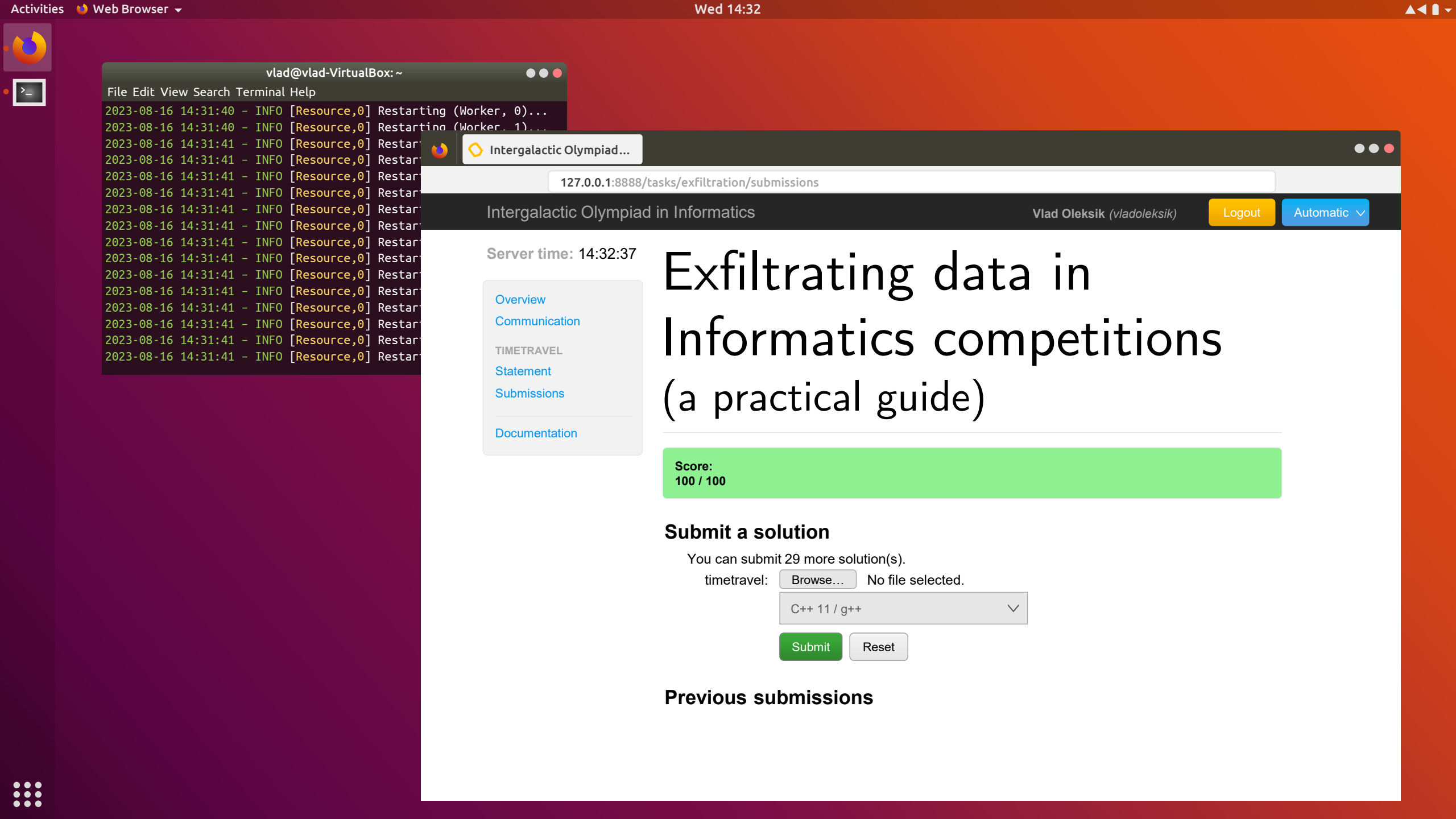

Activities
Web Browser
Wed 14:32
vlad@vlad-VirtualBox: ~
File Edit View Search Terminal Help
2023-08-16 14:31:40 - INFO [Resource,0] Restarting (Worker, 0)...
2023-08-16 14:31:40 - INFO [Resource,0] Restarting (Worker, 1)...
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
2023-08-16 14:31:41 - INFO [Resource,0] Restar
Intergalactic Olympiad...
127.0.0.1:8888/tasks/exfiltration/submissions
Intergalactic Olympiad in Informatics
Vlad Oleksik (vladoleksik)
Logout
Automatic
Server time: 14:32:37
Overview
Communication
TIMETRAVEL
Statement
Submissions
Documentation
Exfiltrating data in Informatics competitions
(a practical guide)
Score:
100 / 100
Submit a solution
You can submit 29 more solution(s).
timetravel:
Browse…
No file selected.
C++ 11 / g++
Submit
Reset
Previous submissions

The state of Informatics competitions

But what is a coding competition?

With the first such competition (and the most prestigious) ever being the International Olympiad in Informatics, what is known in the English-speaking world as a ‘coding competition’ brings together talented minds to solve **algorithmic problems** in an extremely competitive environment.

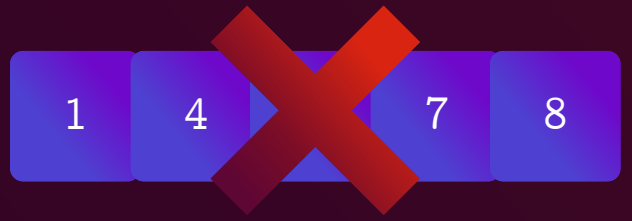
Algorithmic problems

Problems like those from the IOI don't ask for a single response – a number, an essay, or even a general proof – they ask you for a *recipe*.

Input: 

any list of numbers

Statement: **Sort** the numbers.

Desired output: 

⟵ This answer does **not** apply for every list of numbers!

Desired solution:

Put the numbers given in any order on the table.
Now, for every number, from the first to the last, search through all the numbers found after it and find their minimum. Swap the current number with that minimum and do the same for all the remaining numbers.

What a solution is

Put the numbers given in any order on the table, in a list.

Now, for every number, from the first to the last:

Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.

```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```

What a solution is

Put the numbers given in any order on the table, in a list.

Now, for every number, from the first to the last:

Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.

```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```

What a solution is

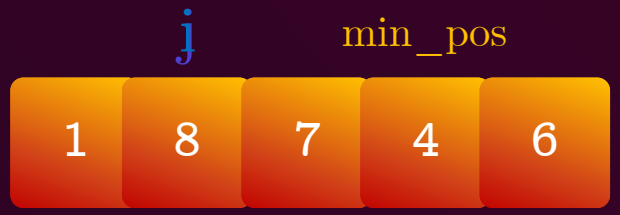
Put the numbers given in any order on the table, in a list.

Now, for every number, from the first to the last:

Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.




```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```

What a solution is

Put the numbers given in any order on the table, in a list.

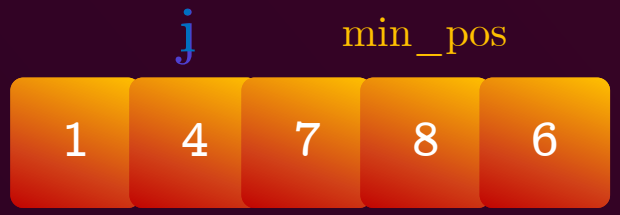
Now, for every number, from the first to the last:

Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.

```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```



What a solution is

Put the numbers given in any order on the table, in a list.

Now, for every number, from the first to the last:

Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.


```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```

What a solution is

Put the numbers given in any order on the table, in a list.

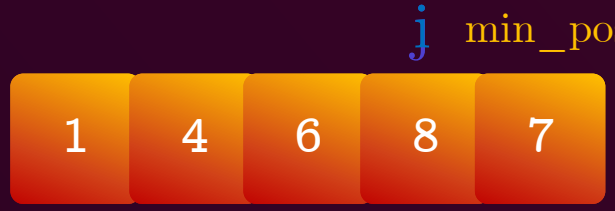
Now, for every number, from the first to the last:

Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.

```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```



What a solution is

Put the numbers given in any order on the table, in a list.

Now, for every number, from the first to the last:

Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.


```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```

What a solution is

Put the numbers given in any order on the table, in a list.

Now, for every number, from the first to the last:

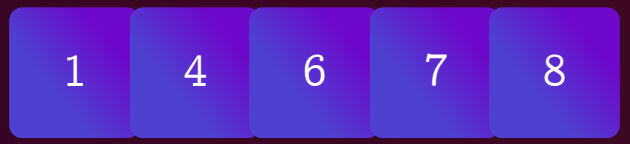
Search through every other number found after it

and find their minimum. (If this number is less that the smallest we had before, it is the new minimum).

Swap the current number with that minimum and do the same for all the remaining numbers.

```
int n, list[ ];
int i, j, min_pos;

for (i = 0; i < n - 1; i++)
{
    min_pos = i;
    for (j = i + 1; j < n; j++)
    {
        if (list[j] < list[min_pos])
            min_pos = j;
    }

    if (min_pos != i)
        swap(list[min_pos], list[i]);
}
```

This solution **does** apply for every list of numbers!



Scoring – test cases

Since the 1970s, some national feeder competitions used computers to grade each solution provided by a participant.

Still, it is difficult even for a human or an AI to prove that *any* algorithm either does or does not provide the correct answer for *every* acceptable input given. When using a simple program to grade the solutions, it is almost impossible to make it decide how a certain algorithm would behave just by looking at it.

A different approach is needed.

How to judge a solution

Thus, a convention has been adopted for evaluating an algorithm by actually running it.

Convention: Test the algorithm by running it for a large number of random, secret examples. If it produces the desired output every time, it is considered correct*.

Test case	Input	Algorithm output	Correct output	Points
1.	2 3 1		1 2 3	**10**/10
2.	6 4 2 9	2 4 6 9	2 4 6 9	**10**/10
3.	10 9 8 7	7 8 9 10	7 8 9 10	**10**/10
...				
n.	7 8 1 4 6	1 4 6 7 8	1 4 6 7 8	**10**/10

Selection sort
Put the numbers given in any order on the table. Now, for every number, from the first to the last, search through all the numbers found after it and find their minimum. Swap the current number with that minimum and do the same for all the remaining numbers.

How to judge a solution

If, instead, the algorithm does not produce the desired output for some inputs, we can conclude it is wrong and score it accordingly.

Convention: Test the algorithm by running it for a large number of random, secret examples. If it produces the desired output every time, it is considered correct*.

A wrong sorting algorithm
This algorithm does not perform well for values larger than 9 in the input.

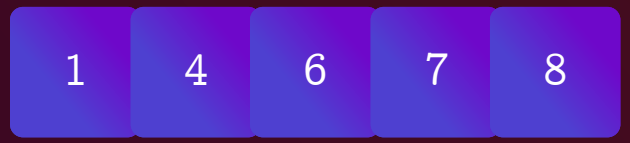
Test case	Input	Algorithm output	Correct output	Points
1.	2 3 1		1 2 3	**10**/10 ✓
2.	6 4 2 9	2 4 6	2 4 6 9	**0**/10 ✗
3.	10 9 8 7	7 8	7 8 9 10	**0**/10 ✗
...				
n.	7 8 1 4 6	1 4 6 7 8	1 4 6 7 8	**10**/10 ✓

How to judge a solution *the catch:

Selection sort
Put the numbers given in any order on the table. Now, for every number, from the first to the last, search through all the numbers found after it and find their minimum. Swap the current number with that minimum and do the same for all the remaining numbers.

This solution usually requires around n^2 operations.

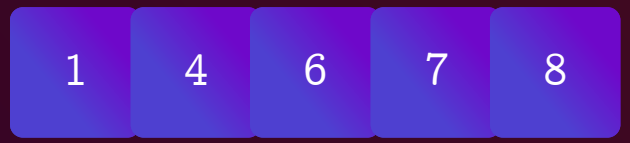
$\longrightarrow$ $O(n^2)$



This one may require only around $n \cdot \log(n)$ operations, no matter the numbers.

Merge sort
Split the list of numbers into two batches. Split every subsequent batch into two until they only have one number. Now, when merging similar sized batches together, always take the smallest number from each batch in order.

$\longrightarrow$ $O(n \cdot \log(n))$



How to judge a solution *the catch:

Selection sort
Put the numbers given in any order on the table. Now, for every number, from the first to the last, search through all the numbers found after it and find their minimum. Swap the current number with that minimum and do the same for all the remaining numbers.

$O(n^2)$

$<$

Merge sort
Split the list of numbers into two batches. Split every subsequent batch into two until they only have one number. Now, when merging similar sized batches together, always take the smallest number from each batch in order.

$O(n \cdot \log(n))$

Upon analysis, it is immediately obvious that the second algorithm, 'Merge sort', is better as it does what it is designed to do *faster*.

Another metric by which we can compare two algorithms is how much memory they use. In the example above, 'Merge sort" uses a bit more memory than 'Selection sort', but, in this case, the time saved is much more important.

These nuances are crucial to make competitive programming *competitive*.

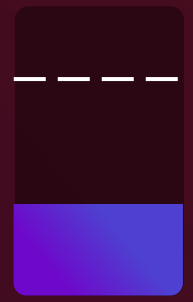
How to judge a solution

Thus, a way to take efficiency into account is needed. As such, every time the program is run, it has a *time* and *memory* budget. If it overspends, it is terminated and that case is marked as failing.

Selection sort
Put the numbers given in any order on the table. Now, for every number, from the first to the last, search through all the numbers found after it and find their minimum. Swap the current number with that minimum and do the same for all the remaining numbers.

$\longrightarrow$

$O(n^2)$



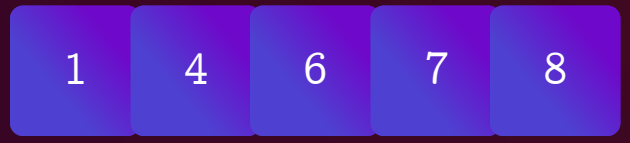
Memory

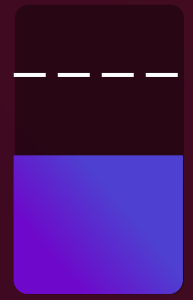
Time

Merge sort
Split the list of numbers into two batches. Split every subsequent batch into two until they only have one number. Now, when merging similar sized batches together, always take the smallest number from each batch in order.

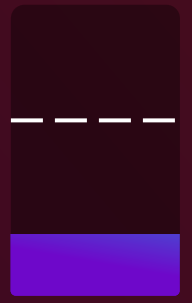
$\longrightarrow$

$O(n \cdot \log(n))$





Memory



Time

How to judge a solution

Convention: Test the algorithm by running it for a large number of random, secret examples. It receives points for giving the correct output, within the specified time and memory span.

Time limit: 0.1 s **Memory limit:** 64 MB

Test case	Input	Algorithm output	Correct output	Time	Memory	Points
1.	2 3 1	1 2 3	1 2 3	0.000s	2KB	**10**/10
2.	6 4 2 9	2 4 6 9	2 4 6 9	0.002s	2KB	**10**/10
3.	10 9 8 7	7 8 9 10	7 8 9 10	0.010s	3KB	**10**/10
...						
n.	7 8 1 4 6	N/A	1 4 6 7 8	0.100s	4KB	**0**/10

Selection sort
Put the numbers given in any order on the table. Now, for every number, from the first to the last, search through all the numbers found after it and find their minimum. Swap the current number with that minimum and do the same for all the remaining numbers.

How to judge a solution

Convention: Test the algorithm by running it for a large number of random, secret examples. It receives points for giving the correct output, within the specified time and memory span.

Time limit: 0.1 s **Memory limit:** 64 MB

Test case	Input	Algorithm output	Correct output	Time	Memory	Points
1.	2 3 1		1 2 3	0.000s	4KB	**10**/10
2.	6 4 2 9	2 4 6 9	2 4 6 9	0.001s	4KB	**10**/10
3.	10 9 8 7	7 8 9 10	7 8 9 10	0.001s	6KB	**10**/10
...						
n.	7 8 1 4 6	1 4 6 7 8	1 4 6 7 8	0.002s	8KB	**10**/10

Merge sort
Split the list of numbers into two batches. Split every subsequent batch into two until they only have one number. Now, when merging similar sized batches together, always take the smallest number from each batch in order.

Scoring – before interactive platforms

For quite a long time, the standard procedure for running coding competitions worked like this:

1. All contestants received the tasks and designed their programs on a specially monitored PC.
2. After the contest time was over, each contestant would submit the final version of their programs.
3. Each program would be run on the same computer on a number of very secret examples, each run being time- and memory-conditioned. Each run would be graded accordingly, the competitors being ranked by the total number of points their solutions obtained.

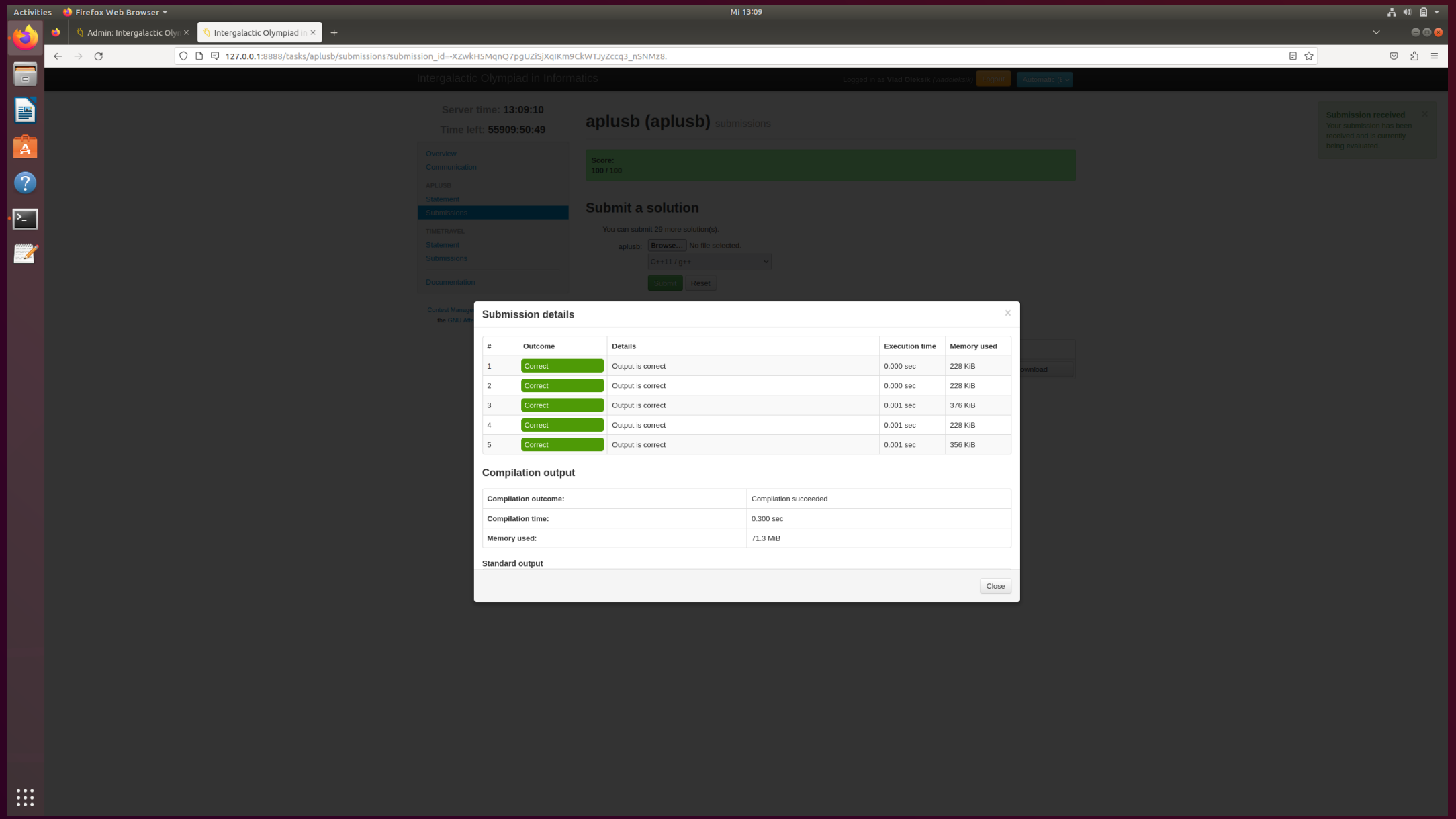
Scoring – interactive platforms

Since 2003's Google Code Jam competition, an interactive platform has been introduced to manage the contest. Their model has since been used in most coding competitions worldwide, including the IOI.

Instead of submitting the code at the end of the contest, the participant now (usually) has 30 attempts, during the contest, to upload his/her solutions. After every submission, he instantly has access to the score of the last attempt, along with a breakdown of each test case, the outcome, as well as the time and memory used. The only thing that is still kept secret is the actual test data.

After that, he can upload another program he designed, the final score being (usually) the maximum among his submission scores.

Interactive platforms



CMS – the most widely used platform

Scoring – interactive platforms

Thus, each contestant receives more feedback that he/she has the opportunity to act upon to improve the performance of the algorithms.

This allows for more dynamic competition and implies harder challenges.

Test #	Result	Time	Memory	Points
1.	Wrong answer.	0.000s	2KB	**0**/10
2.	Answer is correct.	0.002s	2KB	**10**/10
3.	Memory limit exceeded.	0.010s	64MB	**0**/10
4.	Time limit exceeded.	0.100s	2KB	**0**/10
5.	Time limit exceeded.	0.100s	4KB	**0**/10

30 attempts max.

Evaluation results – short dictionary

- **Output is correct.** – AC/OK:
  The answer is correct for the respective test case.
- **Output isn't correct.** – WA:
  The answer is wrong for the current test case.
- **Execution timed out.** – TLE:
  The time limit was exceeded for this test case.
- **Execution killed (could be triggered by violating memory limits)** – MAV/MLE:
  The program tried to access memory outside its designated region. *May (MLE) or may not (other kind of MAV) mean that the memory limit itself was exceeded.*
- **Execution failed because the return code was nonzero.** – RTE:
  The program returned an error as something it tried to do was not permitted (i.e., divide by zero).

Performing a data exfiltration scheme

What is exfiltration?

Exfiltration is the process of stealthily (and sometimes forcefully) extracting some data from a program – in this case, making CMS give away some other data that it wouldn't share with us otherwise.

What to exfiltrate?

The most important piece of information we could hope to obtain is the content of all the secret input data. For each test case, there is a file containing this input that is fed into the algorithm when the platform runs it.

Why exfiltrate the test inputs?

The importance of the test input values, as well as the reason why they are kept secret, is that, by knowing all the examples the program will be tested on, a competitor could easily 'fake' the algorithm by simply writing such a program:

If the input data is: 

then just output: 1 2 3

If the input data is: 

then just output: 2 4 6 9

If the input data is: 

then just output: 7 8 9 10

If the input data is: 

then just output: 1 4 6 7 8

If the input data is something else: ¯_(ツ)_/¯ (since we all know it won't happen ☺)

A computer wouldn't know the program *only* outputs the correct answers for *a few, select cases*, and would consider it correct.

How to exfiltrate the inputs?

Let's first consider a practical example. You are at the Intergalactic Olympiad in Informatics and you receive a problem like this:

TimeTravel

Input file	`timetravel.in`
Output file	`timetravel.out`

Alice is a student who doesn't really like Maths. For this reason, her uncle, Michel Leonard, gave her a special, magical present – a ***quadratic sieve***. It can decompose any number into its prime factors with an incredible speed.

Unfortunately, though, Alice got used to not doing Maths exercises with pen on paper, but exclusively using her magic machine. Bob, her teacher, is quite upset with this and decided to give her a 3-second blitz test every day. For these tests, Bob gives her a number n that is known to be a product of two primes p and q, and Alice has to quickly find the two prime factors and write them on the paper in increasing order.

To Alice's despair, she isn't that much into divisions, and her teacher does not allow her to use her quadratic sieve at school. What is more, the 3 seconds wouldn't suffice even if she were to use the sieve to find the requested prime factors. How could Alice provide the correct answers to her tests?

Input Data

For each test case, the input file contains the number n.

Output Data

For each test case, the first line of the output file shall contain the prime numbers p and q, with $p \leq q$ and $n = p \cdot q$, separated by a single space character.

Constraints

- It is guaranteed that there is such a pair of prime factors for each n – thus, each test case has exactly one correct solution
- $10^6 \leq n \leq 10^{59}$

#	Points	Constraints	Remarks
1	10	$n \leq 10^7$	$\log_2 n \leq 23$
2	10	$n \leq 10^{18}$	$\log_2 n \leq 58$
3	80	Initial constraints apply	$\log_2 n \leq 196$

Example

timetravel.in	timetravel.out	Explanation
51	3 17	51 = 3 * 17

The problem statement

TL;DR:

Given: n

With: $n < 2^{196}$

Find: p q

Such that: p and q are **primes**

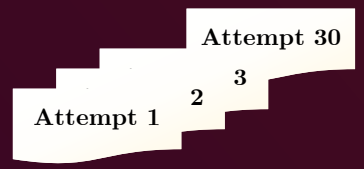
and $p \cdot q = n$

within 3 s, 1024 MB

10 tests

Test #	Input
1.	n_1
2.	n_2
3.	n_3
4.	n_4
5.	n_5
6.	n_6
7.	n_7
8.	n_8
9.	n_9
10.	n_{10}

30 attempts



*

The catch

A number around 2^{196} looks like this:

28285555926474195574107447017275555828319920146830138017429

No known algorithm could factor it in under 3 seconds on the machine that performs the evaluation. Plainly, it is unanimously established that there is *no way* to *honestly* get 100 points on this challenge.

The *actual* problem

The idea

If we could obtain all these big numbers, we *could* factor them in well under a minute each on some machine of ours using the *Quadratic sieve* algorithm.

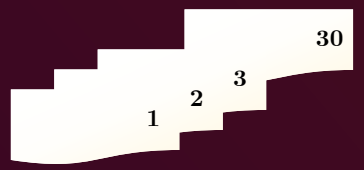
2828555592647419557410744701727555582831992014683013801742 9

= 311141115191216151315181917 19 * 909090909090909090909090909091

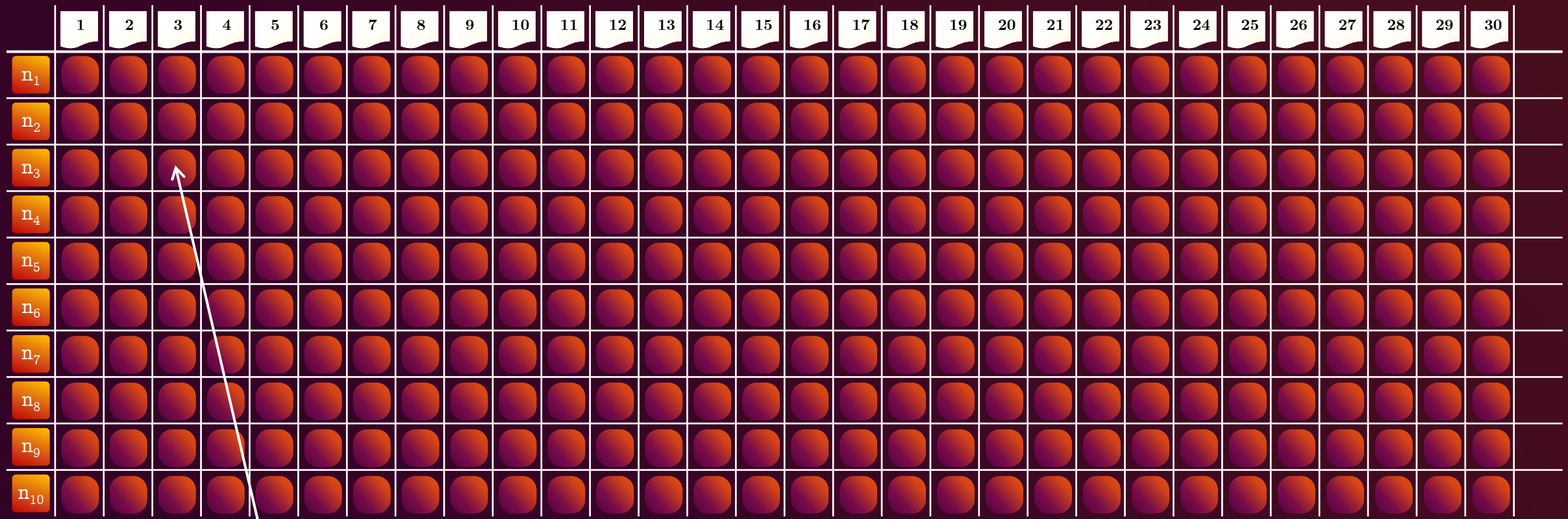
Then, we could write a simple program to just "deliver" the right answers to the known "questions" on time.

Test #	Input
1.	n_1
2.	n_2
3.	n_3
4.	n_4
5.	n_5
6.	n_6
7.	n_7
8.	n_8
9.	n_9
10.	n_{10}

10 tests × 30 attempts



The *actual* problem

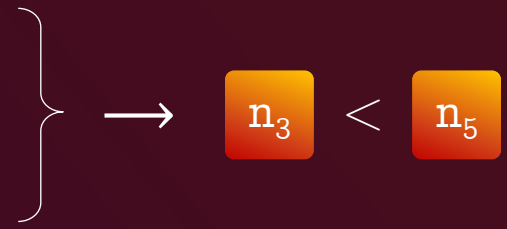


3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

} *information*

Information

3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts
5.	WA | Output isn't correct.	0.256 s	128 MiB	0/10 pts



The statement that test outcomes contain information in and of themselves seems quite self-obvious.

Just by looking at these outcomes, you may deduce that the workload for processing example no. 5 is heavier than the one needed to produce an output for case 3.

Thus, it should occur to anyone in a normal competition that, for case 5, they are dealing with (a) higher number(s). This assumption we are supported to make is, means, by definition, that we obtain *information* from the (so-called *blind*) test reports. But how to quantify and handle it?

Information

1 bit
(2 possibilities)

2.58496 bits
(6 possibilities)

$$I = \log_2(possibilities)$$

A number less than 2^{196}

$$28285555926474195574107447017275555828319920146830138017429$$

$$= \begin{matrix} 01001000000110010010111110111010111110000010011000 \\ 00010011000100001110001111101101101000101110110 11 \\ 10000010010101001000001001011111000000010100000 11 \\ 01110000011010001110111011011000000000110100101 01 \end{matrix}$$

196 bits

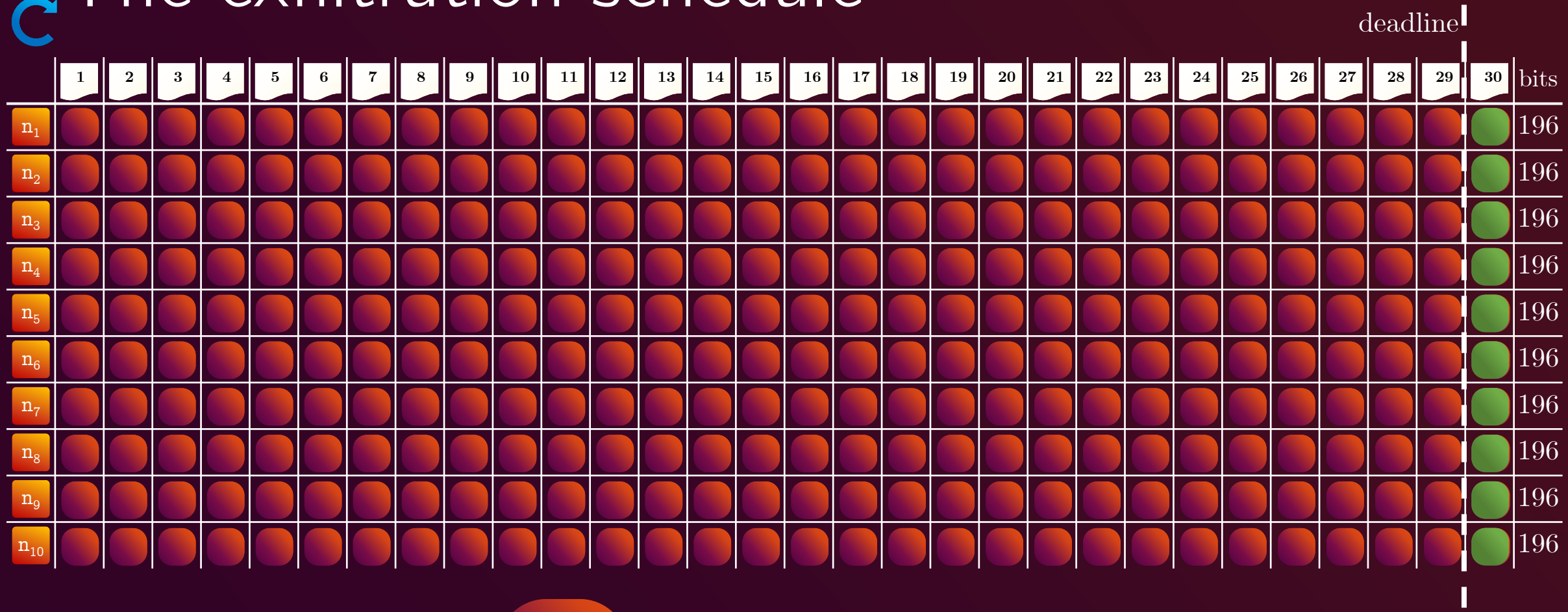
This is our target. We need to get precisely 196 bits of information about each number.

3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

? bits

How many *useful* bits a test feedback holds depends on how we are trying to interpret it.

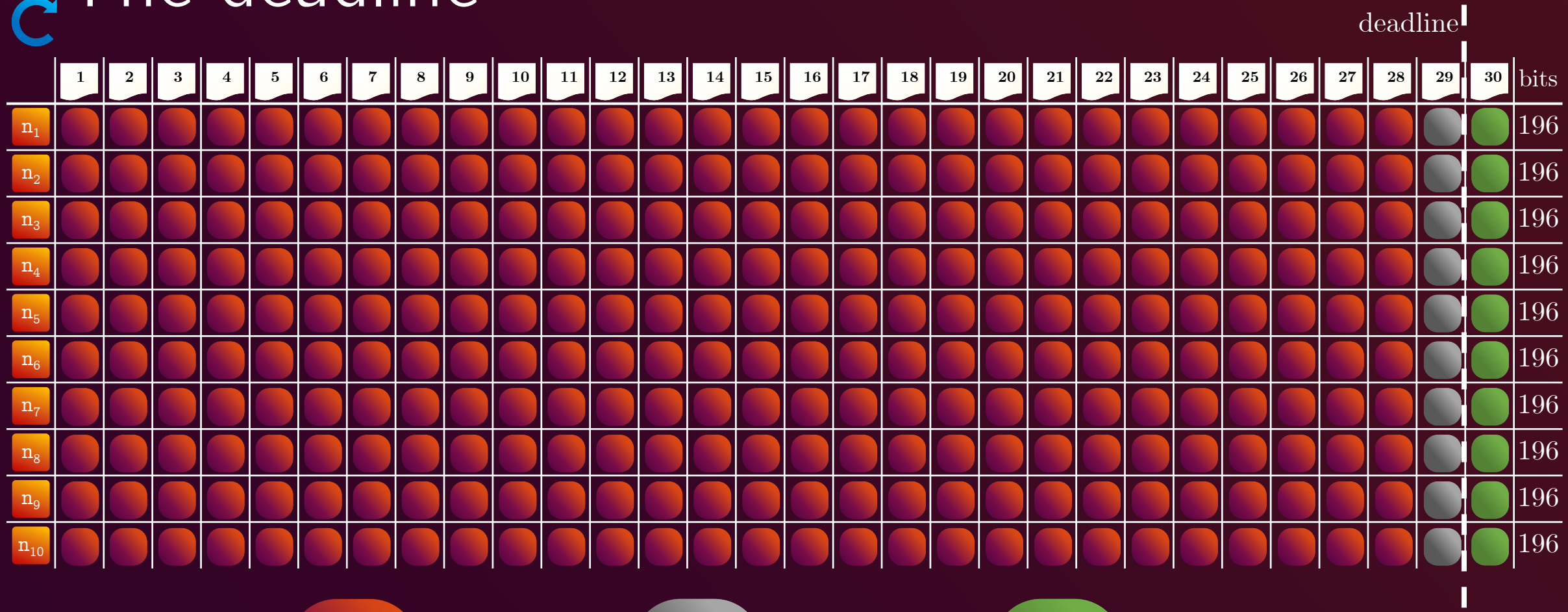
The exfiltration schedule



? bits

$$\frac{196 \text{ bits}}{28 \text{ attempts}} = 7.00 \text{ bits/attempt}$$

The deadline



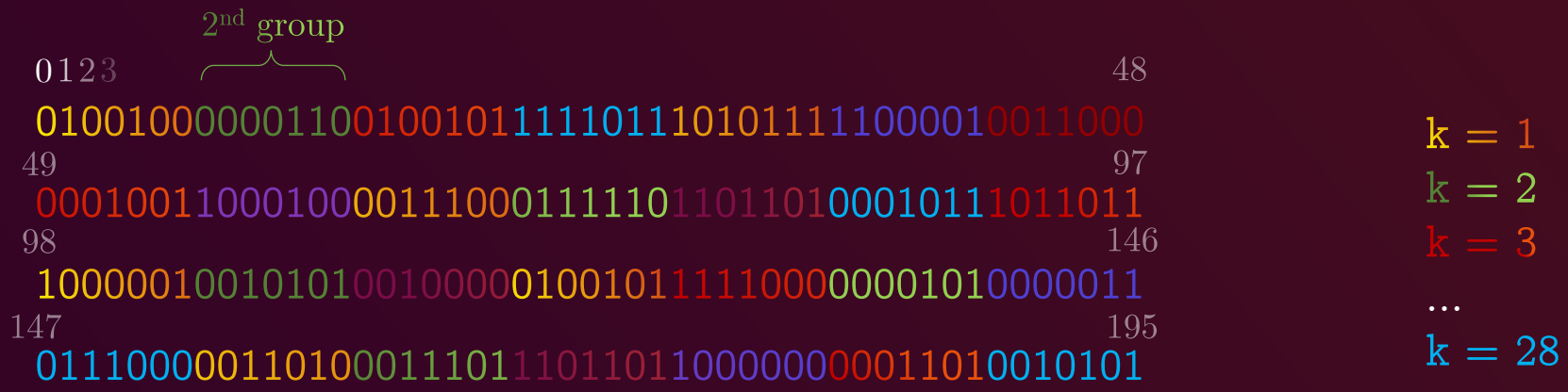
Knowing our bits

Now that we know that we plan to exfiltrate 7 bits every time the platform runs our program, we should establish what those bits represent.

```
01001000000110010010111110111010111110000100110000
00010011000100001110001111101101101000101110110111
10000010010101001000001001011111000000010100000011
01110000011010001110111011011000000000110100101010
```

Knowing our bits

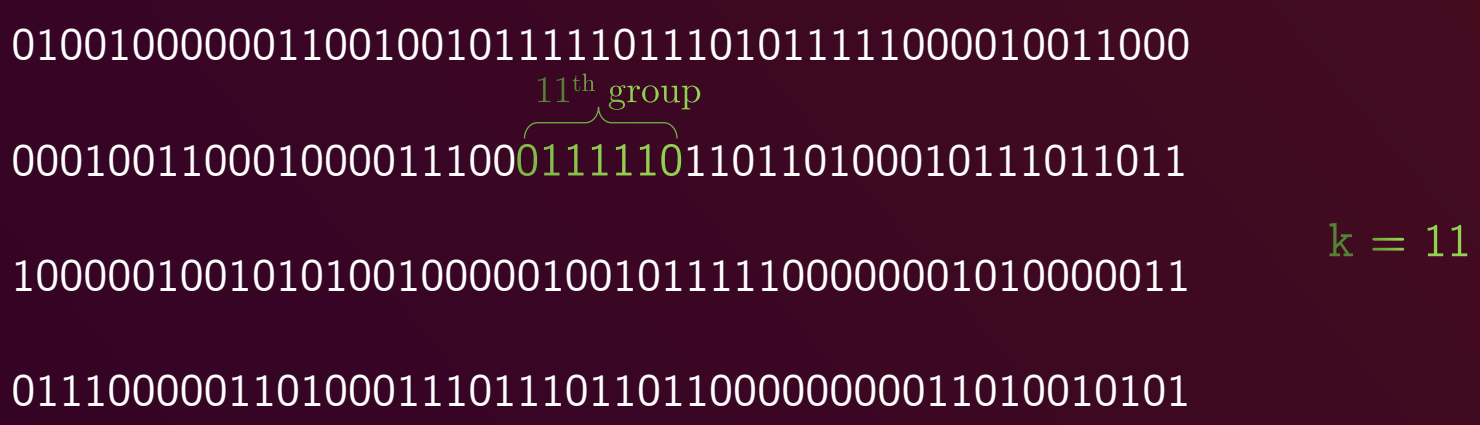
Now that we know that we plan to exfiltrate 7 bits every time the platform runs our program, we should establish what those bits represent.



The bit at position x belongs to the group $k = (x \ / \ 7) + 1$. Conversely, group number k spans from bit $7 \times (k - 1)$ to bit number $7 \times (k - 1) + 6$.

Knowing our bits

As such, the program from attempt number k will search for the bits on the positions from $7 \times (k - 1)$ to $7 \times (k - 1) + 6$ and try to exfiltrate those bits.



To get those bits, we shall divide the number by 2^{28-k} and get the remainder modulo $2^7 (= 128)$.

Fitting 7 bits

For each attempt k, we know which 7 bits to exfiltrate. We just need to find out if each outcome can provide us with at least 7 useful bits, and where to "squeeze" them.

0111110

Attempt k

3.	MAV | Execution failed because the return code was nonzero.	0.008 s	121 MiB	0/10 pts

Fitting 7 bits

Indicates the test case. Holds no other useful information.

We cannot produce an OK answer, but we have 4 distinct outcomes that we can control:

- WA
- RTE
- MAV
- TLE

As there are four possibilities, this field holds $\log_2 4 =$ **2 bits**.

Since the TLE verdict already implies 3.0 s being spent, this field can not reliably store any useful information on its own. We could sacrifice a bit from the verdict field to try gaining at least a bit from the time field, but this is neither more reliable nor any more convenient.

Any amount of memory can theoretically be allocated between 0B and 1024MiB. However, precision is found to be of about 4MiB and only for some smaller intervals. We have to devise a scheme for efficiently transmitting 5 bits.

Will pretty much always be 0/10. Holds no other useful information.

The verdict bits

0111110 ← We take the 2 least significant bits and produce an outcome accordingly.

MAV	Execution failed because the return code was nonzero.	**2 bits**

00 – WA:
Produced by a `return 0;` ending statement in the program.
This makes the program stop running as it reached the end of the instructions.

01 – RTE:
Produced by a `return 7;` ending statement in the program.
This makes the program stop running as if it encountered an error.

10 – MAV:
Produced by an `int *p=NULL;(*p)++;` statement in the program.
This accesses a forbidden memory address and emulates running out of memory without any impact to the actual measured memory usage.

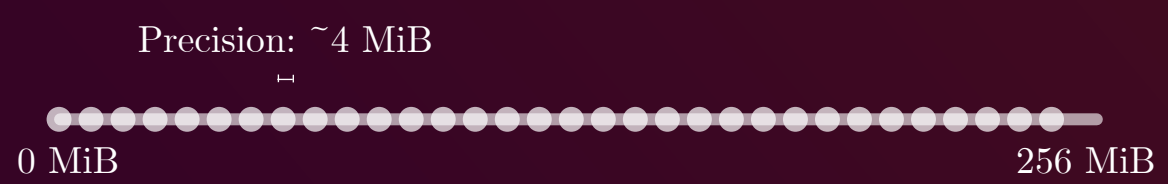
11 – TLE:
Produced by a `while(true);` statement in the program.
This makes the program run an infinite loop and run out of time.

The memory bits

121 MiB **5 bits** 01111110

We take the 5 most significant bits and produce an outcome accordingly.

TL;DR:



For extra reliability, we've taken 8MiB as our unit. Thus, we have $\frac{256\ \text{MiB}}{8\ \text{MiB/interval}} = 32\ \text{intevals}$ of 8MiB width to represent our bits on.

We'll take the most significant 5 bytes and convert them to a value from 0 to 31 in base 10. We'll then multiply this number by 8 MiB and this is the size we are going to allocate.

$$01111_{(2)} = 15_{(10)} \quad 15 \times 8\text{MiB} = 120\ \text{MiB}$$

```
vector <bool> v; v.resize(15*8*1048576*8);
```

1 bit; 1MiB; 1B; 8MiB

Tech zone:

1) Statements like `malloc(size);` or `v.reserve(size);` do not work when simply appended to the source as the compiler optimises them, essentially removing their effect as they only intend allocate the memory for a specific purpose and never use it.
We instead use `v.resize(size);`, which also populates the vector.

2) Since the environment used for testing had little extra memory for running the OS, the actual safe and reliably reported memory limit was 256MiB. It does impact how many bits we can exfiltrate, but the exfiltration as we conceived it is nevertheless possible.

The k^{th} exfiltration/attempt program

1. Start and read the number from the file.
2. Convert the number to base 2 and get the k^{th} batch of bits.
3. Get the 5 most significant bits and reserve memory corresponding to their value.
4. Get the 2 least significant bits from the batch and exit under the corresponding circumstances.

```
#include <iostream>
#include <fstream>
#include <vector>
#include <string>

using namespace std;

ifstream fin("timetravel.in");

string s;

vector <bool> v;

int main()
{
    short int k=1,a,b,i;
    unsigned int x;
    bool r;
    fin >> s;
    a=(k-1)*7;
    b=(k-1)*7+6;
    x=0;
    i=0;
    while(s.compare("0") != 0)
    {
        r=(s.back()-'0')%2;
        if((195-i)>=a && (195-i)<=b)
        {
            x+=(r<<i-a);
        }
        i++;
        s=longDivision(s,2);
    }
    x&=0b1111111;

    v.resize((x>>2)*8*1048576*8);

    if(x&0b11 == 0)
    {
        return 0;
    }
    if(int(x&0b11) == 1)
    {
        return 7;
    }
    if(int(x&0b11) == 2)
    {
        int * p=NULL;
        (*p)++;
        return 0;
    }
    if(int(x&0b11) == 3)
    {
        while(true);
        return 0;
    }
}
```

Reading the bits from the outcome

3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

When provided with an output, one has to divide the memory used by 8 MiB, round the value to the nearest integer, and convert it into base 2. The rounding is needed because slightly more memory might be needed by the program. The resulting five bits are the first five of the seven exfiltrated bits.

Reading the bits from the outcome

3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

When provided with an output, one has to divide the memory used by 8 MiB, round the value to the nearest integer, and convert it into base 2. The rounding is needed because slightly more memory might be needed by the program. The resulting five bits are the first five of the seven exfiltrated bits.

$$96.1 \text{ MiB} \ / \ 8 \text{ MiB} = 12.0125 \approx 12 = 01100_{(2)}$$

Reading the bits from the outcome

3.	**WA** | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

When provided with an output, one has to divide the memory used by 8 MiB, round the value to the nearest integer, and convert it into base 2. The rounding is needed because slightly more memory might be needed by the program. The resulting five bits are the first five of the seven exfiltrated bits.

$$96.1 \text{ MiB} \ / \ 8 \text{ MiB} = 12.0125 \approx 12 = 01100_{(2)}$$

Next, take the verdict and get the corresponding last bits from the table:

WA	00
RTE	01
MAV	10
TLE	11

01100__

Reading the bits from the outcome

3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

When provided with an output, one has to divide the memory used by 8 MiB, round the value to the nearest integer, and convert it into base 2. The rounding is needed because slightly more memory might be needed by the program. The resulting five bits are the first five of the seven exfiltrated bits.

$$96.1 \text{ MiB} \ / \ 8 \text{ MiB} = 12.0125 \approx 12 = 01100_{(2)}$$

Next, take the verdict and get the corresponding last bits from the table:

WA	00
RTE	01
MAV	10
TLE	11

01100__

Reading the bits from the outcome

3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

When provided with an output, one has to divide the memory used by 8 MiB, round the value to the nearest integer, and convert it into base 2. The rounding is needed because slightly more memory might be needed by the program. The resulting five bits are the first five of the seven exfiltrated bits.

$$96.1 \text{ MiB} \ / \ 8 \text{ MiB} = 12.0125 \approx 12 = 01100_{(2)}$$

Next, take the verdict and get the corresponding last bits from the table:

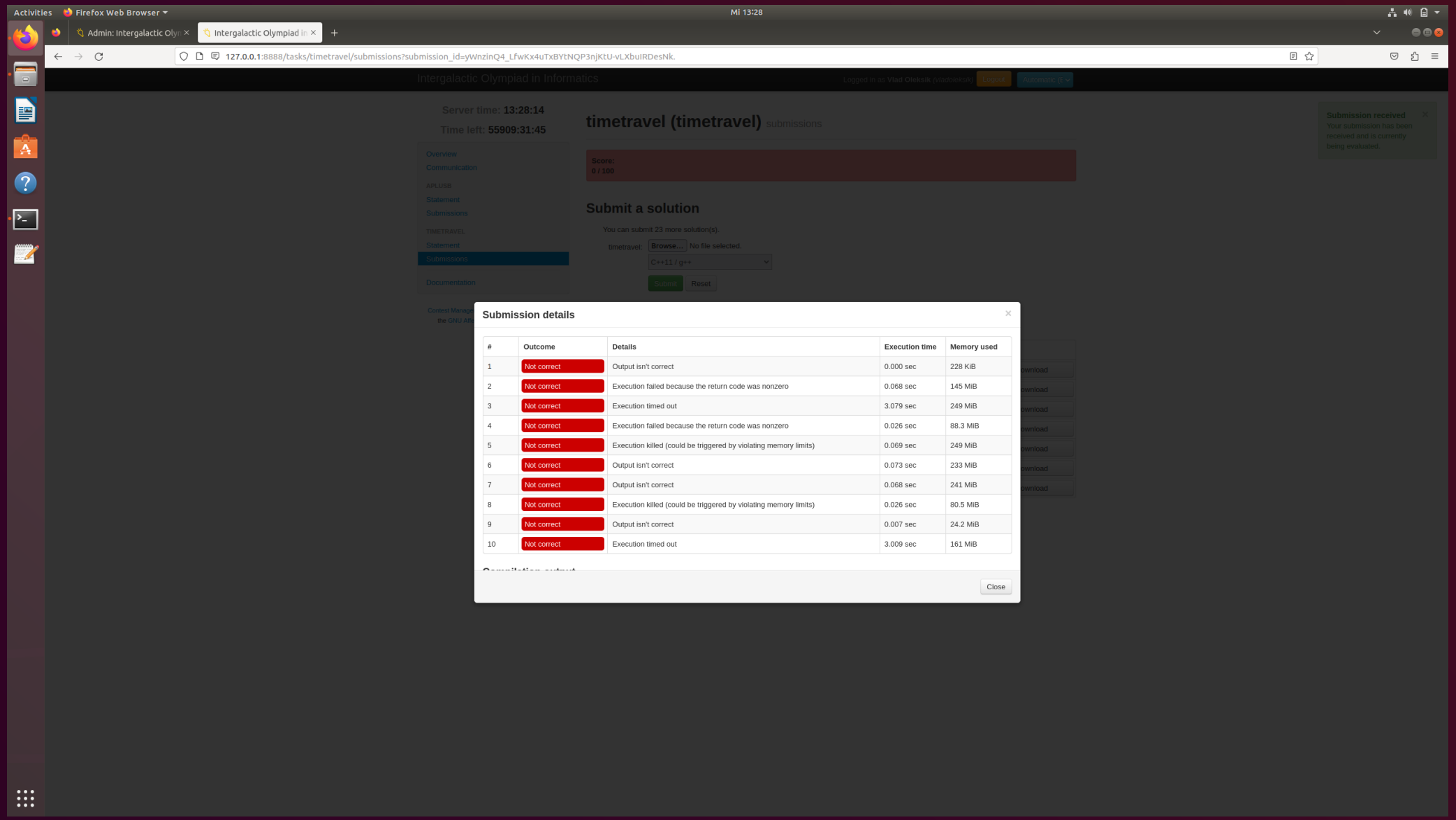
WA	00
RTE	01
MAV	10
TLE	11

0110000

Reading the bits from the outcome

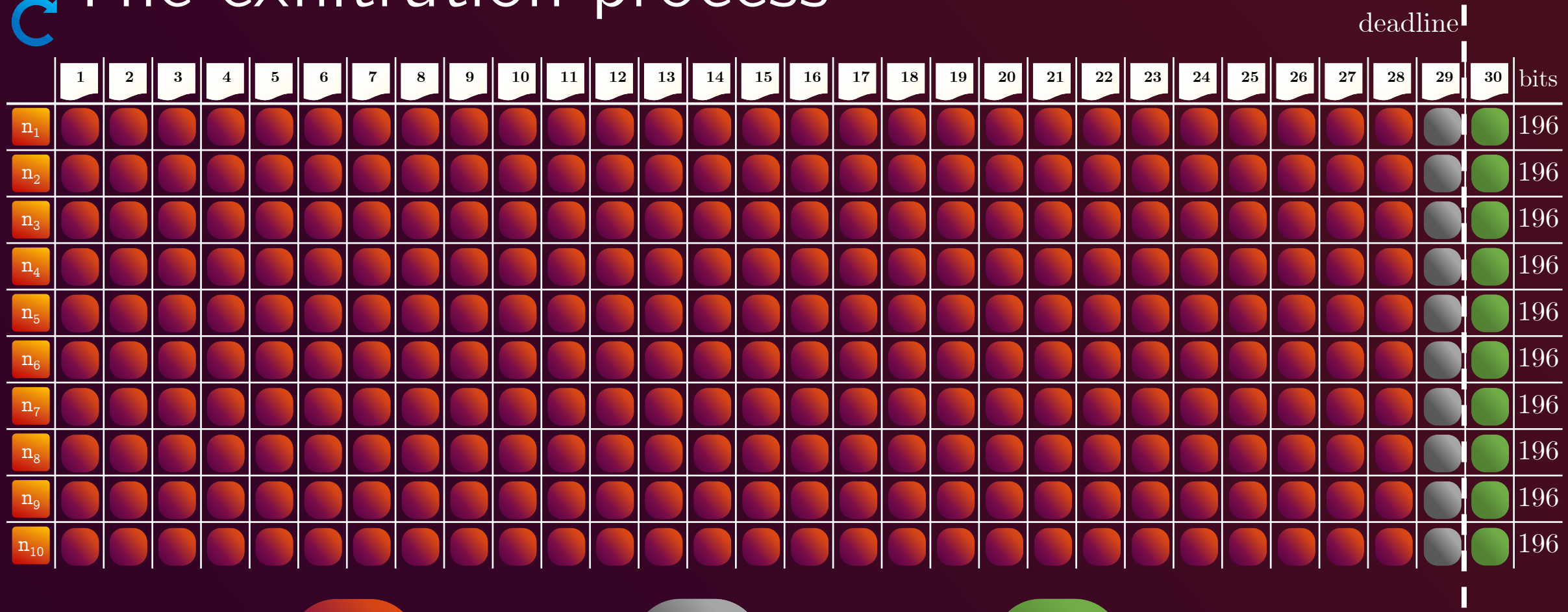
3.	WA | Output isn't correct.	0.048 s	96.1 MiB	0/10 pts

Getting the data

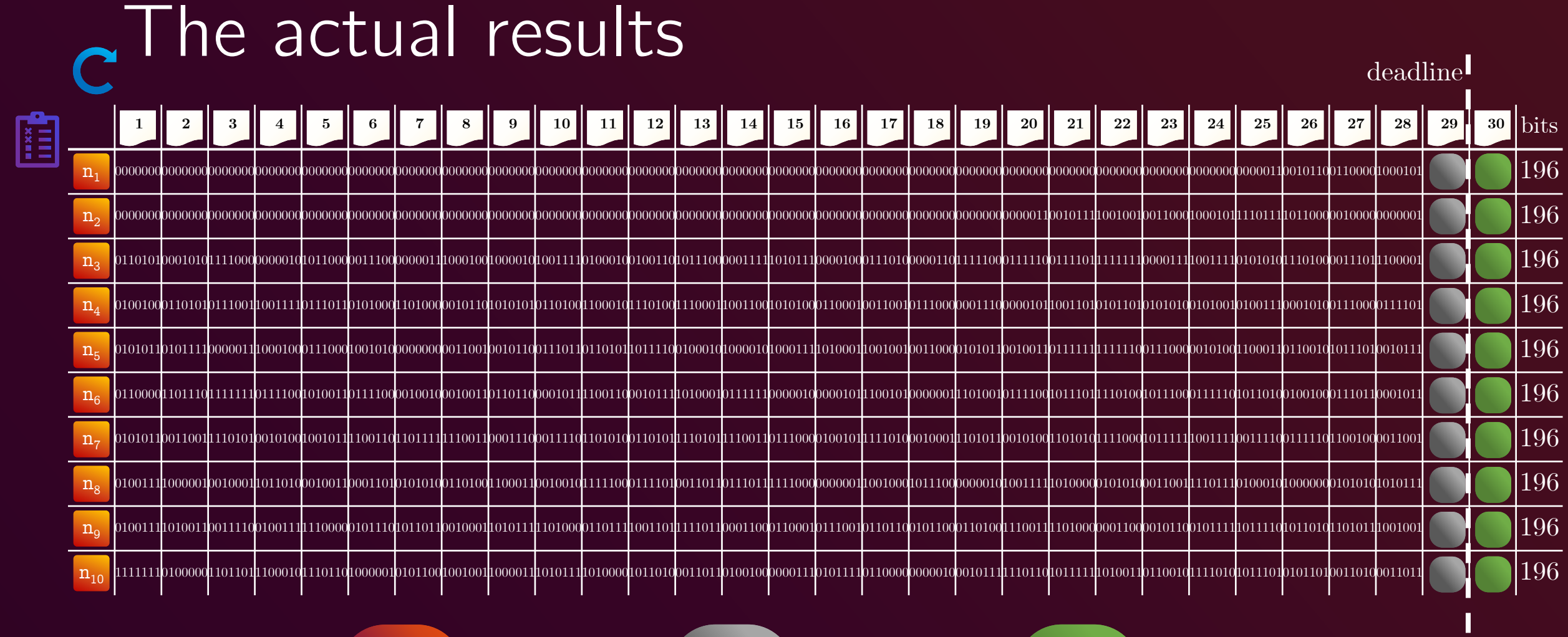


Obtaining the actual, practical exfiltration feedback in CMS – Exfiltrating batch no. 22

The exfiltration process



The actual results



7 bits unused scoring

The actual results

n_1 0001100101100110000100010 1

n_2 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000001100101111001001001100010001011110111101100000100000000001

n_3 0110101000101011110000000001010110000001110000000111000100100001010011110100010010011010111000001111101011100001000111010000011011111000111110011110111111110000111100111101010101110100001110111000 01

n_4 0100100011010101110011001111011101101010001101000000101101010101011010011000101110100111000110011001010100011000100110010111000000011100000101100110101011010101010010100101001110001010011100000111101

n_5 0101011010111100000011100010001110001001010000000000110010010110011101101101011011110010001010000101000111101000110010010011000010101100100110111111111111001110000010100110001101100101011101001011 1

n_6 0110000110111011111111011110010100110111100001001000100110110110000101111001100010111101000101111110000010000010111001010000000111010010111100101110111101001011100011111010110100100100011101100010 11

n_7 0101011001100111101010010100100101111001101101111111001100011100011110110101001101011110101111001101110000100101111101000100011101011001010011010101111000101111110011110011110011111011001000011001

n_8 0100111100000100100011011010001001100011010101010011010011000110010010111110001111010011011011101111111000000000110010001011100000000101001111101000000101010001100111101110100010100000001010101011 1

n_9 0100111101001100111100100111111000000101110101101100100011010111110100001101111001101111101100011000110001011100101101100101100011010011100111101000000011000010110010111110111101011010110101110010 01

n_{10} 1111111010000011011011100010111011010000001010110010010011000011101011110100001011010001101101001000000111010111101100000000010001011111101101011111101001101100101111010101110101011010011010001101 1

The test values, revealed…

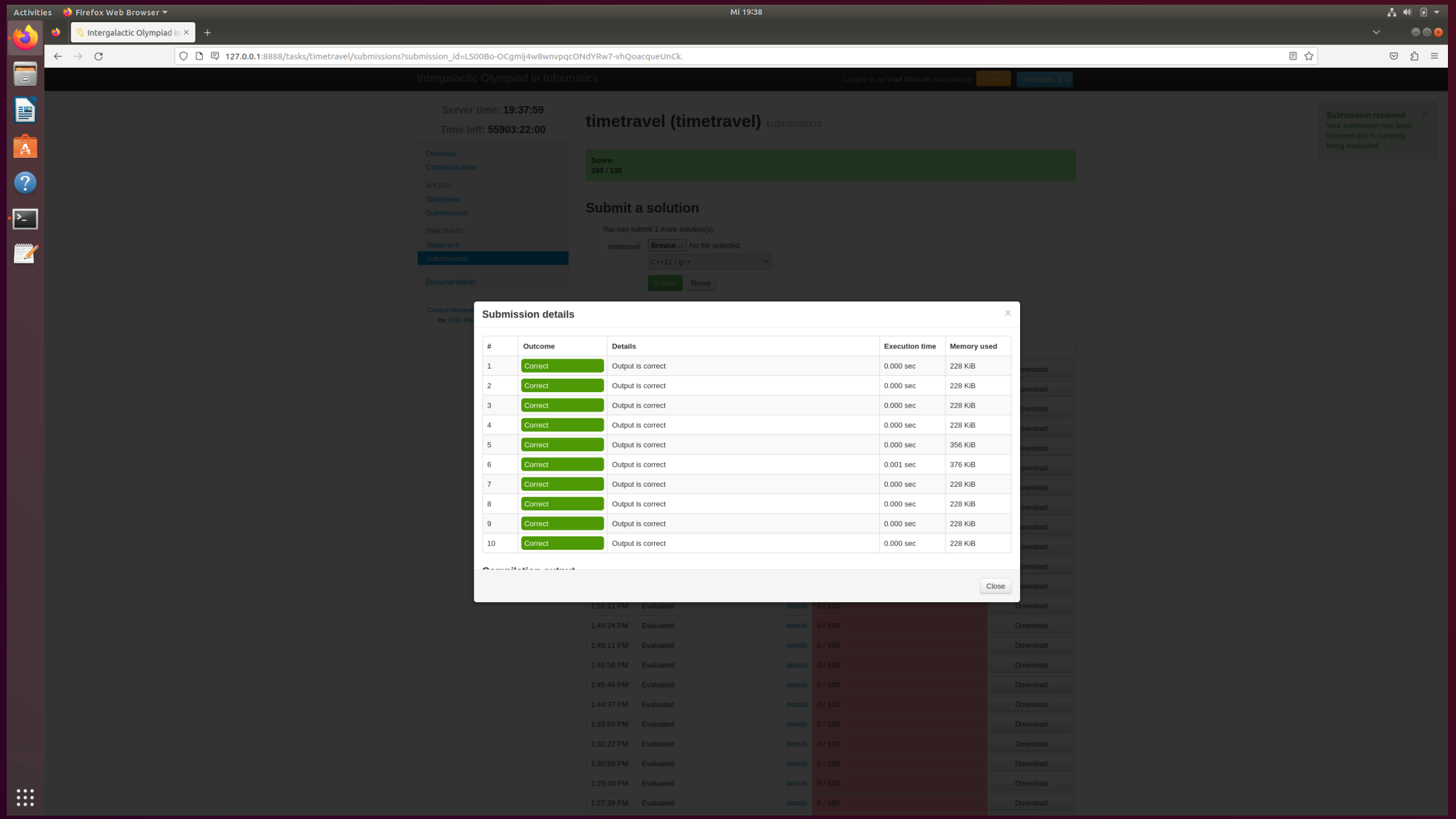
within ~~3 s~~ 1024 MB

#	n	Quadratic sieve 😴	p	≤	q
n_1					
n_2		2294425318445568 01			
n_3	4165284668300280686943035245599372335575205888685 42370321 61				
n_4	2857460635238413016190822254239995233328566661899994809 0429				
n_5	3402770026804173478318022986022417530565291870363936336655 1				
n_6	3834301323462670454015058000677810546177623265546222600948 3				
n_7	3389828227307414936935783784094971927702913185806389711311 3				
n_8	3100016766773050528042617304762349830637317127714930321544 7				
n_9	3111111111422253336444444444757444444444633352224111111111 3				
n_{10}	9985040655399021904836653226643576316006373274194532390018 7				

The final attempt

```cpp
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

ifstream fin("timetravel.in");
ofstream fout("timetravel.out");

string s;

int main()
{
    fin >> s;
    if(!s.compare("6658117"))
    {
        fout << "2017 3301";
        return 0;
    }
    if(!s.compare("229442531844556801"))
    {
        fout << "479001599 479001599";
        return 0;
    }
    if(!s.compare("4165284668300280686943035245599372335575205888685423703216 1"))
    {
        fout << "37335331319118115113110111753 11156415441175914623267940741 37";
        return 0;
    }
    if(!s.compare("285746063523841301619082225423999523332856666189999480904 29"))
    {
        fout << "285714285714285714285714285714 10001112223334445556667877889 99";
        return 0;
    }
    if(!s.compare("340277002680417347831802298602241753056529187036393633665 51"))
    {
        fout << "311141115191216151315181917191 109364203593374449019394878932 9";
        return 0;
    }
    if(!s.compare("383430132346267045401505800067781054617762326554622260094 83"))
    {
        fout << "111111777775555555333333333333 345085048608250393021332197155 1";
        return 0;
    }
    if(!s.compare("338982822730741493693578378409497192770291318580638971131 13"))
    {
        fout << "108888694504183521607680000011 311311311311311311311311311311 3";
        return 0;
    }
    if(!s.compare("310001676677305052804261730476234983063731712771493032154 47"))
    {
        fout << "105133911935073745000518620692 948636372141740238893441247963 ";
        return 0;
    }
    if(!s.compare("311111111142225333644444444475744444444463335222411111111 13"))
    {
        fout << "100000000010001000100000000011 311111111111111111111111111113 ";
        return 0;
    }
    if(!s.compare("998504065539902190483665322664357631600637327419453239001 87"))
    {
        fout << "191347024000932780814494239172 521828897393792061612773868631 1";
        return 0;
    }
    return 0;
}
```

Job done



Solving the challenge

Talking tech & Data for reproducibility

Setting up the OS

Ubuntu 18.04.6 (64-bit) was used in a virtual machine. The machine was configured using Oracle VirtualBox, having the following settings changed from the default:

- 2 allocated cores;
- 2048MB of memory;
- 25GB of storage space;
- 128MB of video memory.

For an optimised experience, VirtualBox Guest Additions were installed after first running:

```
sudo apt update
sudo apt install build-essential dkms linux-headers-$(uname -r)
```

Setting up the OS

It is *very* important to run the same version of Ubuntu (18.04) so that CMS can be properly installed. Otherwise, very dubious errors will occur and you may enter the nine circles of Python's Dependency Hell.

Setting up CMS

CMS was installed as recommended in this documentation: https://cms.readthedocs.io/en/v1.4/. For your interest, no `venv` was used and optional dependencies were not installed. CMS was downloaded as an archive, for Python dependencies to be installed using `pip`.

Setting up CMS

The file `config/cms.conf.sample` was copied to `/usr/local/etc/cms.config`, the only modifications done being changing the connection string so that the current Ubuntu user and password are specified, as well as changing the default `secret_key` for a random hex string and `keep_sandbox` to `false`.

Running the “contest”

CMS was run as described in the documentation, by first starting the Admin interface and setting up a contest. Please note that contest names **must not** contain spaces, or the contestant login process will error out!

Running the “contest”

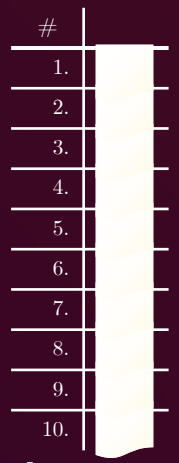
The feedback level was set to *full* for all tasks and the entire contest. The time limit was set to 3 seconds and the memory limit was set to 1024MiB (essentially, so as not to prematurely bottleneck the memory-consuming exfiltration). These limits are sensible for actual Olympiad tasks. The interval between two submissions was set to 60 seconds to ensure reliability and the total number of submissions for every task was set to 30 (– standard IOI practice as well).

P.S. It is worth mentioning that CMS warns the admin of the risks of malicious practices when setting the feedback level to *full* so that participants are able to see the time and memory usage and the verdicts. This feedback level is set to full in Olympiads nevertheless.

On memory usage

When using a single machine, with the total machine memory set to 2048MiB and the per-test-case memory to 1024MiB, problems might (and probably will) arise.

- Worker 0
- Worker 1
- Worker 2
- Worker 3
- Worker 4
- Worker 5
- Worker 6
- Worker 7
- Worker 8
- Worker 9
- Worker 10
- Worker 11
- Worker 12
- Worker 13
- Worker 14
- Worker 15

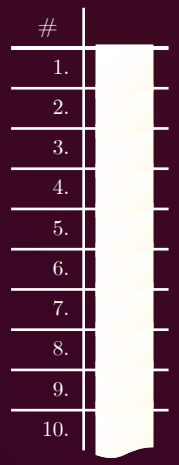


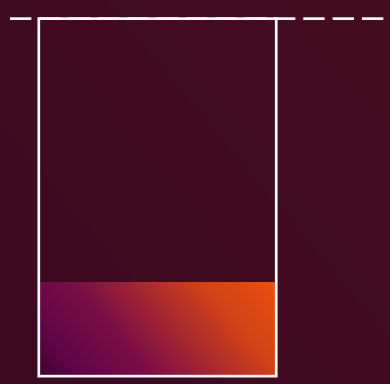
There are 16 worker processes that work in parallel and queue up test case evaluations when the contestants submit them. Here, *each* process will try to use 1024MiB, often at the same time. This can not only occasionally cause crashes, but will also consistently affect the reported memory usage of each test and, thus, the exfiltration.

On memory usage

By disabling all but one worker process, it will get the entire memory pie and, despite taking more physical time, serialising the evaluation will produce much more reliable results.

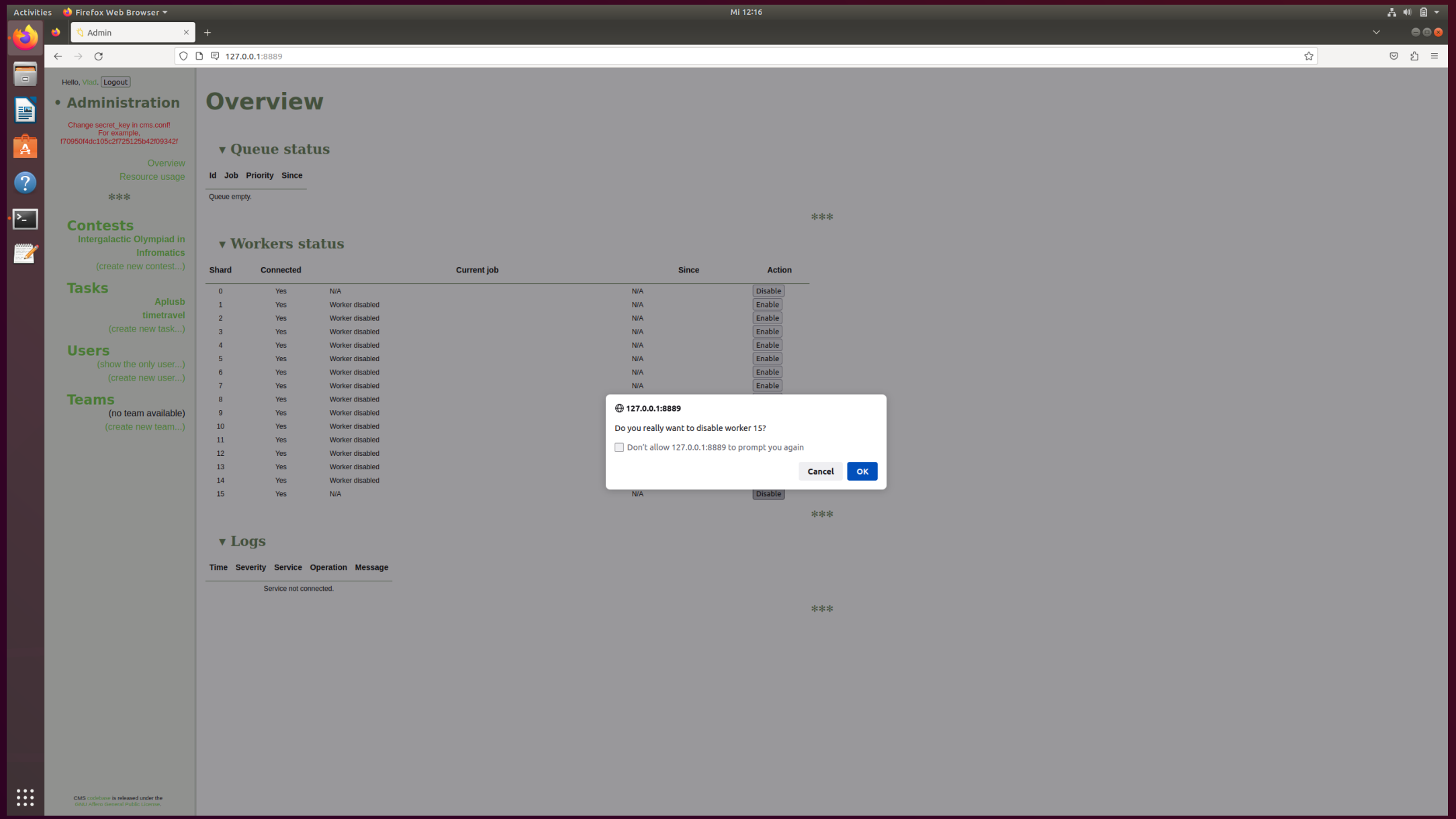
- Worker 0
- Worker 1
- Worker 2
- Worker 3
- Worker 4
- Worker 5
- Worker 6
- Worker 7
- Worker 8
- Worker 9
- Worker 10
- Worker 11
- Worker 12
- Worker 13
- Worker 14
- Worker 15





This setup will emulate the ‘professional’ configurations of CMS, spanning an entire network of single-worker machines, at the only cost of not being able to support multiple contestants at the same time as easily.

Disabling workers



Disabling every worker except from Worker 0 to ensure fairness during evaluation.

Further developments

Increasing the throughput

87 bits?

More granular memory allocation? Taking in 6 bits (64 values) for the memory and multiplying them by only 4MiB?

Trying to use more than 256MiB of memory?

Trying to only exfiltrate 1 bit for the verdict and instead use a busy loop to involve the time in the useful feedback?

Using another base and rethinking the scheme entirely?

Going next level?

Going next level?

8+ bits? error correction scoring

Getting rid of errors

error correction

Taking the suspicious exfiltration, rotating the bits to be exfiltrated by three places, then running the new attempt again?

~~exf(1010011)~~ → try exf(0111010)

This would ensure a correct transmission.

Using some error correcting codes for analog domains (0-31) to further improve reliability?

Thanks for your attention!