Exercise Session 01

Solve the following exercises. The exercises that are more involved are marked with a star. If you need some guidance for solving the exercise, place your trash bin in front of your group room's door and the first available teaching assistant will come to help you out.

Exercise 1.
Consider the problem of finding the two smallest numbers in a nonempty sequence of numbers $\langle a_1, \ldots, a_n \rangle$ not necessarily sorted.

(a) Formalise the above as a computational problem (be careful to precisely define the input, the output, and their relationship).

(b) Write the pseudocode of an algorithm that solves the above computational problem assuming the sequence is given as an array $A[1\,..\,n]$.

(c) Assume that line i of your pseudocode takes constant time c_i to execute. What is the worst case running time of your algorithm?

Solution 1.
(a) In this exercise it is very important to define what the output must be, when there are numbers occurring more than once in the sequence. In the following, we define the output so that adding duplicates of any elements in the array does not change the answer.

Input: A sequence of $n > 0$ numbers $\langle a_1, \ldots, a_n \rangle$.

Output: A pair (min_1, min_2) such that $min_1 < min_2$ and

$$\begin{aligned} min_1 &= \min\{a_i \colon 1 \le i \le n\} \text{ and,} \\ min_2 &= \min\{a_i \colon 1 \le i \le n \text{ and } min_1 \ne a_i\} \end{aligned}$$

We will use the convention that $\min \emptyset = \infty$ to cover the case when no min_2 exists, which occurs when all values in the sequence are equal.

(b) Consider the following function FIND-2SMALLEST which takes an array $A[1\,..\,n]$ with $n \ge 1$.

```
FIND-2SMALLEST($A$)
1  $min_1 = A[1]$
2  $min_2 = \infty$
3  for $i = 2$ to $A.length$
4      if $A[i] < min_1$
5          $min_2 = min_1$
6          $min_1 = A[i]$
7      elseif $min_1 < A[i] < min_2$
8          $min_2 = A[i]$
9  return $(min_1, min_2)$
```

Please remember that there are many possible correct solutions, therefore if your own solution is different it might be still correct. We will discuss in the following lectures different techniques to prove correctness for an algorithm. Intuitively, the above algorithm maintains the property that $min_1 < min_2$ during the whole execution of the algorithm moreover it ensures that the following properties $min_1 = \min\{a_j \colon 1 \le j \le i\}$ and $min_2 = \min\{a_j \colon 1 \le j \le i \text{ and } min_1 \ne a_j\}$ before each iteration of the for loop. Therefore, when $i = A.length = n$ the property required for the output to be correct hods true.

(c) Let n be the size of the array A and c_i ($i = 1\,..\,9$) the time it takes to execute the statement i in FIND-2SMALLEST. The worst case occurs when the condition $A[i] < min_1$ holds true for each iteration of the for loop. This happens for instance when the sequence of elements is (strictly) increasing, that is $a_1 > a_2 > \cdots > a_n$. In this case the lines **??** and **??** are executed n times, leading to the following worst-case runtime for FIND-2SMALLEST:

$$\begin{aligned} T(n) &= c_3(n+1) + (c_4 + c_5 + c_6)n + c_1 + c_2 + c_9 \\ &= (c_3 + c_4 + c_5 + c_6)n + c_1 + c_2 + c_3 + c_9 \end{aligned}$$

Notice that the above is a linear function in n (i.e., of the form $an + b$ for some constants a and b depending on c_i ($i = 1, \ldots, 9$)).

Exercise 2.
Consider the following pseudocode for the function FIND-ELEMENT takes as input an array $A[1\,..\,n]$ and a number a.

```
FIND-ELEMENT($A, a$)
1  for $i = 1$ to $A.length$
2      if $A[i] = a$
3          return i
4  return i
```

(a) Count the number of iterations of the for loop in the above pseudocode for the execution of FIND-ELEMENT($[1, 0, 5, 2, 4], 5$) and report the value returned at the end of the call.

(b) Is the above algorithm solving the element search problem presented in class? Justify your answer.

Solution 2.
(a) When executing FIND-ELEMENT($[1, 0, 5, 2, 4], 5$) the for loop is executed 3 times because the element 5 is first found when the index counter i holds the value 3. Thus, the value returned by the call is 3.

(b) Recall that for an algorithm to be correct, we require that for any valid input instance it halts with the correct output. This is not the case for FIND-ELEMENT, since for example the call FIND-ELEMENT($[1], 3$) returns 1 instead of 0.

Exercise 3.
Write a pseudocode of an algorithm to reverse an array of numbers, i.e., the last element should become the first, the second last should become the second, etc. For example, the reverse of $[1, 2, 3, 4]$ is $[4, 3, 2, 1]$.

(a) What is the worst-case running time of your algorithm?

(b) Try now to propose an algorithm that use only a constant amount of extra space. What is the worst-case running time of your algorithm?

Solution 3.
We solve only the point (b) because its solution works in particular as a solution for (a). The algorithm is described below.

```
REVERSE($A$)
1  for $i = 1$ to $\lfloor \frac{A.length}{2} \rfloor$
2      $key = A[i]$
3      $A[i] = A[A.length + 1 - i]$
4      $A[A.length + 1 - i] = key$
```

The procedure REVERSE swaps the i-th element from the start of the array with the i-th element from the end, until its reached the index in the middle. Let c_i ($i = 1\,..\,4$) the cost of executing the i-th statement, then the worst-case running time of the procedure REVERSE is

$$T(n) = c_1(\lfloor n/2 \rfloor + 1) + (c_2 + c_3 + c_4)\lfloor n/2 \rfloor\,.$$

Exercise 4.
Look at the table in Exercise 1-1 in the textbook. Fill in the columns for 1 second and 1 minute. *Hint:* 1 microsecond = 10^{-6} seconds.

★ **Exercise 5.**
Let A and B be two arrays of numbers sorted in non-decreasing order, respectively of length n and m. Write the pseudocode of an algorithm that checks whether the set of elements in A is equal to the set of elements in B, i.e., all elements of A are contained in B and vice versa. What is the worst-case running time of your algorithm?

Solution 5.
We begin by formalising the corresponding computational problem

Input: Two sequences of numbers $\langle a_1, \dots, a_n\rangle$ and $\langle b_1, \dots, b_m\rangle$ sorted in non-decreasing order i.e., $a_1 \le a_2 \le \cdots a_n$ and $b_1 \le b_2 \le \cdots b_m$.

Output: *true* if and only if $\{a_i \colon 1 \le i \le n\} = \{b_i \colon 1 \le i \le m\}$.

For our algorithm we will use the usual respresentation of sequences by means to two arrays $A[1\,..\,n]$ and $B[1\,..\,m]$. Before start writing the pseudocode, we shall note that n and m may not be equal, and even if $n = m$, it does not suffice to check whether $A[i] = B[i]$ for all $1 \le i \le n$ since A and B may have repeated occurrences of the same numbers which are not necessarily positioned in the same index, e.g., $A = [1, 1, 2]$ and $B = [1, 2, 2]$ have the same set of elements but $A[2] = 1 \ne 2 = B[2]$.

```
EQUALSORTEDSETS($A, B$)
1   $i = 1$
2   $j = 1$
3   while $i \le A.length$ and $j \le B.length$ and $A[i] = B[j]$
4       while $i < A.length$ and $A[i] = A[i+1]$
5           $i = i + 1$
6       while $j < B.length$ and $B[j] = B[j+1]$
7           $j = j + 1$
8       $i = i + 1$
9       $j = j + 1$
10  return $i > A.length$ and $j > B.length$
```

The algorithm stores two indices i and j which point at the first occurrence on an element in the arrays A and B respectively. This property is maintained at right before each iteration of the outer while loop, by the two inner loops. Their job is move the indices forward until the last occurrence of an element is found. If the outer while loop terminates because both the indices exceeded the length of the corresponding array, then the two arrays represent the same set of numbers, otherwise one of the two arrays has an element which the other doesn't.

Before diving into the worst-case analysis, we observe that the size of the input depends both on m and n. Therefore we express the running time of EQUALSORTEDSETS as a function in two arguments, i.e., $T(n, m)$.

The worst-case running time occurs when the two arrays are equal, i.e.,

$$\{a_i \colon 1 \le i \le n\} = \{v_1, \dots, v_k\} = \{b_i \colon 1 \le i \le m\}$$

for some (pairwise distinct) numbers $v_1\,..\,v_k$. In this case, the number of iterations of the outer loop corresponds to k. We denote respectively by $A(v_i)$ and $B(v_i)$ the number of occurrences of the element

v_i $(i = 1 \,..\, k)$ in the arrays A and B. For each iteration of the outer loop, the two inner loops perform a number of iterations that corresponds to the amount of occurrences of the current element in each array.

$$\begin{aligned} T(n,m) =& c_1 + c_2 + (k+1)c_3 + c_4 \left(\sum_{i=1}^{k} A(v_i) + 1 \right) + c_5 \sum_{i=1}^{k} A(v_i) + \\ & + c_6 \left(\sum_{i=1}^{k} B(v_i) + 1 \right) + c_7 \sum_{i=1}^{k} B(v_i) + (c_8 + c_9)k + c_{10} \end{aligned}$$

note that $\sum_{i=1}^{k} A(v_i) = n$ and $\sum_{i=1}^{k} B(v_i) = m$, hence

$$= (c_4 + c_5)n + (c_6 + c_7)m + (c_3 + c_8 + c_9)k + \left(\sum_{i=1}^{4} c_i \right) + c_6 + c_{10}$$

At this point the formula still depends on k. The value of k is maximal when one of the two arrays has no repeated elements. In this case $k = \min\{n, m\}$ and the formula simplifies as follows.

$$= (c_4 + c_5)n + (c_6 + c_7)m + (c_3 + c_8 + c_9) \min\{n, m\} + \left(\sum_{i=1}^{4} c_i \right) + c_6 + c_{10} \,.$$