Assignment 4

Alessandro Artale, Werner Nutt,
Ognjen Savković, Francesco Sportelli

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

CHECKSORTEDNESS(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 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.

(12 Points)

2. Merging Two Sorted Array Segments

We have seen that the Merge Sort algorithm can sort an array of length n in time $O(n \log n)$. It relies on a subroutine MERGE with the following specification:

MERGE(A, l, m, r)
Input: Array $A[1..n]$ of integers;
positive integers l, m, and r, with $1 \leq l \leq m < r \leq n$
Precondition: the array segments $A[l..m]$ and $A[m+1..r]$ are sorted
Postcondition: the entire segment $A[l..r]$ is sorted.

Your task is to design an algorithm for this merge operation.

1. Describe your overall idea for the algorithm. Explain which control structure the algorithm will have and how you choose the boundaries for loops.

2. Write up pseudocode for MERGE that follows this idea.

Hint: You need extra space for the merge operation. Either you copy the two segments into new arrays and then merge the new arrays into the target segment, or you merge the source segments into a new array and copy that array into the target segment.

(8 Points)

3. 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, 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. 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?

(10 Points)

Please, follow the "Instructions for Submitting Course Work" on the Web page with the assignments, when preparing your coursework.

Also, include name, student ID, code of your lab group (A, B, or C), and email address in your submission.

Submission: Until Mon, 13 April 2015, 11:55 pm, to

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