elements into a set and to extract the maximum of a set. Priority queues are a crucial building block of many complex algorithms. They form an *abstract* data type because we do not specify how to implement them.
You have learnt about heaps as a data structure that allows one to retrieve the maximum in a set of numbers in constant time. Moreover, a heap can be maintained in logarithmic time if we insert a new element at the root. Your task will be to realize priority queues using heaps in arrays.
Implement priority queues as instances of the class

```
public class PriorityQueue{
    int maxSize;
    int currentSize;
    int[] queue;

public PriorityQueue(int maxSize){
    this.maxSize = maxSize;
    queue = new int[maxSize];
    currentSize = 0;
  }
}
```

An instance of this class can hold a heap of up to `maxSize` integers. Initially, the heap has size 0, which means that none of the elements of the array `queue` is considered to be a heap element.
You are now asked to implement the following methods for this class:

1. `boolean empty()`, which returns `true` iff the heap is empty;
2. `boolean full()`, which returns `true` iff the number of integers in the heap is equal to `maxSize`;
3. `int extractMax()`, which returns the maximum element of the heap, deletes it from the heap, and reorganizes the array so that it is a heap of the remaining elements; after this operation the `currentSize` of the priority queue is one less than it was before; clearly, the operation can only be applied if the priority queue is not empty, otherwise, an exception has to be raised;
4. `void insert(int n)`, which inserts a number n and reorganizes the array so that it is a heap of the new, larger set of elements; after this operation the `currentSize` of the priority queue is one greater than it was

before; clearly, the operation can only be applied if the priority queue is not full, otherwise, an exception has to be raised.

Your task is to develop and implement efficient algorithms for these methods and to test them. Explain all your solutions and answers to questions. You find a template for the class `PriorityQueue` in the zip file `DSA_A6.zip`.

1. Develop JUnit tests for the methods above, covering both special and general cases. You may want to write tests that combine several calls to the methods you want to write. For instance, you may want to insert several numbers and then execute `extractMax`.

2. Write pseudocode for the methods `extractMax` and `insert` and for auxiliary methods that you may need.

3. What is the asymptotic worst-case complexity of your algorithms?

4. Implement the four methods listed above.

(10 Points)

Deliverables.

1. Your implementation of the class `PersonSorter` in Exercise 1 and the class `PriorityQueue` in Exercise 3, as well as the code that you wrote for your experiments in Exercise 2.

2. Write one report for all tasks.

Combine all deliverables into one zip file, which you submit via the OLE website of the course. Please, follow the "Instructions for Submitting Course Work" on the Web page with the assignments, when preparing your coursework.
Submission: Until Mon, 2 May 2016, 23:55 hrs, to the OLE submission page of

Lab A / Lab B / Lab C