Technische Universität München
Institut für Informatik
Scientific Computing in Computer Science

WS 2013/2014
Final Exam
23.12.2013

Fundamental Algorithms

December 23, 2013

General Instructions

Material: You may only use one hand-written sheet of paper (size A4, on both pages). Any other material including electronic devices of any kind is forbidden.

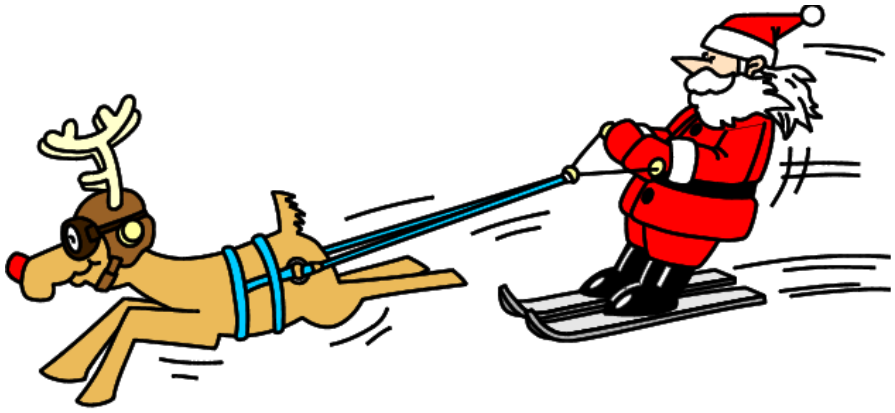
Use only the exam paper that was handed out to solve the exercises – for notes and sketches, you can obtain additional exam sheets.

Do not use pencil, or red or green ink.

General hint: Often, exercises b), c), etc. can be solved without the results from the previous exercise a): if you are stuck with exercise a), then don't immediately skip exercises b), c), etc.

Working time: 40 minutes + 5 minutes reading time.

Please switch off your cell phones!



Merry Christmas and all the best for the new year!

1 Correctness of InsertionSort

Consider the sorting algorithm InsertionSort, as introduced in the lecture:

```
InsertionSort(A: Array[1..n]) {

    for j from 2 to n {
    // insert A[j] into sequence A[1..j-1]

        key := A[j];

        i := j-1;  //initialize i for while loop
        while i>=1 and A[i]>key {
            A[i+1] := A[i];
            i := i-1;
        }
        A[i+1] := key;
    }
}
```

To prove correctness of the algorithm, someone provides you with the following loop invariant for the while-loop:

> At the beginning of the while loop body:
>
> – A[1..i+1] contains the original elements of A[1..i+1] in sorted order
>
> – A[i+2..j] contains the original elements of A[i+1..j-1] in sorted order

a) Prove (or disprove) initialisation and maintenance for this invariant.

b) What does the loop invariant state at termination? Does it help to prove the correctness of the entire InsertionSort? (If not, how do we need to extend the loop invariant?)

Note: the suggested invariant actually contains an error, which makes the exercise more difficult than originally planned – try to work out a correct solution, as well (you will probably need a bit of extra time to figure it out ...

2 Minimum Search on the PRAM

The following parallel algorithm is suggested for searching the minimum in an array:

```
MinPRAM( A: Array [1..n]) : Integer {
    // n assumed to be 2^k

    s := n;
    for i from 1 to k do  {
        s := s/2;
        for j from 1 to s do in parallel
            if A[j] > A[j+s]  then A[j] := A[j+s];
    }
    return A[1];
}
```

Explain whether read and write accesses in this PRAM algorithm are exclusive or concurrent. Draw a diagram for the case $n = 8$ to illustrate the access pattern.

3 QuickSort with Expensive Pivoting

Consider the following variant of the QuickSort algorithm: to determine a good pivot element, we recursively sort the first third of the input array. After sorting, we select the centre element of this third as pivot element. See the following implementation of this algorithm (which – up to the pivoting strategy – is analogous to RandQuickSort discussed in the lectures):

```
TSQuickSort(A:Array[p..r]) {
    n := r−p+1;
    if n<3 then SimpleSort(A)
    else {
        // sort the first third of the array A:
        TSQuickSort(A[p .. p+n/3−1]);
        // make element A[p+n/6] the (new) Pivot element:
        exchange A[p+n/6] and A[p];
        q := Partition(A);
        TSQuickSort(A[p..q]);
        TSQuickSort(A[q+1 .. r]);
    };
}
```

a) In this algorithm, Partition is identical to the respective algorithm defined in the lectures. Describe shortly and in plain words, what the Partition does on the array A, and what the main idea is for its implementation. How many comparisons are required on an input array of size n?

b) Concerning the quality of the selected pivot element: how many elements are guaranteed to be larger (and smaller) than this pivot element. What are the worst-case sizes of the generated partitions?

c) Set up a recurrence equation for the number $T(n)$ of comparisons executed in the worst case by the algorithm TSQuickSort on an input array of n elements (assuming a suitable implementation of SimpleSort).

d) Apply the substitution method on the recurrence equation derived in Exercise 3c) to explain that $T(n) \notin O(n \log n)$.