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.

Maximum score: The total number of points for all exercises is 47, however grades will be computed relative to a maximum score of 42 points.

Working time: 90 minutes + 5 minutes reading time.

Please switch off your cell phones!

Points and Grades:

<table>
<thead>
<tr>
<th>0.0–8.5</th>
<th>9.0–12.0</th>
<th>12.5–15.5</th>
<th>16.0–17.5</th>
<th>18.0–19.5</th>
<th>20.0–21.5</th>
<th>22.0–23.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.0</td>
<td>4.7</td>
<td>4.3</td>
<td>4.0</td>
<td>3.7</td>
<td>3.3</td>
<td>3.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>24.0–26.0</th>
<th>26.5–28.5</th>
<th>29.0–31.0</th>
<th>31.5–33.5</th>
<th>34.0–36.0</th>
<th>36.5–</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.7</td>
<td>2.3</td>
<td>2.0</td>
<td>1.7</td>
<td>1.3</td>
<td>1.0</td>
</tr>
</tbody>
</table>
1 Finding the Smallest Difference (4+5+4 = 13 points)

For a given array A of integers (in arbitrary order), we want to find the two array elements A[i] and A[j] that have the smallest difference \( |A[i] - A[j]| \), where \( | \cdot | \) computes the absolute value. The following divide-and-conquer algorithm is our first attempt to solve this problem:

```
QuickDiff(A: Array[p..r]) : (Integer, Integer) {
    // for small arrays, check all combinations:
    if (p-r < 3) then return SimpleDiff(A[p..r]);

    // for large arrays: partition into two halves:
    q := Partition(A);
    // compute the element pairs with smallest difference
    // in both of the partitions
    (i1, j1) := QuickDiff(A[p..q]);
    (i2, j2) := QuickDiff(A[q+1..r]);

    // return the indices of the pair with the smaller difference:
        then return (i1, j1)
    else return (i2, j2);
}
```

SimpleDiff is a simple algorithm to solve our problem for subarrays with at most 3 elements. Partition is the algorithm you know from Quicksort (using the first array element as “pivot”).

a) Describe shortly and in plain words, what the algorithm Partition (as defined in the lectures) 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) \( T_{QD}(n) \) shall be defined as the number of all accesses to array elements of A performed by a call to QuickDiff for an array with \( n \) elements. Give a recurrence equation to compute \( T_{QD}(n) \) and give asymptotic upper bounds for \( T_{QD}(n) \) for the best, worst and average case (no computation required; refer to similar recurrences solved in the lectures).

For clarification: comparing two elements (as required in Partition) or subtracting two elements requires two accesses to array elements. You can assume that SimpleDiff only requires a constant number of element accesses.

c) The algorithm QuickDiff, as given here, is actually not correct. State what the algorithm does not consider, and describe how the algorithm has to be extended – make sure that the algorithm does not increase the asymptotical complexity!
Consider the following AVL tree:

\[
\begin{array}{c}
\text{n} \\
\text{l} \\
\text{A} \\
\text{D} \\
\text{r} \\
\text{B} \\
\text{C} \\
\end{array}
\]

\[
\begin{align*}
h(A) &= m - 1 \\
h(B) &= m - 2 \\
h(C) &= m - 2 \\
h(D) &= m - 1 \\
\end{align*}
\]

a) Compute the heights \( h \) and height balances \( b \) required to show that the tree satisfies the AVL property (assuming that the subtrees \( A, B, C \) and \( D \) all satisfy the AVL property).

b) We now insert a single additional element into the subtree. Assume that the AVL property is now violated only(!) in the top node \( n \). Describe all possibilities where the element could have been added to cause this violation.

c) For all situations, name the respective rotation (if any) that is required to restore the AVL property. Perform this rotation for one situation and show that the height of the tree is the same as before the insertion.
3 Correctness of a Parallel Prefix Algorithm ($\approx 3+8 = 11$ pts)

Given is the following parallel algorithm (EREW variant, as discussed in the lectures) for the prefix problem:

```plaintext
PrefixPRAM( A: Array [1..n]) {
    // n assumed to be 2^k
    // Model: EREW PRAM (n−1 processors)
```

for L from 0 to k-1 do
    for j from \(2^L+1\) to n do in parallel {
        tmp[j] := A[j-2^L];
    }
}

a) State a loop invariant for the L-loop (i.e., sequential outer loop) that helps to prove its correctness.

b) Prove the correctness of PrefixPRAM using this loop invariant. You may assume that the j-loop (i.e., parallel inner loop) is correct. However, draw a diagram to explain precisely what the j-loop computes to make the entire algorithm correct.
4 Parallel Matrix-Vector Multiplication ($\approx 3+5 = 8$ pts)

The following algorithm is suggested to compute the Matrix-Vector product $y = Ax$ for a band matrix $A$ (i.e., elements $A_{ij} = 0$, if $|i - j| > k$). The author of the algorithm claims that the algorithm runs correctly on a CRCW-PRAM with $n(2k + 1)$ processors.

```
BandMV_CRCW(A: Array[1..n,1..n], x: Array[1..n], y: Array[1..n]) {
    for i from 1 to n do in parallel {
        y[i] := 0;
    }
    for i from 1 to n do in parallel {
        for j from i-k to i+k do in parallel {
            if j>=1 and j<=n then y[i] := y[i] + A[i,j]*x[j];
        }
    }
}
```

a) Describe where concurrent read or write accesses occur during parallel execution of this algorithm (on $3n$ processors). What assumptions have to be made on the concurrent accesses such that the algorithm works correctly?

b) Describe how to turn BandMV_CRCW into an EREW PRAM algorithm. State the number of processors that your EREW algorithm uses. Describe in detail why your algorithm fits into the EREW classification. You may use algorithms known from the lecture to make your algorithm EREW. In that case, describe why this algorithm is EREW.
5 Connected Graphs ($\approx 4 + 2 = 6$ points)

The following (incomplete) algorithm IsConnected is an attempt to construct an algorithm to test whether a (non-directed) graph is connected, i.e., whether there exists a path between any two nodes of the graph. The graph is represented by its adjacency matrix.

IsConnected (A: Array [1..n, 1..n]) {
  ! Input: adjacency matrix A
  ! A[i,j] = 1, if i connected to j, otherwise 0

  // part 1 of the algorithm:
  for k from 1 to n do
    for i from 1 to n do
      for j from 1 to n do
        if (A[i,k]=1 and A[k,j]=1) then A[i,j] = 1
      end do
    end do

  // part 2 of the algorithm:
  /* ... -> is missing ... */
}

a) Part 1 of the algorithm adapts the idea of an algorithm that was discussed in the lecture. State the name of this algorithm, explain its main idea and describe how it is exploited here to decide whether the graph is connected.

b) Provide an implementation for the missing “part 2” of the algorithm. It should return true, if the graph is connected, and false otherwise.

...