- This problemset has *two* questions. The programming problems may be harder, or require more time, than their point value suggests.
- Please send the solutions to `gawry1+EDSCourse2014@gmail.com`
- The deadline is 31.05.2014 anywhere on Earth.

(20) 1. As input we are given an array A of elements $a_1, \ldots, a_n$, where element a_i occupies $1 \leq b_i \leq \ell$ bits, where $\ell \ll n$. We wish to support the following operations on A: `Access(i)` which returns element a_i, and `Modify(i, a_i', b_i')` which replaces elements a_i with a new element a_i' that occupies $1 \leq b_i' \leq \ell$ bits. Suppose we are given a dynamic bit vector data structure as a black box which can store u bits using $u + o(u)$ space, and support the operations: `Rank(i)`, `Select(i)`, `Access(i)`, `Insert(i,{0,1})`, and `Delete(i)` all in $O(Q(u))$ time. Let $m = \sum_i b_i$ be the current total number of bits in A. Show how to represent A using $2m + o(m)$ bits, and $O(\ell Q(m))$ time for `Access(i)`, and $O(\ell Q(m))$ time for `Modify(i, a_i', b_i')`.

(60) 2. We want to solve the following problem: preprocess a collection of numbers $0 \leq a_1 < a_2 < \ldots < a_n < U$ so that later, given an interval $[\text{from}, \text{to}]$, we can either find some $a_i \in [\text{from}, \text{to}]$ or detect that there are none. If there are multiple such a_i's, we can return any of them! This is sometimes called an IfAny structure. As in the lecture, we assume that n and every a_i fit into a single machine word.

(a) Assume that we can implement an IfAny structure in linear space and **constant** time per query. Show how to construct an efficient structure for 1D range reporting, which is to preprocess a collection of numbers so that later, given an interval $[\text{from}, \text{to}]$, we can report all numbers inside the interval in the sorted order. The size of the new structure should be linear and the query time should be $\mathcal{O}(1+k)$, where k is the number of reported points.

(b) Show that an IfAny structure can be implemented in $\mathcal{O}(n \log U)$ space and constant time per query. Imagine a binary trie as in the x-fast trees. To answer a query, find the node corresponding to the longest common prefix of the binary representations of from and to. Argue that if there is no such node in the trie, we don't have to return anything. In the other case, show how to store a constant amount of data at the node, so that we can answer a query in constant time.

(bonus) (c) Can you decrease the above space complexity? As far as I know the simple indirection doesn't help here.