# Introduction. Greedy Idea. (Cycle's) Invariant

**1** [ Shen 1.2.21 ]. Two arrays `x[1]` $\leq \ldots \leq$ `x[k]` and `y[1]` $\leq \ldots \leq$ `y[l]` are given. Find their "intersection", i.e., an array `z[1]` $\leq \ldots \leq$ `z[m]` that contains their common elements. The multiplicity of each element in `z` should be equal to the smaller of its multiplicities in `x` and `y`. The number of operations should be of order `k` + `l`.

**2.** There is an array of pairs $[(l_1, r_1), \ldots, (l_n, r_n)]$. A pair $(l_i, r_i)$ defines a segment $[l_i, r_i]$ on a line. Construct an $O(n \log n)$ algorithm that finds a set of maximal size that contains non-intersecting segments (from the array). The output is an array of pairs $(l_i, r_i)$.

**3.** There are two arrays $[a_1, \ldots, a_n]$ and $[b_1, \ldots b_n]$. Construct an $O(n)$ algorithm that computes the sum $\sum\limits_{i \neq j} a_i \times b_j$.

**4.** Construct an $O(n)$ algorithm that sorts a binary array, i. e. each element is either 0 or 1.

**5** [ Hoar's Partition ]. An array $a = [a_1, \ldots, a_n]$ is stored in RAM. The input of the problem is a predicate $P$ such that $P(a[i])$ is computed in $O(1)$. Construct an algorithm that swaps elements in the array so that for some $k \geqslant 0$ the assertions $P([a_i]) = 0, i \leqslant k$ and $P([a_i]) = 1, i > k$ hold. The algorithm shall have the $O(n)$ time complexity and use $O(1)$ additional RAM (i.e., finite number of variables). So it is not allowed to copy the array.

**6.** The input of the problem is an array $a = [a_1, \ldots, a_n]$ and a number $S$. Construct an algorithm that verifies whether there are elements $a_i$ and $a_j$ $(i \neq j)$ such that $a_i + a_j = S$. Consider the following variations of the problem.

1. Construct an $O(n \log n)$ algorithm.

2. Assume that you have a data structure Set such that you can add an element $x$ to the set in $O(1)$ and you can test, whether $x$ belongs to the set in $O(1)$. Construct an $O(n)$ algorithm using Set.

3. Assume that the array $a$ has been already sorted. Construct an $O(n)$ algorithm that uses $O(1)$ additional RAM.

**7.** An array $A[1...n]$ is said to have a majority element if more than half of its entries are the same. Given an array, the task is to design an efficient algorithm to tell whether the array has a majority element, and, if so, to find that element. The elements of the array are not necessarily from some ordered domain like the integers, and so there can be no comparisons of the form "is $A[i] > A[j]$?". (Think of the array elements as GIF files, say.) However you can answer questions of the form: "is $A[i] = A[j]$?" in constant time. Construct an effective algorithm that finds out the majority element of an array or verifies that it does not exists. There are $O(n \log n)$ and $O(n)$ algorithms for this problem.