Assignment 4

David Blumenthal, Florian Hofer, Werner Nutt, Daniele Porello

Loop Invariants and Performance of Sorting Algorithms

1. Loop Invariants

In this exercise we want to review loop invariants and how they can be used to understand algorithms.
Below is pseudocode for an algorithm that is supposed to check whether an array is sorted.

Input: Nonempty array $A[1..n]$ of integers
Output: TRUE if the array is sorted, FALSE otherwise

```
BOOLEAN CHECKSORTEDNESS(INT[] A)
    n:=A.length
    i:=1
    while i<n and A[i]<=A[i+1] do
        i++
    if i=n
        then return TRUE
        else return FALSE
```

Our goal in this exercise is to show that CHECKSORTEDNESS does in fact check whether an array is sorted.

1. Write down a formal definition of the statement, "Array A is sorted."
2. State a loop invariant for the while loop of CHECKSORTEDNESS by which you can show that the algorithm in fact is checking sortedness.
3. Give arguments that your loop invariant holds when the algorithm reaches the while loop for the first time (initialization).
4. Give arguments that your loop invariant is maintained by each execution of the loop (maintenance).

5. Give arguments that the loop terminates (termination).
6. Give arguments that the answer `TRUE` is returned only if the array was sorted, and `FALSE` only if it was not sorted.

(15 Points)

2. Comparison of Sorting Algorithms

In this exercise you are asked to empirically compare two sorting algorithms, one with a worst-case running time of $O(n^2)$ and another one with a worst-case running time of $O(n \log n)$. In particular, we would like to know for which length of input arrays the second algorithm is faster than the first.

1. Write a Java program implementing the Insertion Sort algorithm.
2. Write a Java program implementing the Merge Sort algorithm.
3. Compare the performance of the two algorithms:
   (a) Write code that generates a random array A, then runs each algorithm on A, and records the time.
   (b) Repeat this for several arrays of the same size, still recording the running times.
   (c) Gradually increase the size of the arrays, until you see that one algorithm is consistently faster than the other.

   Is the theoretical analysis confirmed by your experiments? For which array size is Merge Sort faster than Insertion Sort?

(15 Points)

Deliverables.

1. For question 3, hand in the Java file that you wrote.
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 Tue, 4 April 2017, 23:55 hrs, to the OLE submission page of

Lab A / Lab B / Lab C