Assignment 1

Julien Corman, Tiziano Dalmonte
Werner Nutt

Methods for Integer Arrays

Instructions: Your assignment should represent your own effort. However, you are not expected to work alone. It is fine to discuss the exercises and try to find solutions together, but each student shall write down and submit his/her solutions separately. It is good academic standard to acknowledge collaborators, so if you worked together with other students, please list their names.

Constraints: Your solutions should only use data structures that have been studied so far in this course. For this assignment, the only type of data structure that you may use is arrays.

1. Implementation of Array Algorithms

Implement in Java the class `IntegerArrays`, which offers operations over one-dimensional integer arrays. *All* methods *must* be implemented as class methods (i.e., `static` methods). The signature of the methods in the `IntegerArrays` class are the following:

1. `public static void swap(int[] A, int i, int j):`
   swaps (i.e., exchanges) the elements in position `i` and `j` in array `A`.

2. `public static boolean sorted(int[] A):`
   checks whether `A` is sorted, that is, whether $\texttt{A}[i] \leq \texttt{A}[j]$ for all indexes $i < j$ of `A`.

3. `public static int maxPos(int[] A, int l, int r):`
   returns a position of the maximal value v in array `A` between position `l` (included) and `r` (included). If v is repeated, the method returns the position of any of its occurrences.

4. `public static int[] reversedCopy(int[] A):`
   returns a reversed copy of `A`, that is, a *new* array where the first element is the last of `A`, the second the last but one of `A`, etc.

5. `public static void reverse(int[] A):`
   reverses the order of the elements of `A` so that the first elements becomes the last, the second the last but one, etc.

6. `public static int localMax(int[] A):`
   returns the position of a local maximum of `A` provided all elements of `A` are distinct. The array `A` has a local maximum at position j if $\texttt{A}[j] > \texttt{A}[j-1]$ and $\texttt{A}[j] > \texttt{A}[j+1]$,

provided $j-1$ and $j+1$ are indices of `A`. Note that an array of length 1 has a local maximum at position $j = 0$. If the array is longer, then the first element of `A` is a local maximum if its neighbour to the right is less than the first element and, similarly, the last element is a local maximum if its neighbour to the left is less than the last element.

Write code so that the running time of the method is logarithmic in the length of the array.

7. `public static void selectionSortMax(int[] A):`

   sorts array `A` according to the Selection Sort principle. That is, (1) finds the position of the maximum of `A` and swaps the maximum with the last array element, (2) finds the position of the maximum among the remaining elements (from the first to the last but one position) and swaps that maximum with the element in the last but one position. Then the method moves ahead analogously, until the array is sorted.

8. `public static int mostFrequentElement(int[] A):`

   returns the most frequent element of `A` or one element with maximal frequency if there are several such elements. For example, if `A` $= [5, 2, 1, 2, 5, 7, 5]$, then `mostFrequentElement(A)` returns 5, while for `B` $= [5, 2, 1, 2, 5, 7, 5, 2]$ it may return 2 or 5.

(Weight: 60% of this CW)

2. Implementation of Algorithms for Sorted Arrays

Implement in Java the class `SortedArrays`, which offers operations over sorted one-dimensional integer arrays. *All* methods *must* be implemented as class methods (i.e., `static` methods). The signatures of the methods in the `SortedArrays` class are the following:

1. `public static int firstPosXSorted(int[] A, int x):`

   is applied to a sorted array and returns the first position where `x` occurs. The array may have duplicate elements. If x does not occur in `A`, the method returns -1. For example, if `A` $= [1, 2, 2, 5, 5, 5, 7]$, then `firstPosXSorted(A,` $5)$ returns 3, while `firstPosXSorted(A,` $6)$ returns -1.

   Implement a variant of binary search, as discussed in the lecture, so that the running time of the method is logarithmic in the length of the array.

2. `public static int lastPosXSorted(int[] A, int x):`

   is applied to a sorted array and returns the last position where `x` occurs. The array may have duplicate elements. If `x` does not occur in `A`, the method returns -1. For example, if `A` $= [1, 2, 2, 5, 5, 5, 7]$, then `lastPosXSorted(A,` $5)$ returns 5, while `lastPosXSorted(A,` $6)$ returns -1.

   Again, implement an algorithm whose running time is logarithmic in the length of the array.

   Hint: Use a variant of the approach for the preceding exercise.

3. `public static int mostFrequentElementSorted(int[] A)`:
   if applied to a sorted array, the method returns the most frequent element of A or one element with maximal frequency if there are several such elements. For example, if $\texttt{A} = [1, 2, 2, 5, 5, 5, 7]$, then `mostFrequentElementSorted(A)` returns 5, while for $\texttt{B} = [1, 2, 2, 2, 5, 5, 5, 7]$ it may return 2 or 5.

(Weight: 40% of this CW)

Deliverables. Submit two copies of your code:

- one via Codeboard (instructions are available here),
- one via the OLE submission page of your lab.

Combine all files into one zip file, which you submit via the OLE submission page of your lab. Please include name, student ID and email address in your submission.

Submission until Wednesday, 19 October 2022, 23:55, to Codeboard and the OLE submission slot for this assignment:

Submission Assignment 1