Assignment 3

Mouna Kacimi, Werner Nutt,
Camilo Thorne

Loop Invariants and Performance of Sorting Algorithms

Instructions: Each student shall write down and submit his/her solutions separately. If you worked together with other students, please list their names.
You can write up your answers by hand (provided your handwriting is legible) or use a word processing system like Latex or Word.
For a programming task, your solution must contain (i) an explanation of your solution to the problem, (ii) the Java code, in a form that we can run it, (iii) instructions how to run it. Also put the source code into your solution document. For all programming tasks, it is not allowed to use any external libraries ("import") if not stated otherwise.
Please, include name, matriculation number and email address in your submission.

1. Loop Invariants

Below is pseudocode for an algorithm that performs linear search.

Input: Array $A[1..n]$ of integers and an integer k.
Output: TRUE if k is found in A, FALSE otherwise.

LINEARSEARCH(A, k):

```
i := 1
found := FALSE
 while  i<=n  and  found=false  do
    if  A[i]=k  then
        found := TRUE
    i := i+1
return  found
```

State a loop invariant for the while loop of the LinearSearch algorithm, and explain why it is correct.

Hint: Try a loop invariant of the form:

> If found is FALSE, then ..., and if found is TRUE then ...

(10 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 $O(n \log n)$.
The question we are interested in is whether the size of the data that are sorted has an influence on the outcome of the comparison. In particular, we would like to know for which length of input arrays the second algorithm is faster than the first, depending on the size of the data in the arrays.

1. Write a Java program implementing the insertion sort algorithm.
2. Write a Java program implementing the merge sort algorithm.
3. Create two versions of each algorithm:
   (a) one for arrays of standard int values of maximum 32-bits;
   (b) another one for arrays of BigInteger values.

   Then compare the performance of the two algorithms
   (a) for arrays with random standard ints and
   (b) for arrays with random BigIntegers in the range from 1024 to 2048 bits, as they may occurs as RSA keys.

   Gradually increase the size of the input array until the difference between the two algorithms is noticeable.

   Hint: Consider also to increase the memory size if needed, by calling your program, say `MySort`, as

   ```
   java -XX:+AggressiveHeap MySort
   ```

4. Does the size of the integers affect the performance of the algorithms? Which array size is the crossover point, from where on the asymptotically better algorithm has the better performance? Provide explanations for your findings.

(20 Points)

Submission: Until Mon, 31 March 2014, 8:30 am, to

`dsa-submissions AT inf DOT unibz DOT it.`