Examples of examination problems

1. For what values of parameter '*a*' the system of linear equations has a unique solution in positive natural numbers (x = 1, 2, ...)? Find this solution for all feasible values of '*a*'.

$$\begin{cases} x_1 + ax_2 = 3 \\ x_1 + 2x_2 = 3 \end{cases}$$

Solution:

Determinant of this system is equal to:

$$\det A = \begin{vmatrix} 1 & a \\ 1 & 2 \end{vmatrix} = 2 - a$$

Let us consider two cases: det $A = 0$ and det $A \neq 0$:

I. det $A = 2 - a = 0$ => $a = 2$

$$\begin{cases} x_1 + 2x_2 = 3 \\ x_1 + 2x_2 = 3 \end{cases} \Leftrightarrow \begin{cases} x_1 = 3 - 2x_2 \geq 1 \\ x_2 \geq 1 \end{cases} \Leftrightarrow \begin{cases} x_1 = 3 - 2x_2 \\ x_2 \leq 1 \\ x_2 \geq 1 \end{cases} \Leftrightarrow \begin{cases} x_1 = 1 \\ x_2 = 1 \end{cases}$$

This is a unique solution in positive natural numbers.

II. det $A = 2 - a \neq 0$ => $a \neq 2$

Using Gauss method we get zeroes under the main diagonal:

$$\left(\begin{array}{cc|c} 1 & a & 3 \\ 1 & 2 & 3 \end{array}\right) \sim \left(\begin{array}{cc|c} 1 & a & 3 \\ 0 & 2-a & 0 \end{array}\right)$$

$$\begin{cases} x_1 + ax_2 = 3 \\ (2-a)x_2 = 0 \end{cases} \Leftrightarrow \begin{cases} x_1 = 3 \\ x_2 = 0 \end{cases}$$

This is a unique solution, but it is not positive.

Answer: the system has a unique solution in positive natural numbers only for a = 2, this solution is $x_1 = 1$, $x_2 = 1$.

Wrong solution:

$$\begin{cases} x_1 + ax_2 = 3 \\ x_1 + 2x_2 = 3 \end{cases} \Leftrightarrow \begin{cases} x_1 = 3 - ax_2 \\ 3 - ax_2 + 2x_2 = 3 \end{cases} \Leftrightarrow \begin{cases} x_1 = 3 - ax_2 \\ (2-a)x_2 = 0 \end{cases} \Leftrightarrow \begin{cases} x_1 = 3 \\ x_2 = 0 \end{cases}$$

Wrong answer 1: For any '*a*' value we have a unique solution in natural numbers.

Wrong answer 2: This is not a positive solution, so there no such values of '*a*', for which the system has a solution in positive natural numbers.

2. Independent random variables *X* and *Y* have the following distributions:

$$X \sim \begin{array}{c|c} 0 & 1 \\ \hline 1/3 & 2/3 \end{array}, \quad Y \sim \begin{array}{c|c|c} 0 & 1 & 2 \\ \hline 1/3 & 1/3 & 1/3 \end{array}$$

Find the expectation and dispersion of the random variable $Z = (Y - X)^2$.

Solution 1:

$\mathrm{E}[Y]=1/3+2/3=1,\ \mathrm{E}[Y^2]=1/3+4/3=5/3,\ \mathrm{E}[Y^3]=1/3+8/3=3,\ \mathrm{E}[Y^4]=1/3+16/3=17/3$

$\mathrm{E}[X]=2/3,\quad \mathrm{E}[X^2]=2/3,\quad \mathrm{E}[X^3]=2/3,\quad \mathrm{E}[X^4]=2/3$

$\mathrm{E}[(Y-X)^2]=\mathrm{E}[Y^2-2YX+X^2]=\mathrm{E}[Y^2]-2\mathrm{E}[Y]\mathrm{E}[X]+\mathrm{E}[X^2]$

$\mathrm{E}[(Y-X)^2]=5/3-2\cdot1\cdot2/3+2/3=1$

$\mathrm{D}[(Y-X)^2]=\mathrm{E}[(Y-X)^4]-\left(\mathrm{E}[(Y-X)^2]\right)^2=\mathrm{E}[Y^4-4Y^3X+6Y^2X^2-4YX^3+X^4]-\left(\mathrm{E}[(Y-X)^2]\right)^2$

$\mathrm{D}[(Y-X)^2]=(17/3-4\cdot3\cdot2/3+6\cdot5/3\cdot2/3-4\cdot1\cdot2/3+2/3)-1=7/3-1=4/3$

<u>Solution 2</u>:

Let us find *Z* values for all possible combinations of *X* and *Y* and their probabilities *P*:

$Z=(Y-X)^2$

$X\backslash Y$	0	1	2
0	0	1	4
1	1	0	1

,

P

$X\backslash Y$	0	1	2
0	1/9	1/9	1/9
1	2/9	2/9	2/9

From this we get the distribution for *Z*, its expectation, and dispersion:

$Z\sim\begin{array}{c|c|c}0&1&4\\\hline 1/3&5/9&1/9\end{array},\ \mathrm{E}[Z]=5/9+4/9=1,\ \mathrm{D}[Z]=\mathrm{E}[Z^2]-(\mathrm{E}[Z])^2=5/9+16/9-1=4/3$

<u>Wrong solution 1</u>:

$\mathrm{E}[Y]=1/3+2/3=1,\ \mathrm{E}[X]=2/3$

$\mathrm{E}[(Y-X)^2]=\mathrm{E}[Y-X]\cdot\mathrm{E}[Y-X]=(\mathrm{E}[Y]-\mathrm{E}[X])(\mathrm{E}[Y]-\mathrm{E}[X])=(1-2/3)(1-2/3)=1/9$

$\mathrm{D}[Y]=\mathrm{E}[Y^2]-(\mathrm{E}[Y])^2=1/3+4/3-1=2/3,\ \mathrm{D}[X]=\mathrm{E}[X^2]-(\mathrm{E}[X])^2=2/3-4/9=2/9$

$\mathrm{D}[(Y-X)^2]=\mathrm{D}[Y-X]\cdot\mathrm{D}[Y-X]=(\mathrm{D}[Y]-\mathrm{D}[X])(\mathrm{D}[Y]-\mathrm{D}[X])$

$\mathrm{D}[(Y-X)^2]=(2/3-2/9)(2/3-2/9)=16/81$

<u>Wrong solution 2</u>:

$\mathrm{D}[(Y-X)^2]=\mathrm{D}[Y^2-2YX+X^2]=\mathrm{D}[Y^2]+4\mathrm{D}[Y]\mathrm{D}[X]+\mathrm{D}[X^2]$

In the following problems it is necessary to suggest the most efficient algorithms. Full points are given for the most efficient algorithm having the lowest computational complexity. The lower the efficiency of the suggested solution, the lower are the points. Например, в последних 2 задачах полный балл ставится за алгоритм со сложностью O(n), средний балл - за O(nlogn) и низший балл за O(n^2).

3. Write a pseudo-code (or code on any programming language) of an algorithm, which finds two elements in an array of integer numbers (positive and negative), such that the product of these elements is maximal. Determine the computational complexity of your algorithm.

<u>Solution</u>:

<u>Bad</u>:

```
get_elements_with_max_product(array):
    max, elem1, elem2 = -inf, 0, 0
    for i = 0, …, |array| - 1
        for j = i+1, …, |array| - 1
            if array[i] * array[j] > max
                max, elem1, elem2 = array[i] * array[j], array[i], array[j]
    return elem1, elem2
```

The computational complexity of this solution is $O(n^2)$, because there are two nested loops with $O(n)$ iterations. More precisely we have $n + n-1 + \ldots + 1 = n(n+1)/2$ operations. Here n is the size of the array.

Normal:

Note that in array [-5, -3, 1, 1, 2, 4] maximal product is (-5) * (-3) = 15 > 8 = 2 * 4. This means that maximal product can be obtained from two minimal elements instead of the two maximal. In the previous algorithm this was not the problem because we checked all the possible pairs.

```
get_elements_with_max_product(array):
    sort(array)
    max_for_two_negative_elements = array[0] * array[1]
    n = |array|
    max_for_two_positive_elements = array[n-2] * array[n-1]
    if max_for_two_positive_elements > max_for_two_negative_elements
        return array[n-2], array[n-1]
    return array[0], array[1]
```

The computational complexity of this solution is $O(n\log n)$ because of sort() procedure.

Good:

```
get_elements_with_max_product(array):
    min1, min2, max1, max2 = +inf, +inf, -inf, -inf
    for i = 0, …, |array| - 1
      if array[i] < min1
          min2, min1 = min1, array[i]
      else if array[i] < min2
          min2 = array[i]
      if array[i] > max1
          max2, max1 = max1, array[i]
      else if array[i] > max2
          max2 = array[i]
    max_for_two_negative_elements = min1 * min2
    max_for_two_positive_elements = max1 * max2
    if max_for_two_positive_elements > max_for_two_negative_elements
        return max2, max1
    return min1, min2
```

The computational complexity of this solution is $O(n)$, because there is only one loop with n iterations.

4. Write a pseudo-code (or code on any programming language) of an algorithm, which finds a leaf in a tree, which is the farthest leaf from the head of this tree. A tree is given by its *head* and every tree node has a list of its children referenced as *node.children*. For example, *head.children* contains and array of the child nodes of the tree head. Determine the computational complexity of your algorithm.

Solution:

```
get_farthest_leaf(head):
    farthest_leaf, max_distance = head, -inf
    get_farthest_leaf_recursive(head, 0, farthest_leaf, max_distance)
    return farthest_leaf


get_farthest_leaf_recursive (node, distance, farthest_leaf, max_distance)
    if node.children = $\varnothing$
        if distance > max_distance
            farthest_leaf, max_distance = node, distance
        return
    for child $\in$ node.children
        get_farthest_leaf_from(node, distance + 1, farthest_leaf, max_distance)
```

This algorithm recursively traverses the tree by means of depth-first search. It will check every node of the tree exactly once whether it is a leaf (*node.children* = ∅) and a distance to it is longer than the current maximal distance (*max_distance*). So its computational complexity is $O(n)$, where n is the number of the tree nodes.

A bad solution here with $O(n^2)$ complexity could be to go to every leaf node separately each time starting from the head.