BABEŞ-BOLYAI UNIVERSITY
FACULTY OF MATHEMATICS AND COMPUTER SCIENCE

Math & CompSci Contest - model

Written Test in Computer Science

Important observations for candidates:

1. All arrays are indexed starting from 1.
2. Multiple answer problem statements (Part A) can have one or more correct answers. Candidates must write the answers on their contest sheet (not the sheets with the problem statements). Solutions are graded only if they identify all the correct answers, and only them.
3. Complete solutions are expected for problem statements in Part B.
   - **a.** Solutions will be described using *pseudocode* or a *programming language* (*Pascal/C/C++).*
   - **b.** The first criterion when evaluating solutions is algorithm ***correctness***, followed by its ***performance*** regarding *time of execution* and *required memory space*.
   - **c.** ***Providing description and justification*** for all (sub)algorithms before their implementation is ***mandatory***. In addition, ***comments*** will be used to describe details of the technical implementation, meaning of used identifiers, data structures and so on. Failure to observe these requirements leads to a 10% penalty from the problem score.
   - **d.** Using predefined functions or libraries is prohibited (for example: *STL*, predefined string functions).

Part A (60 points)

A.1. What does the following subalgorithm do? (6 points)

Consider the `alg(x, b)` subalgorithm, with input parameters natural numbers ***x*** and ***b*** ($1 \leq x \leq 1000$, $1 < b \leq 10$).

```
Subalgorithm alg(x, b):
    s ← 0
    While x > 0 do
        s ← s + x MOD b
        x ← x DIV b
    EndWhile
    return s MOD (b - 1) = 0
EndSubalgorithm
```

What is the effect of the subalgorithm above.

A. calculates the sum of ***x***'s digits in base ***b***.
B. checks whether the sum of ***x***'s digits in base ***b – 1*** is divisible with ***b – 1.***
C. checks whether natural number ***x*** is divisible with ***b – 1***.
D. checks whether the sum of ***x***'s digits in base ***b*** is divisible with ***b – 1.***

A.2. What will be displayed? (6 points)

Consider the following program:

C++/C Version

```
int sum(int n, int a[], int s){
   s = 0;
   int i = 1;
   while(i <= n){
      if(a[i] != 0) s += a[i];
      ++i;
   }
   return s;
}


int main(){
   int n = 3, p = 0, a[10];
   a[1] = -1; a[2] = 0; a[3] = 3;
   int s = sum(n, a, p);
   cout << s << ";" << p; // printf("%d;%d", s, p);
   return 0;
}
```

Pascal Version

```
type vector = array[1..10] of integer;

function sum(n:integer; a:vector; s:integer):integer;
  var i : integer;
  begin
     s := 0; i := 1;
     while (i <= n) do
        begin
           if (a[i] <> 0) then s := s + a[i];
           i := i + 1;
        end;
     sum := s;
  end;
var n, p, s : integer;
          a : vector;
begin
    n := 3; a[1] := -1; a[2] := 0; a[3] := 3; p := 0;
    s := sum(n, a, p);
    writeln(s, ';', p);
end.
```

What result is displayed after running the program?

A. `0;0`
B. `2;0`
C. `2;2`
D. None of the above

A.3. Logical Expression (6 points)

Consider the following logical expression `(X OR Z) AND (NOT X OR Y)`. Choose the values for ***X***, ***Y***, ***Z*** such that the expression is `TRUE`:

A. ***X*** ← FALSE; ***Y*** ← FALSE; ***Z*** ← TRUE;
B. ***X*** ← TRUE; ***Y*** ← FALSE; ***Z*** ← FALSE;
C. ***X*** ← FALSE; ***Y*** ← TRUE; ***Z*** ← FALSE;
D. ***X*** ← TRUE; ***Y*** ← TRUE; ***Z*** ← TRUE;

A.4. Calculate (6 points)

Consider the `calculate(a, b)` subalgorithm, with input parameters ***a*** and ***b*** natural numbers, $1 \leq a \leq 1000$, $1 \leq b \leq 1000$.

```
1.      Subalgorithm calculate(a, b):
2.          If a ≠ 0 then
3.              return calculate(a DIV 2, b + b) + b * (a MOD 2)
4.          EndIf
5.          return 0
6.      EndSubalgorithm
```

Which of the statements below are false?

A. if *a* and *b* are equal, the subalgorithm returns the value of *a*.
B. if ***a*** = 1000 and ***b*** = 2, the subalgorithm calls itself 10 times.
C. the calculated and returned value is ***a*** / 2 + 2 * ***b***
D. the instruction on line 5 is never executed

A.5. Element identification (6 points)

Consider the series (1, 2, 3, 2, 5, 2, 3, 7, 2, 4, 3, 2, 5, 11, ...), built as follows: starting from the natural numbers' series, replace each non-prime number with its proper divisors, with each divisor *d* considered once per number. Which of the following subalgorithms determines the ***n***-th element of this series (***n*** – natural number, $1 \leq n \leq 1000$)?

A.
```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        a ← a + 1, b ← a, c ← c + 1, d ← 2
        f ← false
        While c ≤ n and d ≤ a DIV 2 do
            If a MOD d = 0 then
                c ← c + 1, b ← d, f ← true
            EndIf
            d ← d + 1
        EndWhile
        If f then
            c ← c - 1
        EndIf
    EndWhile
    return b
EndSubalgorithm
```

B.
```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        c ← c + 1, d ← 2
        While c ≤ n and d ≤ a DIV 2 do
            If a MOD d = 0 then
                c ← c + 1, b ← d
            EndIf
            d ← d + 1
        EndWhile
        a ← a + 1, b ← a
    EndWhile
    return b
EndSubalgorithm
```

C.
```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        a ← a + 1, d ← 2
        While c < n and d ≤ a do
            If a MOD d = 0 then
                c ← c + 1, b ← d
            EndIf
            d ← d + 1
        EndWhile
    EndWhile
    return b
EndSubalgorithm
```

D.
```
Subalgorithm identification(n):
    a ← 1, b ← 1, c ← 1
    While c < n do
        b ← a, a ← a + 1, c ← c + 1, d ← 2
        While c ≤ n and d ≤ a DIV 2 do
            If a MOD d = 0 then
                c ← c + 1, b ← d
            EndIf
            d ← d + 1
        EndWhile
    EndWhile
    return b
EndSubalgorithm
```

A.6. Prime factors (6 points)

Consider the `primeFactors(n, d, k, x)` subalgorithm that determines the ***k*** prime factors for natural number ***n***, starting the search from the value ***d***. Input parameters are natural numbers ***n***, ***d*** și ***k***, and the output parameter is array ***x*** consisting of the ***k*** prime factors ($1 \leq n \leq 10000$, $2 \leq d \leq 10000$, $0 \leq k \leq 10000$).

```
Subalgorithm primeFactors(n, d, k, x):
    If n MOD d = 0 then
        k ← k + 1
        x[k] ← d
```

```
    EndIf
    While n MOD d = 0 do
        n ← n DIV d
    EndWhile
    If n > 1 then
        primeFactors(n, d + 1, k, x)
    EndIf
EndSubalgorithm
```

How many times will the `primeFactors(n, d, k, x)` subalgorithm call itself by executing the following instruction sequence:

```
    n ← 120
    d ← 2
    k ← 0
    primeFactors(n, d, k, x)
```

A. 3 times
B. 5 times
C. 9 times
D. the same number of times as in the following sequence:

```
    n ← 750
    d ← 2
    k ← 0
    primeFactors(n, d, k, x)
```

A.7. What does the following subalgorithm do? (6 points)

Consider the `expression(n)` subalgorithm, with ***n*** a natural number ($1 \leq \boldsymbol{n} \leq 10000$).

```
Subalgorithm expression(n):
    If n > 0 then
        If n MOD 2 = 0 then
            return -n * (n + 1) + expression(n - 1)
        else
            return n * (n + 1) + expression(n - 1)
        EndIf
    else
        return 0
    EndIf
EndSubalgorithm
```

What is the mathematical form of the *E(n)* expression calculated by this subalgorithm:

A. $E(n) = 1* 2 - 2 * 3 + 3 * 4 + ... + (-1)^{n+1} * n * (n + 1)$
B. $E(n) = 1* 2 - 2 * 3 + 3 * 4 + ... + (-1)^{n} * n * (n + 1)$
C. $E(n) = 1* 2 + 2 * 3 + 3 * 4 + ... + (-1)^{n+1} * n * (n + 1)$
D. $E(n) = 1* 2 + 2 * 3 + 3 * 4 + ... + (-1)^{n} * n * (n + 1)$

A.8. Logical expression (6 points)

Consider the following logical expression: `(NOT Y OR Z) OR (X AND Y)`. Choose values for ***X***, ***Y***, ***Z*** so that the result of evaluating it is *TRUE*:

A. ***X*** ← *FALSE*; ***Y*** ← *FALSE*; ***Z*** ← *FALSE*;
B. ***X*** ← *FALSE*; ***Y*** ← *FALSE*; ***Z*** ← *TRUE*;
C. ***X*** ← *FALSE*; ***Y*** ← *TRUE*; ***Z*** ← *FALSE*;
D. ***X*** ← *TRUE*; ***Y*** ← *FALSE*; ***Z*** ← *TRUE*;

A.9. Input parameter values (6 points)

Consider the following subalgorithm:

```
Subalgorithm SA9(a):
        If a < 50 then
                If a MOD 3 = 0 then
                        return SA9(2 * a - 3)
                else
                        return SA9(2 * a - 1)
              EndIf
        else
                return a
        EndIf
EndSubalgorithm
```

For which values of input parameter ***a*** will the subalgorithm return 61?

A. 16

B. 61
C. 4
D. 31

A.10. Missing instructions (6 points)

Consider the following subalgorithm:

```
1:    Subalgorithm search(x, n, val):
2:          If n = 1 then
3:                return (x[1] = val)
4:          else
5:                return search(x, n - 1, val)
6:          EndIf
7:    EndSubalgorithm
```

Which instruction(s) must be added and where so that `search(x, n, val)` determines whether element ***val*** is part of array ***x*** with ***n*** elements (***n*** strictly positive natural number)?

A. Line 5 must be changed to: `return ((x[n] = val) and search(x - 1, n, val))`
B. Line 5 must be changed to: `return ((x[n] = val) or search(x, n - 1, val))`
C. Line 5 must be changed to: `if (x[n] = val) then return` *true* `else return search(x, n - 1, val)`
D. No changes are required

Part B (30 points)

1. Prefix (15 points)

A natural number's *control digit* is determined by calculating the sum of its digits, then the sum of its digits' sum and so on until we obtain a single digit number. For example, the *control digit* of 182 is 2 (1 + 8 + 2 = 11, 1 + 1 = 2).

A number ***p*** of exactly ***k*** digits is a *prefix* for a number ***q*** of at least ***k*** digits if the number created by ***q***'s first ***k*** digits, from left to right is equal to ***p***. For example, 17 is a prefix for 174, and 1713 is a prefix for 1 713 242.

Consider natural number ***nr*** ($0 < nr \leq 30\,000$) and matrix ***A*** with ***m*** rows and ***n*** columns ($0 < m \leq 100$, $0 < n \leq 100$), having natural number elements smaller than 30 000. Consider subalgorithm `controlDigit(x)` to determine the control digit for number ***x:***

```
Subalgorithm controlDigit(x):
    s ← 0
    While x > 0 do
        s ← s + x MOD 10
        x ← x DIV 10
        If x = 0 then
            If s < 10 then
                return s
            else
                x ← s
                s ← 0
            EndIf
        EndIf
    EndWhile
    return s
EndSubalgorithm
```

Requirements:

a. Write a *recursive* version (without repetitive structures) for subalgorithm `controlDigit(x)` having the same signature and effect. (**5 points**)
b. Write the mathematical model for the recursive version of subalgorithm `controlDigit(x)` developed at point a. (**3 points**)
c. Write a subalgorithm that determines the longesst ***prefix*** for number ***nr*** that can be built using the control digits of the elements of the given matrix. Control digits can be used more than once when building the prefix. If building a prefix is not possible, then ***prefix*** takes the value -1. The subalgorithm's input parameters are numbers ***nr***, ***m***, ***n***, matrix ***A***, and the output parameter is ***prefix***. (**7 points**)

Example*:** if ***nr = 12319, ***m*** = 3 and ***n*** = 4 and matrix $A = \begin{pmatrix} 182 & 12 & 274 & 22 \\ 22 & 1 & 98 & 56 \\ 5 & 301 & 51 & 94 \end{pmatrix}$,

the longest prefix is ***prefix*** = 1231, with the following control digits:

Matrix element	182	12	274	22	1	98	56	5	301	51	94
Control digit	2	3	4	4	1	8	2	5	4	6	4

2. Robo-garden (15 points)

A gardener with a passion for technology decides to use a swarm of robots to irrigate the flower beds in his garden. He wishes to use water from the spring situated at the end of the main path which crosses the garden. Each garden bed has its own robot, and each robot must water a single garden bed. All the robots start from the spring to water the garden beds at the same time in the morning (for example, at 5:00:00) and work in parallel, without stopping, for a given amount of time. They go through the main path until reaching their flower bed, which they water and then return to the spring to refill their water supply. When the time runs out, all the robots stop immediately, regardless of their current state. Initially, there is a single tap available at the spring. The gardener notices delays in the watering schedule due to the robots having to wait in line for refilling their water supply. As such, he considers that the best solution is to install additional water taps. Each morning, all robots start the day with a full water supply. Two robots cannot fill their water supply from the same tap at the same time.

Known: the time interval ***t*** during which the ***n*** robots work (expressed in seconds), the number of seconds d_i required for the robot to go from the tap to its flower bed, u_i - the number of seconds required to water the flower bed as well as that refilling the water tank requires 1 second for each robot (***t***, ***n***, d_i, u_i – natural numbers, $1 \leq t \leq 20000$, $1 \leq n \leq 100$, $1 \leq d_i \leq 1000$, $1 \leq u_i \leq 1000$, $i = 1, 2, ..., n$).

Requirements:

a. List the robots that will meet at the spring at a certain moment in time ***mt*** ($1 \leq mt \leq t$). Justify your answer. Note: robots are identified using their index. (**3 points**)

b. What is the minimum number of additional taps ***minAdditionalTaps*** that the gardener must install in order to ensure that none of the robots have to wait in line when refilling their water tank? Justify your answer. (2 **points**)

c. Write a subalgorithm to determine the minimum number of additional taps ***minAdditionalTaps.*** Input parameters are numbers ***n*** and ***t***, arrays ***d*** and ***u*** having ***n*** elements each, and the output parameter is ***minAdditionalTaps***. (**10 points**)

Example 1: if ***t*** = 32, ***n*** = 5, ***d*** = (1, 2, 1, 2, 1), ***u*** = (1, 3, 2, 1, 3) then ***minAdditionalTaps*** = 3. Explanation: the robot responsible for flower bed 1 needs 1 second to reach it, one second to water it, and 1 second to return to the spring; it returns to the spring to refill its supply after 1 + 1 + 1 = 3 seconds from the starting time (5:00:00), so at 5:00:03; the robot refills its supply in one second and starts toward the flower bed at 5:00:04; it returns at 5:00:07 to refill its tank, and continues its work; as such, the first, second, fourth and fifth robots meet at the spring at 05:00:23; as such, the gardener must install 3 additional taps.

Example 2: if ***t*** = 37, ***n*** = 3, ***d*** = (1, 2, 1), ***u*** = (1, 3, 2), then ***minAdditionalTaps*** = 1.

Note:

1. All subjects are mandatory.
2. Drafts are not taken into consideration for grading.
3. Default 10 points.
4. Working time is 3.5 hours.

GRADING & SOLUTIONS
DEFAULT ..... 10 points

Part A ..... 60 points
A. 1. C, D ..... **6 points**
A. 2. B ..... **6 points**
A. 3. A, D ..... **6 points**
A. 4. A, C, D ..... **6 points**
A. 5. A ..... **6 points**
A. 6. A, D ..... **6 points**
A. 7. A ..... **6 points**
A. 8. A, B, D ..... **6 points**
A.9. A, B, D ..... **6 points**
A.10. B, C ..... **6 points**

Part B ..... 30 points
B. 1. Prefix ..... 15 points
B.1.a. recursive version of the `controlDigit(x)` **subalgorithm ..... 5 points**
– correct signature ..... 1 point
– condition to stop recursion ..... 1 point
– self-call (logic, parameters) ..... 2 points
– returned values ..... 1 point

```
Subalgorithm controlDigit(x):
   If x > 9 then
      s ← x MOD 10 + controlDigit(x DIV 10)
      If s < 10 then
         return s
      else
         return controlDigit(s)
      EndIf
   else
      return x
   EndIf
EndSubalgorithm
```

B.1.b. mathematical model for `controlDigit(x)` **..... 3 points**

$$controlDigit(x) = \begin{cases} x, & if\ x < 10 \\ controlDigit(x\ MOD\ 10\ +\ controlDigit(x\ DIV\ 10)), & else \end{cases}$$

B.1.c. subalgorithm for longest prefix ..... 7 points

Variant 1: the matrix is iterated over once and an apparition vector is built for the control digits of its elements. The digits of the given number (*nr*) are stored in an array. This array is iterated over, starting with the most significant digit and the apparition of the digits is checked in the previously built array. ..... **7 points**

Note: the same idea can be implemented using a frequency array for the digits.

```
const int MAXDIGITS = 10;
int prefixMaxDigits(int nr, int m, int n, int A[][MAX]){
   bool apparitions[MAXDIGITS];   int digits[MAX];
   int nrDigits;int i = 0;
   for (i = 0; i < MAXDIGITS; i++)                          // initialize frequency array
      apparitions[i] = 0;
   for (i = 0; i < m; i++){
      for (int j = 0; j < n; j++){
         apparitions[controlDigit(A[i][j])] = 1;            // control digit apparitions
      }
   }
   nrDigits = 0;                                               // number of nr's digits
   while (nr > 0){
```

```
      digits[nrDigits++] = nr % 10;                          // elements of nr's digit array
      nr /= 10;
   }
   i = nrDigits - 1;                                          // iterate the digit array
   while ((i >= 0) && (apparitions[digits[i]]))          // control digit = current digit of nr
      i--;
   return nrDigits - i - 1;                                  // length of longest prefix
}
```

Variant 2: Keep (*nr*) number's digits in an array. Search each digit in the matrix (results in several iterations of the matrix). (**5 points**)

```
bool searchDigit(int digit, int m, int n, int A[][MAX]){
   for(int i = 0; i < m; i++){
      for(int j = 0; j < n; j++){
         if(digit == controlDigit(A[i][j]))// matrix of control digits contains the searched digit
            return true;
      }
   }
   return false;                                              // searched digit not found
}
int prefixMaxDigits_v2(int nr, int m, int n, int A[][MAX]){
   int digits[MAX];
   int nrDigits = 0;                                     // number of nr's digits
   while(nr > 0){
      digits[nrDigits++] = nr % 10;                          // elements of nr's digit array
      nr /= 10;
   }
   int i = nrDigits - 1;                                      // iterate over digit array
   int p = 0;
   while(i >= 0){
      if(searchDigit(digits[i], m, n, A)){                // search for the digits in matrix A
         p++;
         i--;
      }
      else
         return p;
   }
   return p;          // number of consecutive digits of nr found in the matrix (prefix length)
}
```

B. 2. Robo-garden .....15 points

if $\boldsymbol{n} = 5$, $\boldsymbol{d} = (1, 2, 1, 2, 1)$, $\boldsymbol{u} = (1, 3, 2, 1, 3)$, $\boldsymbol{t} = 32$, calculate the values for $\boldsymbol{q}$:

2 * 1 + 1 + 1 = 4, 2* 2 + 3 + 1 = 8, 2 * 1 + 2 + 1 = 5, 2 * 2 + 1 + 1 = 6, 2 * 1 + 3 + 1 = 6 => (4, 8, 5, 6, 6)

Build array $\boldsymbol{v}$ having $\boldsymbol{t} = 32$ zero values. We consider the value of $\boldsymbol{q}$ for each robot and increment in the array the corresponding value for the multiples of $\boldsymbol{q}$.

$\boldsymbol{q} = 4$

	1	2	3	**4**	5	6	7	**8**	9	10	11	**12**	13	14	15	**16**	17	18	19	**20**
v	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**

	21	22	23	**24**	25	26	27	**28**	29	30	31	**32**
v	0	0	0	**1**	0	0	0	**1**	0	0	0	**1**

$\boldsymbol{q} = 8$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	0	0	0	2	0	0	0	1	0	0	0	2	0	0	0	1

	21	22	23	24	25	26	27	28	29	30	31	32
v	0	0	0	2	0	0	0	1	0	0	0	2

$\boldsymbol{q} = 5$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	1	0	0	2	0	1	0	1	0	0	1	2	0	0	0	2

	21	22	23	24	25	26	27	28	29	30	31	32
v	0	0	0	2	1	0	0	1	0	1	0	2

$\boldsymbol{q} = 6$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	1	1	0	2	0	1	0	2	0	0	1	2	0	1	0	2

	21	22	23	24	25	26	27	28	29	30	31	32
v	0	0	0	3	1	0	0	1	0	2	0	2

q = 6

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
v	0	0	0	1	1	2	0	2	0	1	0	3	0	0	1	2	0	2	0	2

	21	22	23	24	25	26	27	28	29	30	31	32
v	0	0	0	4	1	0	0	1	0	3	0	2

Determine the maximum value in array ***v***. In our example, the maximum value is 4 (at moment 24), so we require 4 - 1 = 3 extra water taps.

B.2.a. at any given moment in time ***mt*** ($1 \leq mt \leq t$) those robots will meet at the tap for which the value of ***q*** (equal with the sum between the time required to go to and return from the flower bed, time required to wet the flowers and time required to refill the water tank) is a multiple of ***mt*** ..... 3 points
B.2.b. the minimum number of additional water taps is equal to the maximum value in array ***v*** - 1, where the values of array **v** hold, for every moment of time, how many robots meet at the water tap ..... 2 points
B.2.c. Develop the subalgorithm

- V1: use a frequency array for each robot's working times multiples ..... 10 points
  - c.1. observe input and output parameters ..... **2 points**
  - c.2. calculate working time (***q*** = 2 * movement + watering + refill) ..... **1 point**
  - c.3. process frequency array ..... **4 points**
    - c.3.1. initialize array ..... 1 point
    - c.3.2. update frequency ..... 3 points
      1. v1a or v1b: loop over every q^{th} element ..... 3 points
      2. v2a or v2b: loop over every element and check multiples of q ..... 2 points
  - c.4. calculate maximum number of water taps ..... 2 **points**
      3. together with or after increment ..... 1 point
  - c.5. determine number of additional water taps (max - 1) ..... **1 point**

```
int roboV1a(int n, int d[], int u[] int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[i] = 0;
    for (int j = 1; j <= n; j++){
         int q = d[j] * 2 + u[j] + 1;
         for (int i = q; i <= t; i = i + q){
                 aux[i]++;
                 if (max < aux[i])
                         max = aux[i];
         }       //for i
    }    //for j
    return max - 1;
}
```

```
int roboV1b(int n, int d[], int u[], int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[i] = 0;
    for (int j = 1; j <= n; j++){
         int q = d[j] * 2 + u[j] + 1;
         for (int i = q; i <= t; i = i + q){
                 aux[i]++;
         }       //for i
    }    //for j
    for (int i = 1; i <= t; i++){
         if (max < aux[i])
                 max = aux[i];
    }    //for i
    return max - 1;
}
```

```
int roboV2a(int n, int d[], int u[], int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[i] = 0;
    for (int j = 1; j <= n; j++){
         int q = d[j] * 2 + u[j] + 1;
         for (int i = q; i <= t; i = i + 1){
                 if (i % q == 0){
                         aux[i]++;
                         if (max < aux[i])
                                 max = aux[i];
                 }       //if (i % q)
         }       //for i
    }    //for j
    return max - 1;
}
```

```
int roboV2b(int n, int d[], int u[], int t){
    int aux[200000];
    int max = 1;
    for (int i = 1; i <= t; i++)
         aux[i] = 0;
    for (int j = 1; j <= n; j++){
         int q = d[j] * 2 + u[j] + 1;
         for (int i = q; i <= t; i = i + 1){
                 if (i % q == 0){
                         aux[i]++;
                 }       //if (i % q)
         }       //for i
    }    //for j
    for (int i = 1; i <= t; i++){
         if (max < aux[i])
                 max = aux[i];
    }    //for i
    return max - 1;
}
```

- V2: simulation ..... 8 points
  - c.1. observe input and output parameters ..... 2 points
  - c.2. calculate working time (***q*** = 2 * movement + watering + refill) ..... 1 point
  - c.3. loop for time ..... 0.5 points
  - c.4. loop for robots ..... 0.5 points
  - c.5. calculate number of required water taps for a given moment in time ..... 1 point
  - c.6 calculate maximum number of water taps ..... 2 points
  - c.7. calculate number of additional water taps ..... 1 point

```
int roboV3(int n, int d[], int u[], int t){
    int nrMinTaps = 1;
    int timeCrt = 1;
    while (timeCrt <= t){
         int nrCrtTaps = 0;
         for (int j = 1; j <= n; j++){
                 int q = 2 * d[j] + u[j] + 1;
                 if (timeCrt % q == 0) // if at current time the i-th robot is at
                 nrCrtTaps ++;         // the tap
         }
         if (nrCrtTaps > nrMinTaps)
                 nrMinTaps = nrCrtTaps;
         timeCrt ++;
    }
    return nrMinTaps - 1;
}
```