Obligatory Exercises

This document contains the list of obligatory exercises of the course Efficient Algorithms and Problem Complexity. Please prepare your solutions as individual pdf documents, one per exercise, and submit them [here]. If corrections are required, the deadline for the second submission is one week after the ordinary deadline (unless something else has explicitly been announced).

Exercise 1 (due 2014-09-11)

This exercise is derived from Exercises 6.2.17-23 in the textbook by Johnsonbaugh and Schaefer.

First of all, recall the quicksort algorithm. Given an array \( a \) and two indices \( i, j \) with \( i \leq j \) it sorts the array elements \( a[i, \ldots, j] \). (Use the textbook or some other source if you do not remember how it works.) Now, let \( m3quicksort(a, i, j) \) be the version that uses the median of the three array cells \( a[i] \), \( a[(i + j)/2] \), and \( a[j] \) in order to partition \( a[i, \ldots, j] \) (instead of always using \( a[i] \) for that purpose).

1. Write down suitable pseudocode for \( m3quicksort \). (If \( m3quicksort \) uses the procedure \( partition \) from the textbook without changes, you do not need to include that one. This is recommended but not required.)

2. Determine the recurrence relation that describes the running time of the algorithm \( m3quicksort \), if it is applied to a sorted array of \( n \) pairwise distinct elements. Prove by induction that this recurrence relation indeed describes the running time of the algorithm in this special case. In your inductive argument, do not forget to point out why the induction hypothesis applies!

3. Solve the recurrence relation by the Main Recurrence Theorem (or Master Theorem) to determine the running time of the algorithm on this type of input.

Exercise 2 (due 2014-09-15)

Let \( k \) be a constant, and consider the problem of sorting an array of \( n \) integers in the range \( 0, \ldots, n^k - 1 \). Determine the running time of radix sort when using, instead of a binary representation of numbers,

1. a base-10 representation
2. a base-\( n \) representation.

Exercise 3 (due 2014-09-22)

Suppose you want to travel from city \( s \) to city \( t \) via a road network which is represented as a weighted graph \( G \) with nonnegative edge weights representing the distance between two cities. On your way to \( t \) you would like to stop by the city \( u \) if it is not too inconvenient, too inconvenient meaning that it would increase the length of your travel by more than 10%.
(1) Describe an efficient algorithm that determines an optimal path from $s$ to $t$ given your preference for stopping at $u$ along the way if not too inconvenient. Thus, the algorithm should either return the shortest path from $s$ to $t$ or the shortest path from $s$ to $t$ containing $u$, depending on whether or not the latter is convenient. (Hint: Rather than designing your own algorithm from scratch, use one of the know algorithms as a subroutine. In this way, it becomes easy.)

(2) What is the running time of your proposed algorithm, and why?

(3) Argue that your algorithm is correct, i.e., it indeed gives you a shortest convenient path.

Exercise 4 (due 2014-09-29)

Professor Carmel observes that his files consist of only letters A, B, C, D, E, F with the following frequencies:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>5%</td>
<td>5%</td>
<td>15%</td>
<td>20%</td>
<td>25%</td>
<td>30%</td>
</tr>
</tbody>
</table>

Being a slow adopter of technological innovations, the professor still saves his files on floppy disks. Consequently, he wants to save some disk space and decides to use a variable length code to encode the above letters. After some thinking he comes up with the following code:

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>000</td>
<td>101</td>
<td>010</td>
<td>011</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

However, Professor Carmel is unsure whether his variable length code is correct.

(1) Provide your opinion regarding the correctness of the code (including a motivation).

(2) Construct two Huffman codes for the characters in the table, one which is optimal and one which is not.

Exercise 5 (due 2014-10-06)

A cyclic rotation of a string is obtained by chopping off a prefix and attaching it at the end of the string. More formally, $v$ is a cyclic rotation of $u$ if $u$ can be decomposed into $u = u_1 u_2$ such that $v = u_2 u_1$. Clearly, in this case $u$ is also a cyclic rotation of $v$. For example, ALGORITHM and RITHMALGO are cyclic rotations of each other. You are required to design a simple algorithm called Cyclic_Rotation that uses string pattern matching in order to determine in linear time whether a string is a cyclic rotation of another string.

(1) Given two strings $u, v$, construct strings $u', v'$ such that $v$ is a cyclic rotation of $u$ if and only if $u'$ matches $v'$. Explain why your method is correct and illustrate this with the strings ALGORITHM and RITHMALGO.

(2) As a consequence, how efficiently can Cyclic_Rotation be implemented?

(3) Assume finally that you, instead of a pure yes/no answer, want to compute a decomposition of $u$ into $u_1$ and $u_2$ such that $v = u_2 u_1$ in those cases where $v$ actually is a cyclic rotation of $u$. How can you do this?

Exercise 6 (due 2014-10-13)

By definition, a decision problem is NP-complete if it is in NP and all other
problems in NP can be reduced to it by a reduction that runs in polynomial time. Explain why it is important to require that the reduction runs in polynomial time, as opposed to accepting arbitrary computable reductions. (Hint: Argue that the notion of NP-completeness that is obtained by dropping the polynomial-time requirement would be useless for learning anything about the difference between P and NP.)

**Exercise 7 (due 2014-10-16)**

Call a graph $G$ *almost Hamiltonian* if it has a cycle that contains every node of $G$ at least once and at most twice.

Show that the problem whether $G$ is almost Hamiltonian is NP-complete. (If you do not manage to come up with a construction that works, describe your best attempt and explain why it does not work.)

**Exercise 8 (due 2014-10-20)**

Is the following problem NP-complete?

**Input:** A digraph $G = (V, E)$ and two nodes $u, v \in V$.

**Question:** Is there a path in $G$ from $u$ to $v$?

Note: One sentence is sufficient as an answer, but neither *yes* nor *no* alone qualify. Be careful and think! It is quite easy to answer this question too quickly and carelessly.

**Exercise 9 (due 2014-10-27)**

Recall the problem CLIQUE:

**Input:** A graph $G = (V, E)$ and a number $k$.

**Question:** Does $G$ have a $k$-clique, i.e., a complete subgraph on $k$ nodes?

Assume that we find a quadratic algorithm that solves CLIQUE. Now we want to solve the function problem that, given a graph $G$, returns a $k$-clique of $G$, where $k$ is the largest number such that $G$ has a $k$-clique.

Describe a polynomial algorithm that solves this problem and analyze its running time.