Advice on how to do the assignment

- Assignments should be typed and submitted as pdf (no pdf containing text as images, no handwriting).
- Start by typing your student ID at the top of the first page of your submission. Do **not** type your name.
- Submit only your answers to the questions. Do **not** copy the questions.
- When asked to give a plain English description, describe your algorithm as you would to a friend over the phone, such that you completely and unambiguously describe your algorithm, including all the important (i.e., non-trivial) details. It often helps to give a very short (1-2 sentence) description of the overall idea, then to describe each step in detail. At the end you can also include pseudocode, but this is optional.
- In particular, when designing an algorithm or data structure, it might help you (and us) if you briefly describe your general idea, and after that you might want to develop and elaborate on details. If we don't see/understand your general idea, we cannot give you marks for it.
- Be careful with giving multiple or alternative answers. If you give multiple answers, then we will give you marks only for "your worst answer", as this indicates how well you understood the question.
- Some of the questions are very easy (with the help of the slides or book). You can use the material presented in the lecture or book without proving it. You do not need to write more than necessary (see comment above).
- When giving answers to questions, always prove/explain/motivate your answers.
- When giving an algorithm as an answer, the algorithm does not have to be given as (pseudo-)code.
- If you do give (pseudo-)code, then you still have to explain your code and your ideas in plain English.
- Unless otherwise stated, we always ask about worst-case analysis, worst case running times, etc.
- As done in the lecture, and as it is typical for an algorithms course, we are interested in the most efficient algorithms and data structures.
- If you use further resources (books, scientific papers, the internet,...) to formulate your answers, then add references to your sources and explain it in your own words. Only citing a source doesn't show your understanding and will thus get you very few (if any) marks. Copying from any source without reference is considered plagiarism.

As a first step go to the last page and read the section: Advice on how to do the assignment.

Problem 1. (10 points)
Using O-notation, upperbound the running time of the following algorithm, where A is an array containing n integers. You can assume that mod is a basic mathematical operation that takes $O(1)$ time.

```
1: function ALGORITHM(A)
2:     b ← true
3:     for i ← 0; 2i < n; i++ do
4:         if A[2i] mod 2 ≠ 0 then
5:             b ← false
6:     for i ← 0; 2i < n − 1; i++ do
7:         if A[2i + 1] mod 2 ≠ 1 then
8:             b ← false
9:     return b
```

Problem 2. (25 points)
We want to build a stack for integer elements that in addition to the operations we saw during the lecture (PUSH(e), POP(), TOP(), SIZE(), and ISEMPTY()), also supports a GETMINIMUM() operation that returns the minimum value of all elements stored in the stack. All operations should run in $O(1)$ time. Your data structure should take $O(n)$ space, where n is the number of elements currently stored in the data structure.

Example execution:

PUSH(23)		[23]
PUSH(4)		[23, 4]
GETMINIMUM()	returns 4	[23, 4]
POP()	returns 4	[23]
GETMINIMUM()	returns 23	[23]

Your task is to:

a) Design a data structure that supports the required operations in the required time and space.

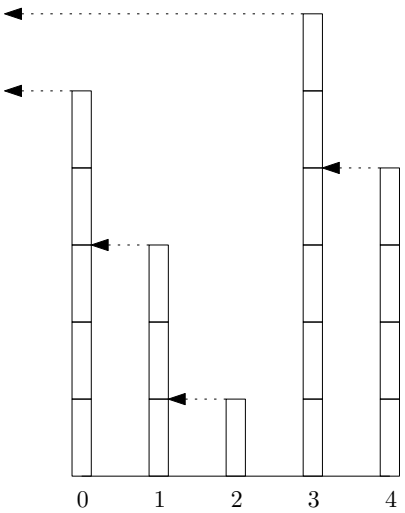
b) Briefly argue the correctness of your data structure and operations.

c) Analyse the running time of your operations and space of your data structure.

Problem 3. (25 points)
We have $n \geq 1$ lighthouses of distinct positive integer height on a line and these lighthouses need to communicate with each other. We are given the heights of these lighthouses in an array A, where $A[i]$ stores the height of lighthouse i. Unfortunately, the area these lighthouses are in is so remote that it doesn't have phone lines and cell phone coverage is spotty at best. So the lighthouse keepers have found a different way to talk to each other: paper airplanes.

Assume all lighthouse keepers are (former) olympic champions in paper airplane throwing, meaning that they can reach any lighthouse as long as there's no lighthouse higher than the thrower's lighthouse between them (the lighthouses are equipped with paper airplane catchers, so hitting the wall of the lighthouse counts as reaching it). Of course, because of the wind coming from the ocean, lighthouse keepers can only send their paper planes to the left (towards the land). More formally, lighthouse k can send a paper airplane to lighthouse i ($i < k$), if there is no lighthouse j such that $i < j < k$ and $A[k] < A[j]$. Note that lighthouse i itself can be higher than lighthouse k because of the paper airplane catchers. We are interested in determining the leftmost lighthouse that each of the lighthouse keepers can reach. To distinguish between a paper airplane stopping at lighthouse 0 and passing over all lighthouses to the left of its starting point, we record the latter as -1 (imagining an infinitely high lighthouse preceding all others where the airplane stops).

Example:



$A : [5, 3, 1, 6, 4]$: return $[-1, 0, 1, -1, 3]$. A paper airplane thrown by the lighthouse keeper of lighthouse 0 passes over all lighthouses to its left and is thus recorded as -1. Lighthouse 1 can send their airplanes to lighthouse 0, but not beyond. Lighthouse 2 can communicate with lighthouse 1, but not with lighthouse 0, since lighthouse 1 stops any airplanes throw from lighthouse 2. Lighthouse 3 is higher than any of the previous lighthouses and thus its airplanes pass over them. Finally, lighthouse 4 can send airplanes to lighthouse 3, but no further.

Your task is to design an algorithm that computes for each lighthouse the leftmost lighthouse that its keeper can reach. For full marks, your algorithm should

run in $O(n)$ time.

a) Describe your algorithm in plain English.

b) Prove the correctness of your algorithm.

c) Analyze the time complexity of your algorithm.