Data structures, exercise 3, 1-5 February

1. Continuing from last week's problem 4. You can calculate the sum 1+2+. . . +n recursively as follows:

   ```
   Sum(k)
       if k==0
           return 0
       else
           return sum( k-1 ) + k
   ```

   Define the time and space complexity for the algorithm with input n. Can you calculate the sum so that the time complexity is only $\mathcal{O}(1)$?

2. Write the algorithm sketched below as pseudo-code:

   > If there is a set of numbers in a table, the following algorithm searches for a way to divide the numbers into two groups so that the sum of the numbers in each group is as close to the other group as possible. The algorithms splits at each number according to two possibilities: the number goes into either group A or group B. The algorithm keps account of the sums of the groups and of divisions that produce the lowest difference.

3. Express the time complexity of the algorithm in the previous problem as a recursion equation

   Analyse in more detail what the time complexity of the algorithm is in relation to length of the input n.

   It is not necessary to solve the recursion equation on the basis of what you learn during this course. An easier way to make the analysis is to 'count' how many times the recoursive calls is executed in relation to the length of the input, and by observing the time complexity of the function body itself. The figure on page 70 in the handout will help you with this analysis. The result of problem 2 from the first week's exercises may also help.

4. $\mathcal{O}$-philosophy

   (a) Space complexity refers to how much memory the algorithm uses in addition to the input. It is not uncommon that the time complexity of an algorithm is $\mathcal{O}(n)$, but the space complexity only $\mathcal{O}(1)$. However, is it possible that an algorithm's time complexity is $\mathcal{O}(1)$ but its space complexity is $\mathcal{O}(n)$?

   (b) Let us look at a table with integers:

5	2	1	3	1

You can shrink the whole table into one integer by multiplying consecutive prime numbers with the numbers in the table as exponentials: $2^5 \cdot 3^2 \cdot 5^1 \cdot 7^3 \cdot 11^1 = 5433120$. You can find out the original numbers from this number by finding its primal elements and reading their exponentials. It would seem that it is a waste to use up five spaces in the table, when one would be enough:

5433120	-	-	-	-

The same method works with any table with n integers. It looks like the space complexity of the original table is $\mathcal{O}(n)$, while the new table requires only $\mathcal{O}(1)$. Is this the correct conclusion?

5. Show how to simulate a queue with the help of two stacks. Assume here that the queue and stacks are unlimited in size. What is the time complexity for the queue operations?

6. Show how to implement the queue as a linked structure so that the operations take constant time. Implementation here means a similar presentation in pseudo-code as the stacks on pages 93-95 in the handout, or a version written 'as clearly as possible' in some language. It is not allowed to use the existing libraries for real languages.

Do not forget to illustrate the function of the queue operations with figures etc.