UNIVERSIDAD NACIONAL DE SAN AGUSTÍN DE AREQUIPA

FACULTAD DE PRODUCCION Y SERVICIOS

ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

Curso: Laboratorio de Análisis y Diseño de Algoritmos

Práctica 10

Estudiante: Tapara Quispe, Fabiola Grissel

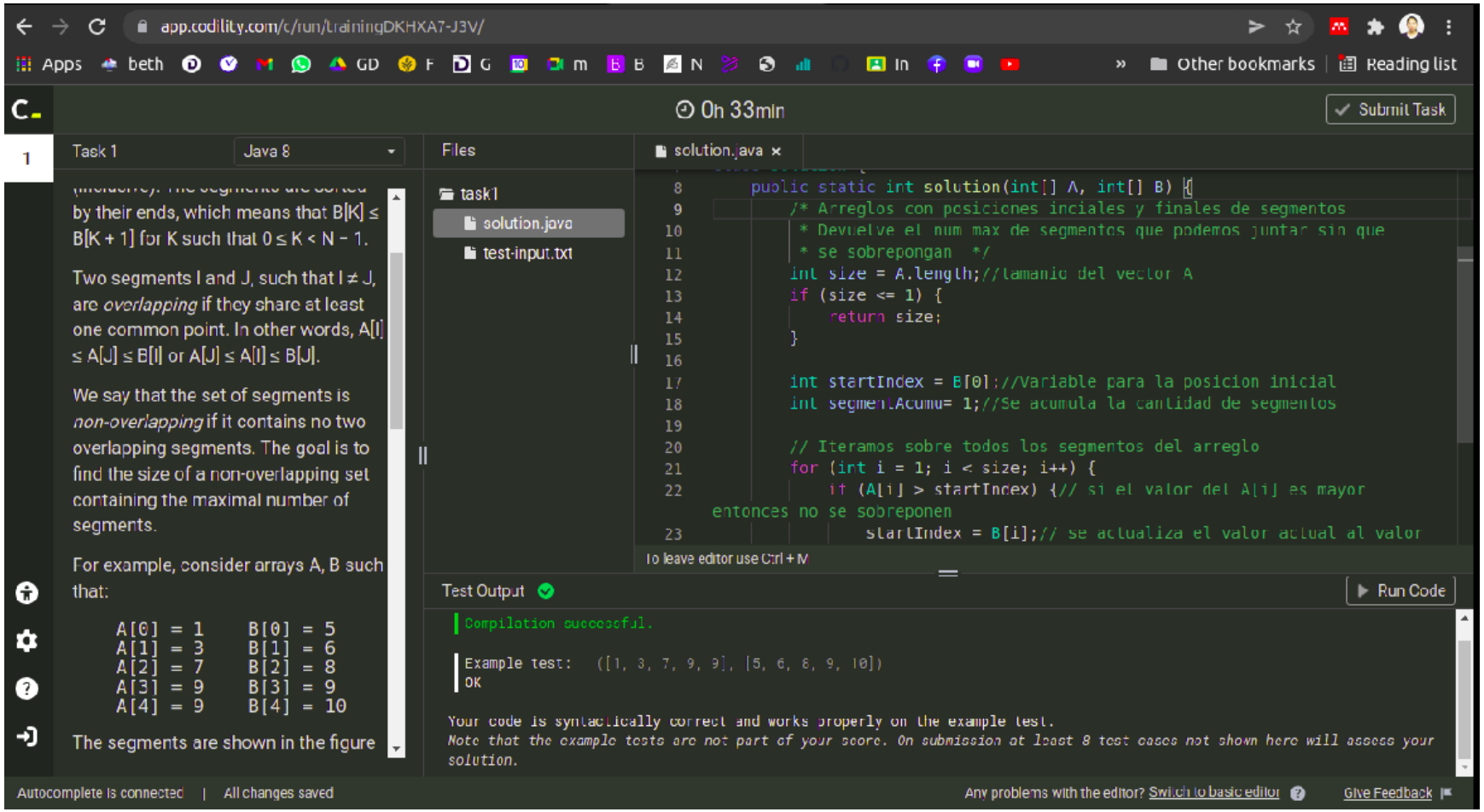
Docente: Alex Josue Florez Farfan

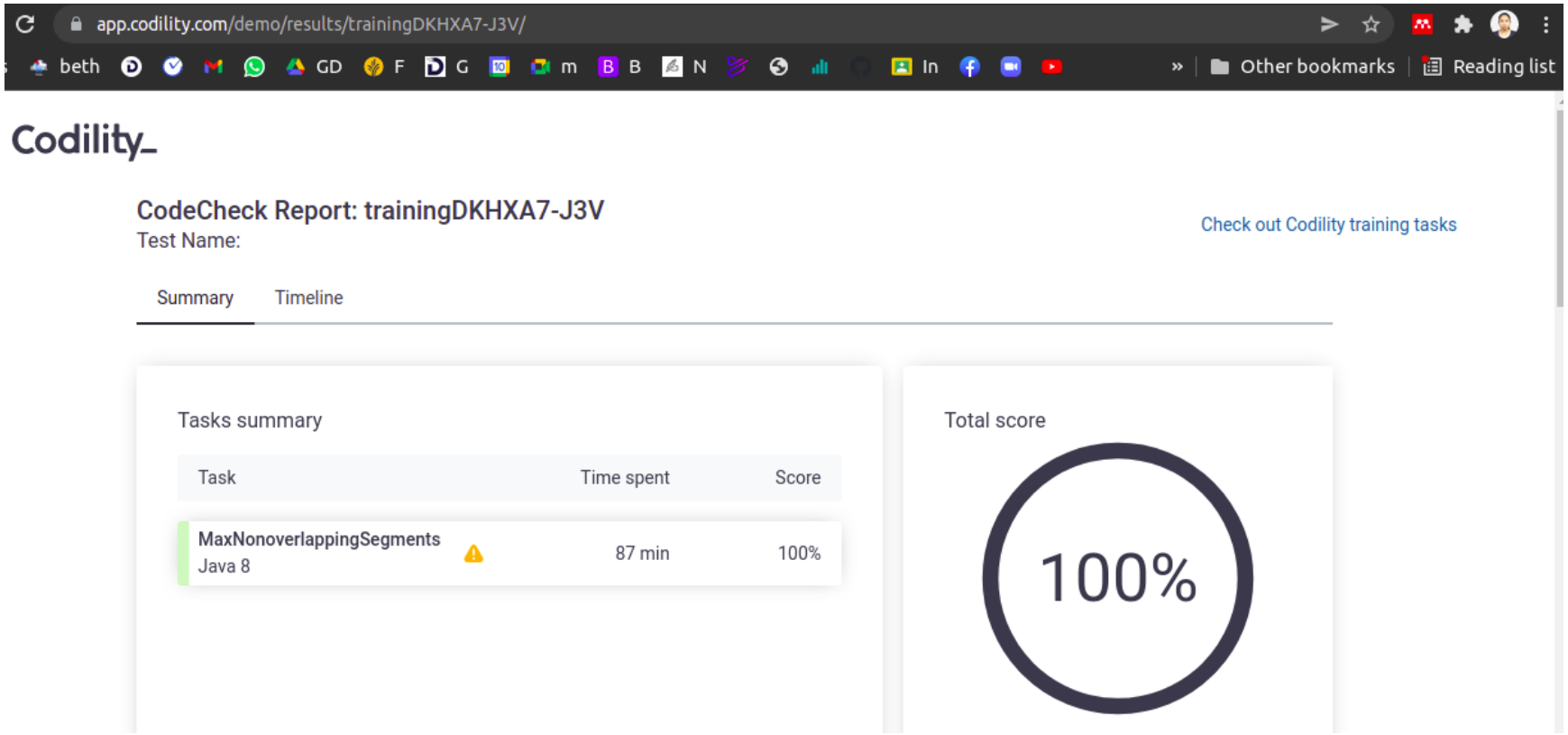
Sección: “B”

Arequipa - Perú

Diciembre 2021

Ejercicio 1


app.codility.com/c/run/trainingDKHXA7-J3V/
Apps beth GD Other bookmarks Reading list
0h 33min
Submit Task
Task 1
Java 8
Files
task1
solution.java
test-input.txt
solution.java ×
by their ends, which means that B[K] ≤ B[K + 1] for K such that 0 ≤ K < N − 1.
Two segments I and J, such that I ≠ J, are overlapping if they share at least one common point. In other words, A[I] ≤ A[J] ≤ B[I] or A[J] ≤ A[I] ≤ B[J].
We say that the set of segments is non-overlapping if it contains no two overlapping segments. The goal is to find the size of a non-overlapping set containing the maximal number of segments.
For example, consider arrays A, B such that:
A[0] = 1 B[0] = 5
A[1] = 3 B[1] = 6
A[2] = 7 B[2] = 8
A[3] = 9 B[3] = 9
A[4] = 9 B[4] = 10
The segments are shown in the figure
public static int solution(int[] A, int[] B) {
/* Arreglos con posiciones inciales y finales de segmentos
* Devuelve el num max de segmentos que podemos juntar sin que
* se sobrepongan */
int size = A.length;//tamanio del vector A
if (size <= 1) {
return size;
}
int startIndex = B[0];//Variable para la posicion inicial
int segmentAcumu= 1;//Se acumula la cantidad de segmentos
// Iteramos sobre todos los segmentos del arreglo
for (int i = 1; i < size; i++) {
if (A[i] > startIndex) {// si el valor del A[i] es mayor entonces no se sobreponen
startIndex = B[i];// se actualiza el valor actual al valor
To leave editor use Ctrl+M
Test Output
Run Code
Compilation successful.
Example test: ([1, 3, 7, 9, 9], [5, 6, 8, 9, 10])
OK
Your code is syntactically correct and works properly on the example test.
Note that the example tests are not part of your score. On submission at least 8 test cases not shown here will assess your solution.
Autocomplete is connected | All changes saved
Any problems with the editor? Switch to basic editor
Give Feedback


app.codility.com/demo/results/trainingDKHXA7-J3V/
beth
GD
F
G
m
B
N
In
Other bookmarks
Reading list
Codility_
CodeCheck Report: trainingDKHXA7-J3V
Test Name:
Check out Codility training tasks
Summary
Timeline
Tasks summary
Task
Time spent
Score
MaxNonoverlappingSegments
Java 8
87 min
100%
Total score
100%

app.codility.com/demo/results/trainingDKHXA7-J3V/

Apps beth GD F G m B N In Other bookmarks Reading

Tasks Details

Easy

1. **MaxNonoverlappingSegments**
Find a maximal set of non-overlapping segments.

Task Score	Correctness	Performance
100%	100%	100%

Task description

Located on a line are N segments, numbered from 0 to N − 1, whose positions are given in arrays A and B. For each I ($0 \le I < N$) the position of segment I is from A[I] to B[I] (inclusive). The segments are sorted by their ends, which means that $B[K] \le B[K + 1]$ for K such that $0 \le K < N - 1$.

Two segments I and J, such that $I \ne J$, are *overlapping* if they share at least one common point. In other words, $A[I] \le A[J] \le B[I]$ or $A[J] \le A[I] \le B[J]$.

We say that the set of segments is *non-overlapping* if it contains no two overlapping segments. The goal is to find the size of a non-overlapping set containing the maximal number of segments.

For example, consider arrays A, B such that:

```
A[0] = 1    B[0] = 5
A[1] = 3    B[1] = 6
A[2] = 7    B[2] = 8
A[3] = 9    B[3] = 9
A[4] = 9    B[4] = 10
```

Solution

Programming language used:	Java 8
Total time used:	87 minutes
Effective time used:	87 minutes
Notes:	*not defined yet*

Task timeline

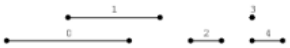
20:50:26 22:17:02

segments. The goal is to find the size of a non-overlapping set containing the maximal number of segments.

For example, consider arrays A, B such that:

```
A[0] = 1    B[0] = 5
A[1] = 3    B[1] = 6
A[2] = 7    B[2] = 8
A[3] = 9    B[3] = 9
A[4] = 9    B[4] = 10
```

The segments are shown in the figure below.



The size of a non-overlapping set containing a maximal number of segments is 3. For example, possible sets are {0, 2, 3}, {0, 2, 4}, {1, 2, 3} or {1, 2, 4}. There is no non-overlapping set with four segments.

Write a function:

```
class Solution { public int solution(int[] A, int[] B); }
```

that, given two arrays A and B consisting of N integers, returns the size of a non-overlapping set containing a maximal number of segments.

For example, given arrays A, B shown above, the function should return 3, as explained above.

Write an **efficient** algorithm for the following assumptions:

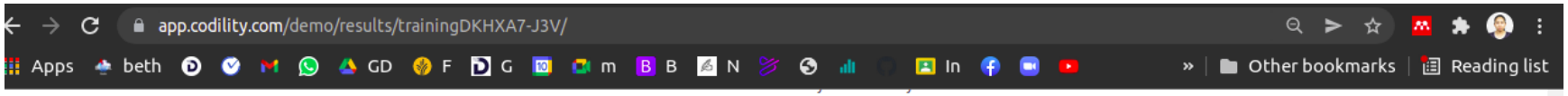
- N is an integer within the range [0..30,000];
- each element of arrays A and B is an integer within the range [0..1,000,000,000];
- A[I] ≤ B[I], for each I (0 ≤ I < N);
- B[K] ≤ B[K + 1], for each K (0 ≤ K < N − 1).

Task timeline

20:50:26 22:17:02

Code: 22:17:01 UTC, java, final, score: 100 show code in pop-up

```java
// you can also use imports, for example:
// import java.util.*;

// you can write to stdout for debugging purposes, e.g.
// System.out.println("this is a debug message");

class Solution {
        public static int solution(int[] A, int[] B) {
                /* Arreglos con posiciones inciales y finales
                 * Devuelve el num max de segmentos que podemo
                 * se sobrepongan  */
                int size = A.length;//tamanio del vector A
                if (size <= 1) {
                        return size;
                }

                int startIndex = B[0];//Variable para la posic
                int segmentAcumu= 1;//Se acumula la cantidad d

                // Iteramos sobre todos los segmentos del arre
                for (int i = 1; i < size; i++) {
                        if (A[i] > startIndex) {// si el valor
                                startIndex = B[i];// se actual
                                segmentAcumu++;//aumenta segme
                        }
                }
                return segmentAcumu;// devolvemos la cantidad
        }
}
```



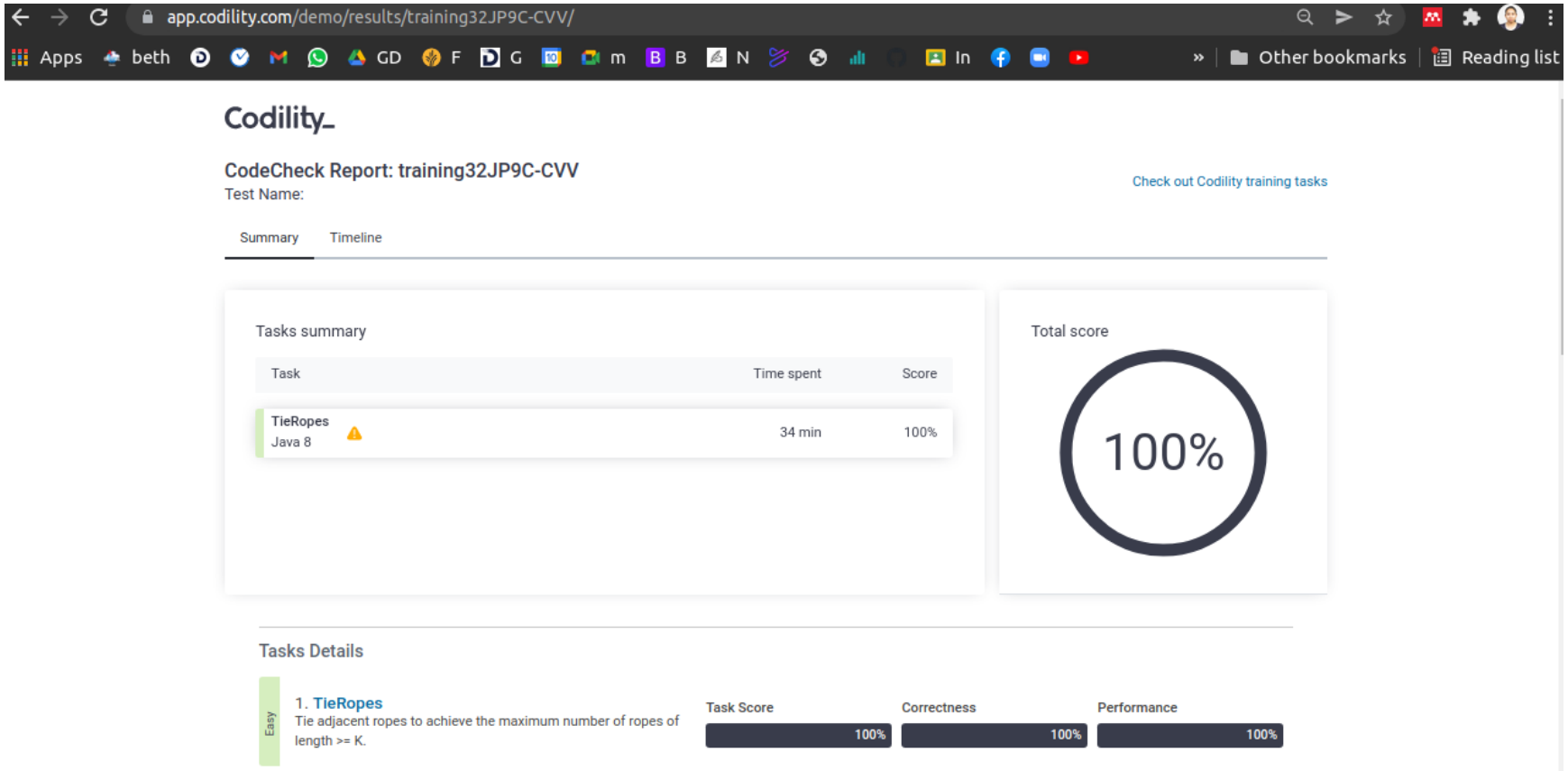
The solution obtained perfect score.

Analysis

Detected time complexity: **O(N)**

expand all	Example tests	
▶	example example test	✔ OK
expand all	**Correctness tests**	
▶	extreme_empty_and_single empty and single element	✔ OK
▶	small_functional many overlapping	✔ OK
▶	small_non_overlapping all non-overlapping	✔ OK
▶	small_all_overlapping small functional	✔ OK
▶	small_random_same_length small random, length = ~40	✔ OK
expand all	**Performance tests**	
▶	medium_random_differ_length medium random, length = ~300	✔ OK
▶	large_points all points, length = ~30,000	✔ OK
▶	large_random_many_overlapping large random, length = ~30,000	✔ OK
▶	large_random_few_overlapping large random, length = ~30,000	✔ OK
▶	extreme_large large size of intervals, length = ~30,000	✔ OK

Ejercicio 2


app.codility.com/demo/results/training32JP9C-CVV/
Apps
beth
GD
F
G
m
B
N
In
Other bookmarks
Reading list
Codility_
CodeCheck Report: training32JP9C-CVV
Test Name:
Check out Codility training tasks
Summary
Timeline
Tasks summary
Task
Time spent
Score
TieRopes
Java 8
34 min
100%
Total score
100%
Tasks Details
Easy
1. TieRopes
Tie adjacent ropes to achieve the maximum number of ropes of length >= K.
Task Score
100%
Correctness
100%
Performance
100%

app.codility.com/demo/results/training32JP9C-CVV/

Apps beth GD F G m B B N In Other bookmarks Reading list

Task description

There are N ropes numbered from 0 to N − 1, whose lengths are given in an array A, lying on the floor in a line. For each I (0 ≤ I < N), the length of rope I on the line is A[I].

We say that two ropes I and I + 1 are *adjacent*. Two adjacent ropes can be tied together with a knot, and the length of the tied rope is the sum of lengths of both ropes. The resulting new rope can then be tied again.

For a given integer K, the goal is to tie the ropes in such a way that the number of ropes whose length is greater than or equal to K is maximal.

For example, consider K = 4 and array A such that:

```
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 1
A[5] = 1
A[6] = 3
```

Solution

Programming language used:	Java 8
Total time used:	34 minutes
Effective time used:	34 minutes
Notes:	*not defined yet*

Task timeline

11:41:18 12:15:11

app.codility.com/demo/results/training32JP9C-CVV/

Apps beth GD F G m B B N In Other bookmarks Reading list

The ropes are shown in the figure below.

We can tie:

- rope 1 with rope 2 to produce a rope of length A[1] + A[2] = 5;
- rope 4 with rope 5 with rope 6 to produce a rope of length A[4] + A[5] + A[6] = 5.

After that, there will be three ropes whose lengths are greater than or equal to K = 4. It is not possible to produce four such ropes.

Write a function:

```
class Solution { public int solution(int K, int[] A); }
```

that, given an integer K and a non-empty array A of N integers, returns the maximum number of ropes of length greater than or equal to K that can be created.

For example, given K = 4 and array A such that:

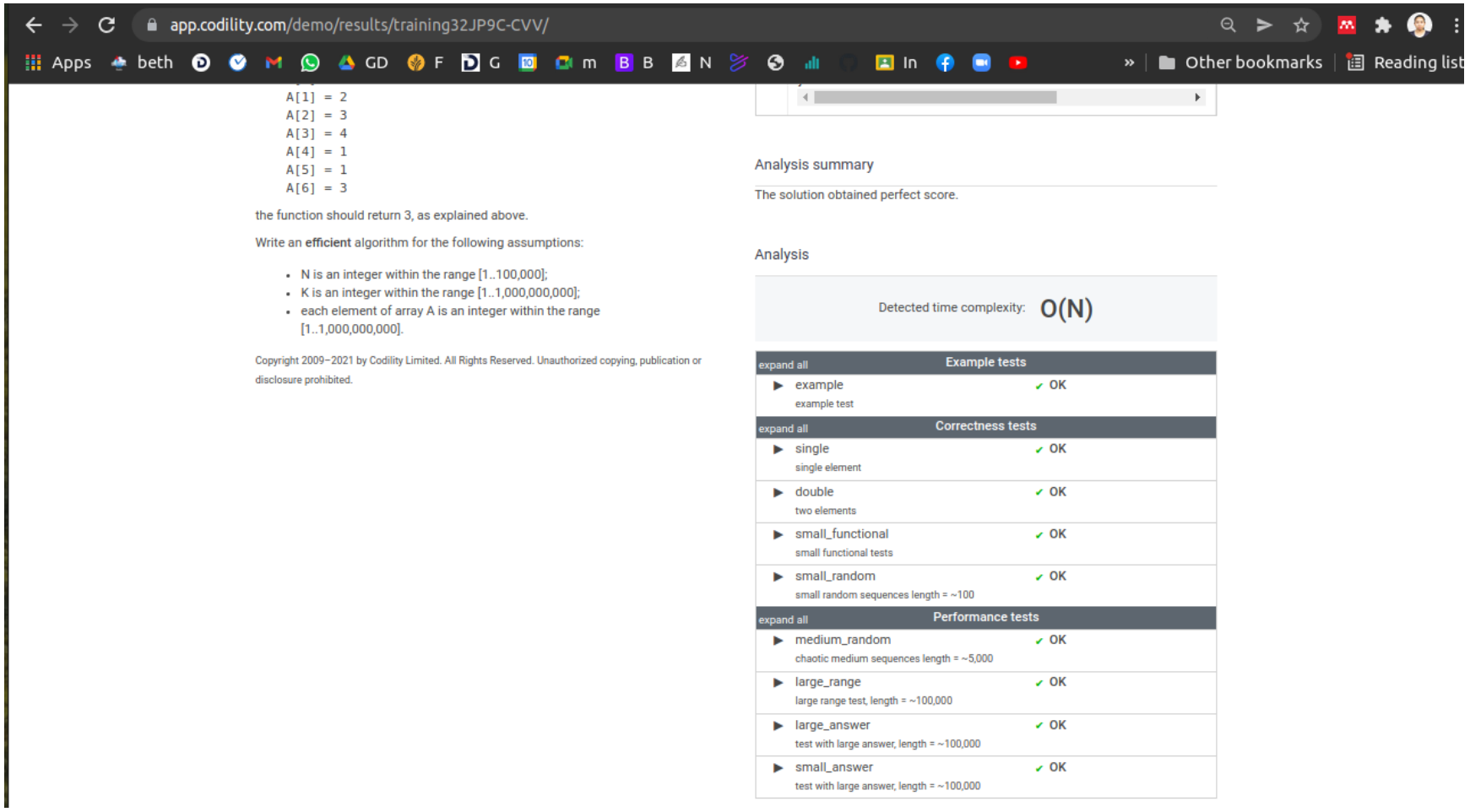
```
A[0] = 1
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 1
A[5] = 1
A[6] = 3
```

Code: 12:15:11 UTC, java, final, score: 100 show code in pop-up

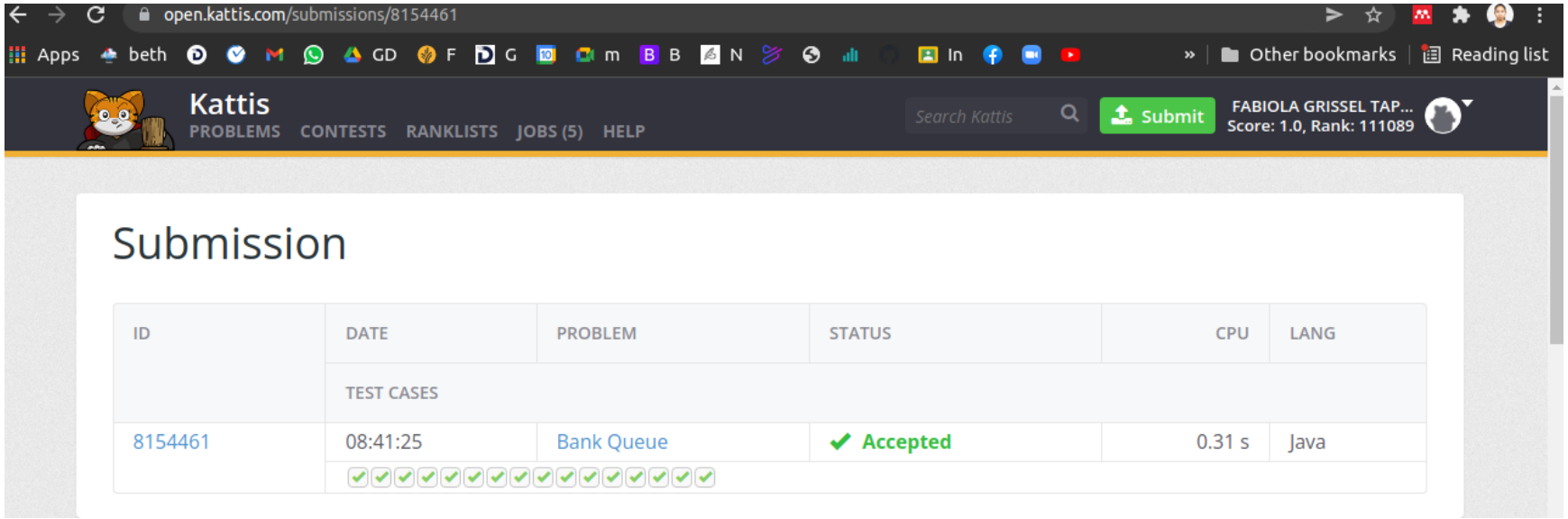
```
1  // you can also use imports, for example:
2  // import java.util.*;
3
4  // you can write to stdout for debugging purposes, e.g.
5  // System.out.println("this is a debug message");
6
7  class Solution {
8          public static int solution(int k, int[] a) {
9                  int countRopes = 0;// numeros de cuerdas amarr
10                 int ropeTam = 0;// guarda el tamanio de la cue
11                 for(int i = 0; i < a.length; i++) {// Iterarem
12                         ropeTam += a[i];// Agregamos la longit
13                         if(ropeTam >= k) {// Si el tamanio es
14                                 ropeTam = 0;// Actualiza el ta
15                                 countRopes++;// Agregamos una
16                         }
17                 }
18                 return countRopes;//devolvemos el total de cue
19         }
20 }
```

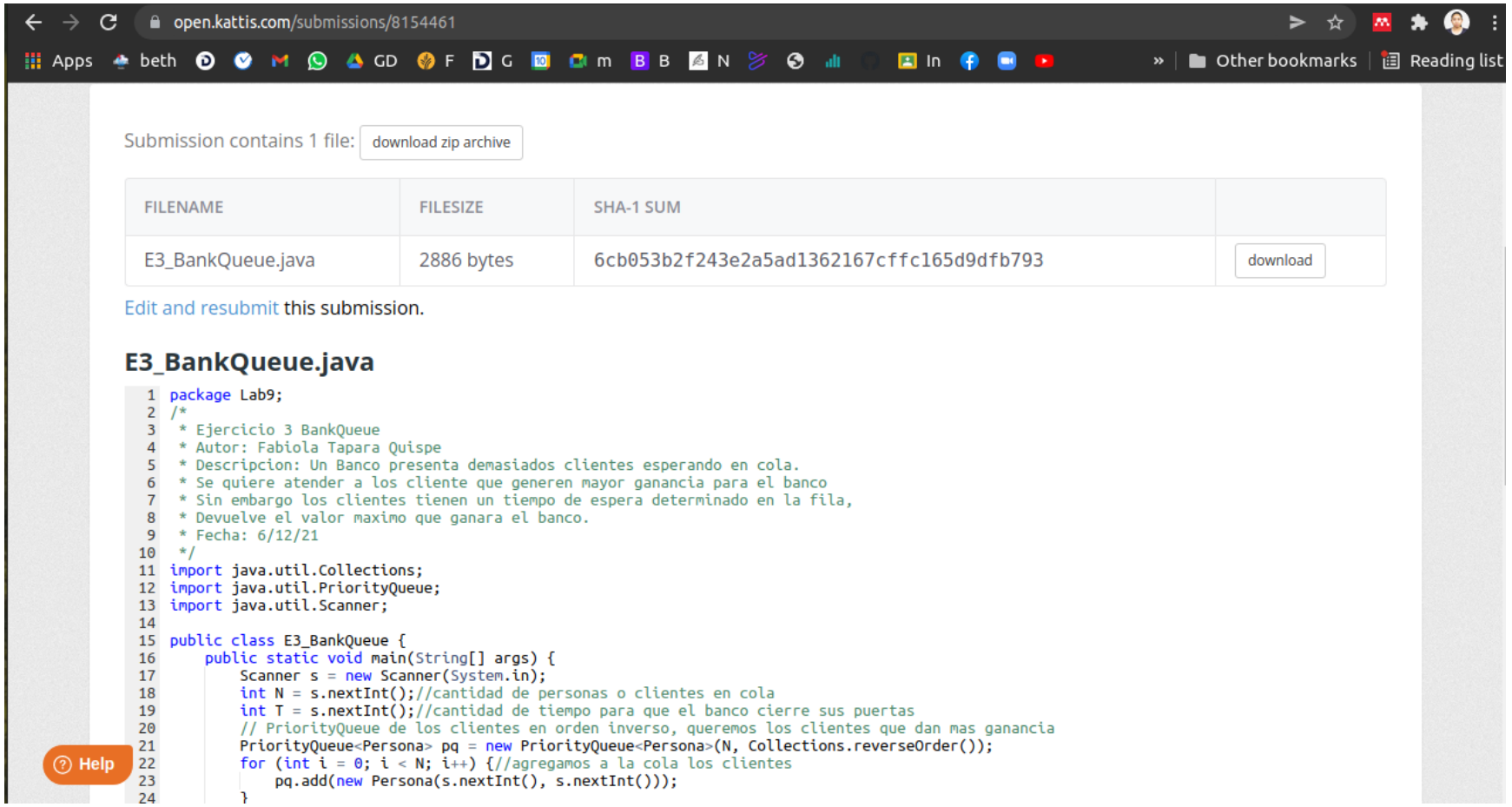
Analysis summary

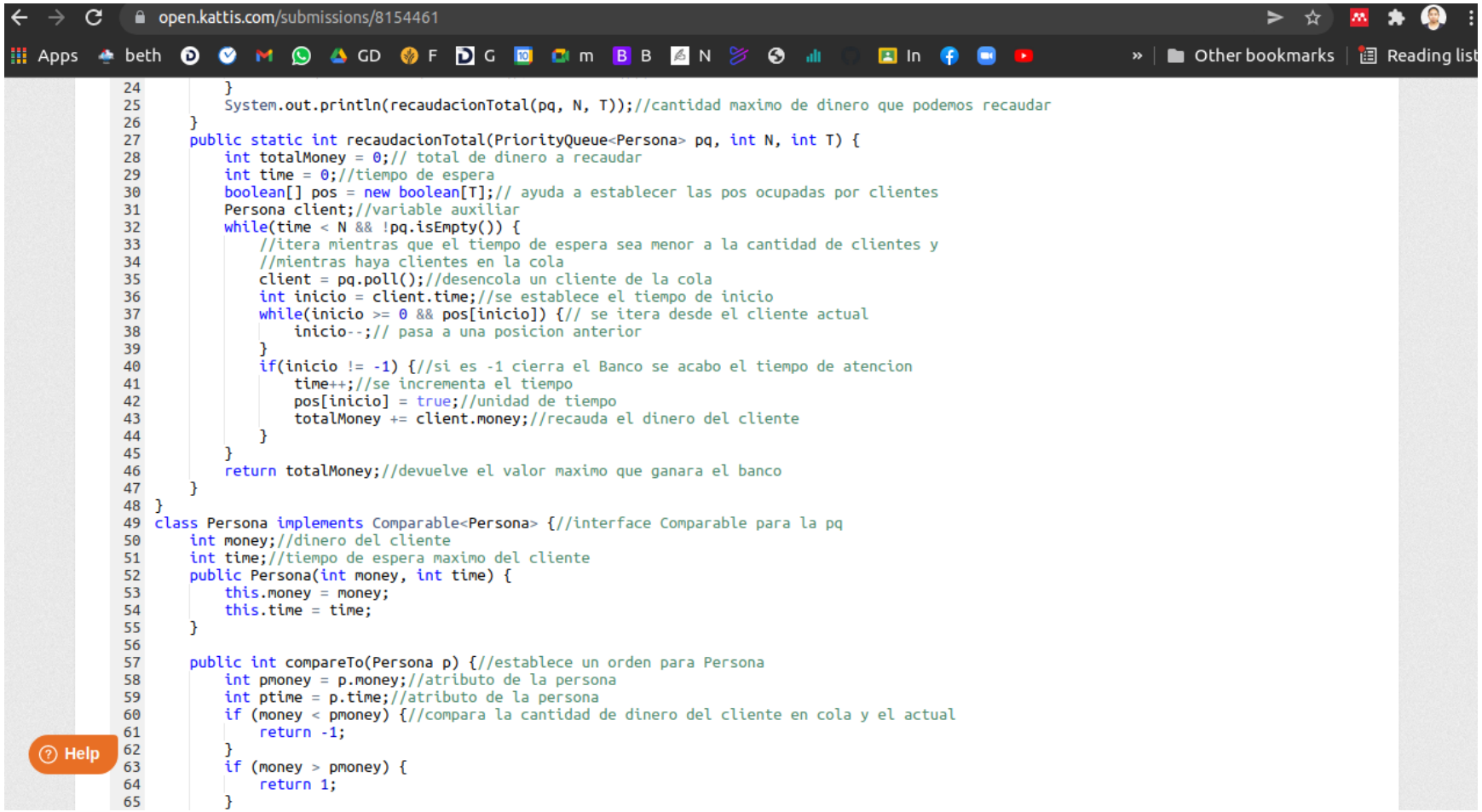
The solution obtained perfect score.


app.codility.com/demo/results/training32JP9C-CVV/
Apps
beth
GD
F
G
m
B
N
In
Other bookmarks
Reading list
A[1] = 2
A[2] = 3
A[3] = 4
A[4] = 1
A[5] = 1
A[6] = 3
the function should return 3, as explained above.
Write an efficient algorithm for the following assumptions:
N is an integer within the range [1..100,000];
K is an integer within the range [1..1,000,000,000];
each element of array A is an integer within the range [1..1,000,000,000].
Copyright 2009–2021 by Codility Limited. All Rights Reserved. Unauthorized copying, publication or disclosure prohibited.
Analysis summary
The solution obtained perfect score.
Analysis
Detected time complexity: O(N)
expand all
Example tests
example
example test
OK
expand all
Correctness tests
single
single element
OK
double
two elements
OK
small_functional
small functional tests
OK
small_random
small random sequences length = ~100
OK
expand all
Performance tests
medium_random
chaotic medium sequences length = ~5,000
OK
large_range
large range test, length = ~100,000
OK
large_answer
test with large answer, length = ~100,000
OK
small_answer
test with large answer, length = ~100,000
OK

Ejercicio 3

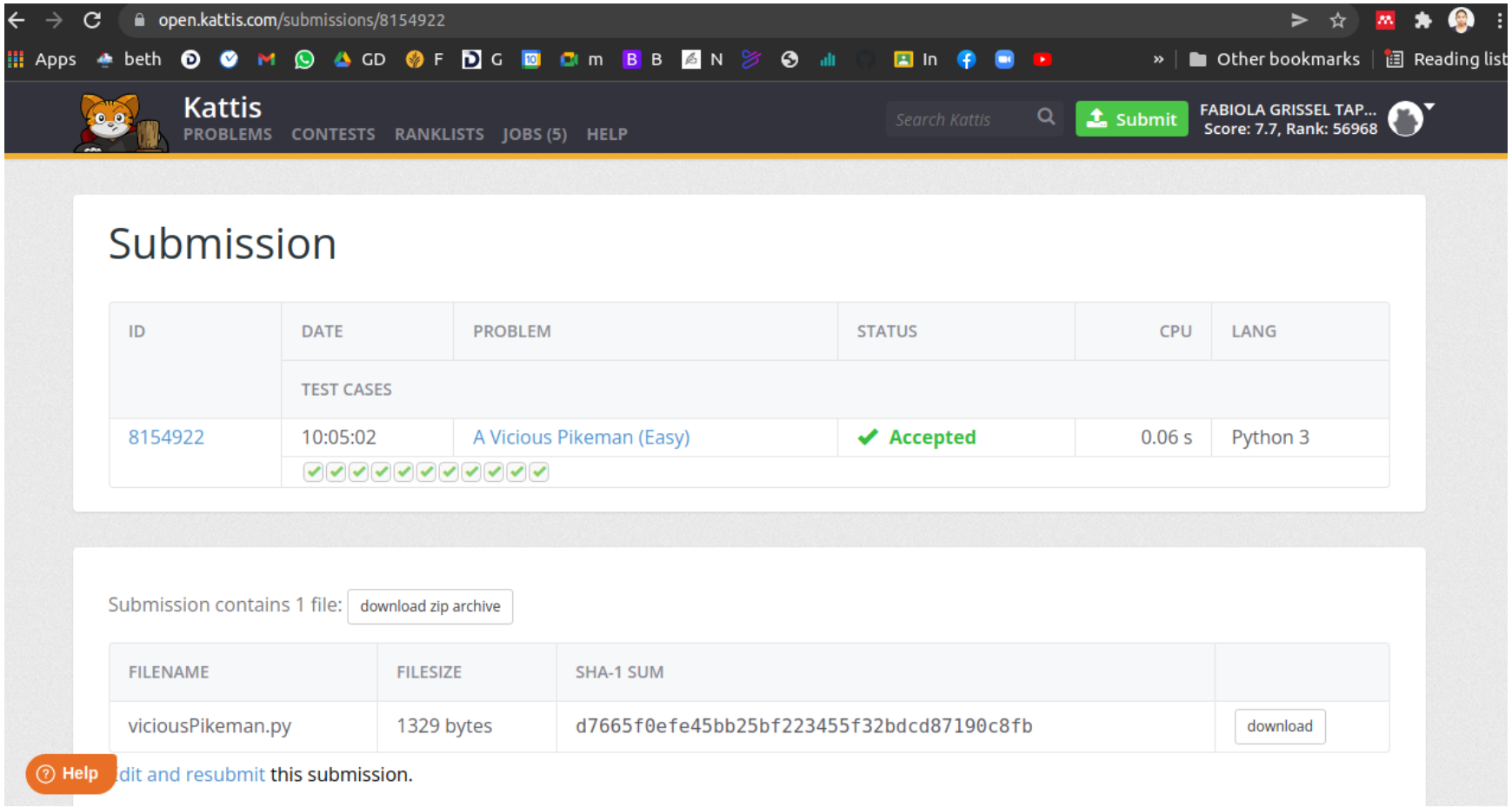

open.kattis.com/submissions/8154461
Apps
beth
GD
F
G
m
B
N
In
Other bookmarks
Reading list
Kattis
PROBLEMS
CONTESTS
RANKLISTS
JOBS (5)
HELP
Search Kattis
Submit
FABIOLA GRISSEL TAP...
Score: 1.0, Rank: 111089
Submission
ID
DATE
PROBLEM
STATUS
CPU
LANG
TEST CASES
8154461
08:41:25
Bank Queue
Accepted
0.31 s
Java

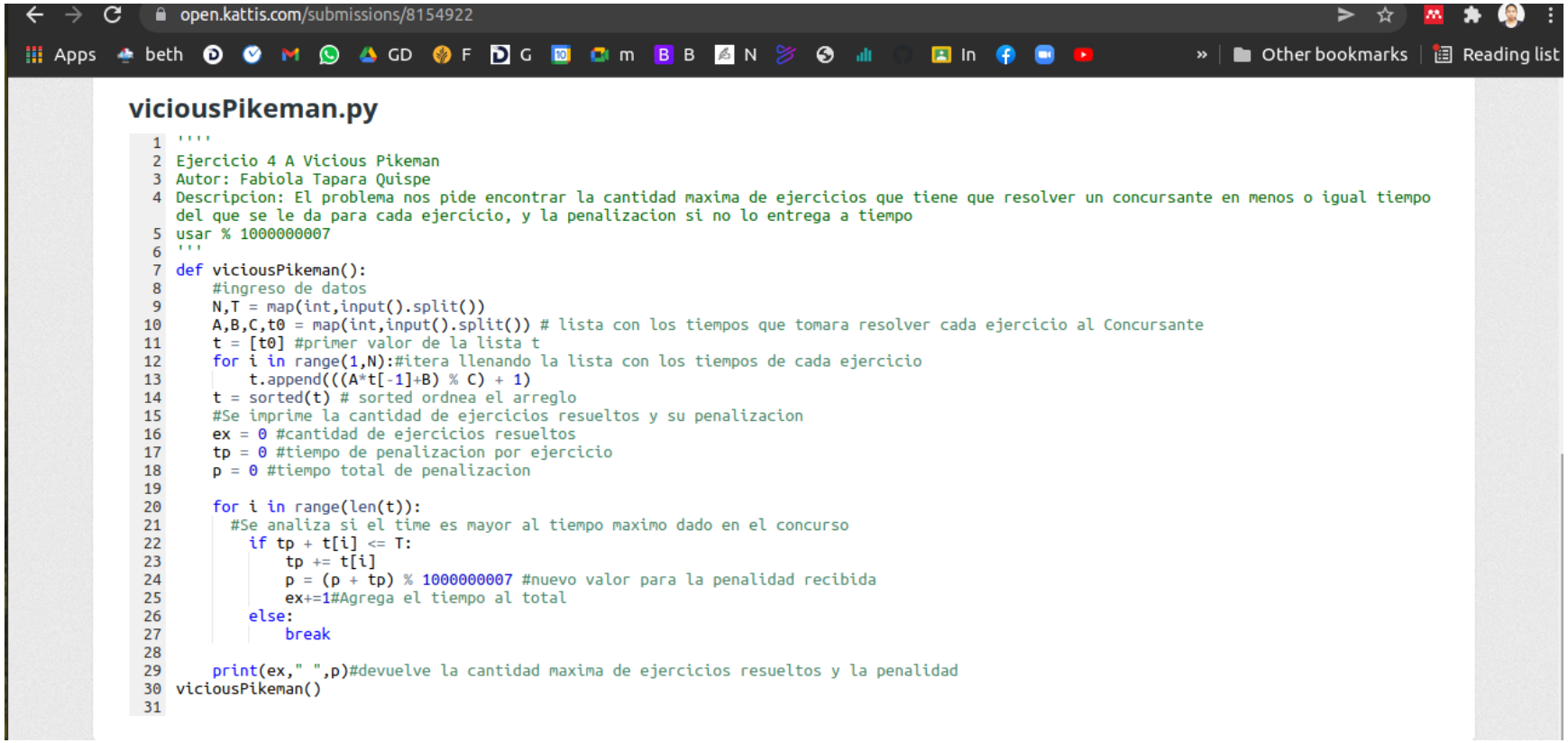

open.kattis.com/submissions/8154461
Apps
beth
GD
F
G
m
B
N
In
Other bookmarks
Reading list
Submission contains 1 file:
download zip archive
FILENAME
FILESIZE
SHA-1 SUM
E3_BankQueue.java
2886 bytes
6cb053b2f243e2a5ad1362167cffc165d9dfb793
download
Edit and resubmit this submission.
E3_BankQueue.java
1 package Lab9;
2 /*
3 * Ejercicio 3 BankQueue
4 * Autor: Fabiola Tapara Quispe
5 * Descripcion: Un Banco presenta demasiados clientes esperando en cola.
6 * Se quiere atender a los cliente que generen mayor ganancia para el banco
7 * Sin embargo los clientes tienen un tiempo de espera determinado en la fila,
8 * Devuelve el valor maximo que ganara el banco.
9 * Fecha: 6/12/21
10 */
11 import java.util.Collections;
12 import java.util.PriorityQueue;
13 import java.util.Scanner;
14
15 public class E3_BankQueue {
16 public static void main(String[] args) {
17 Scanner s = new Scanner(System.in);
18 int N = s.nextInt();//cantidad de personas o clientes en cola
19 int T = s.nextInt();//cantidad de tiempo para que el banco cierre sus puertas
20 // PriorityQueue de los clientes en orden inverso, queremos los clientes que dan mas ganancia
21 PriorityQueue<Persona> pq = new PriorityQueue<Persona>(N, Collections.reverseOrder());
22 for (int i = 0; i < N; i++) {//agregamos a la cola los clientes
23 pq.add(new Persona(s.nextInt(), s.nextInt()));
24 }
Help



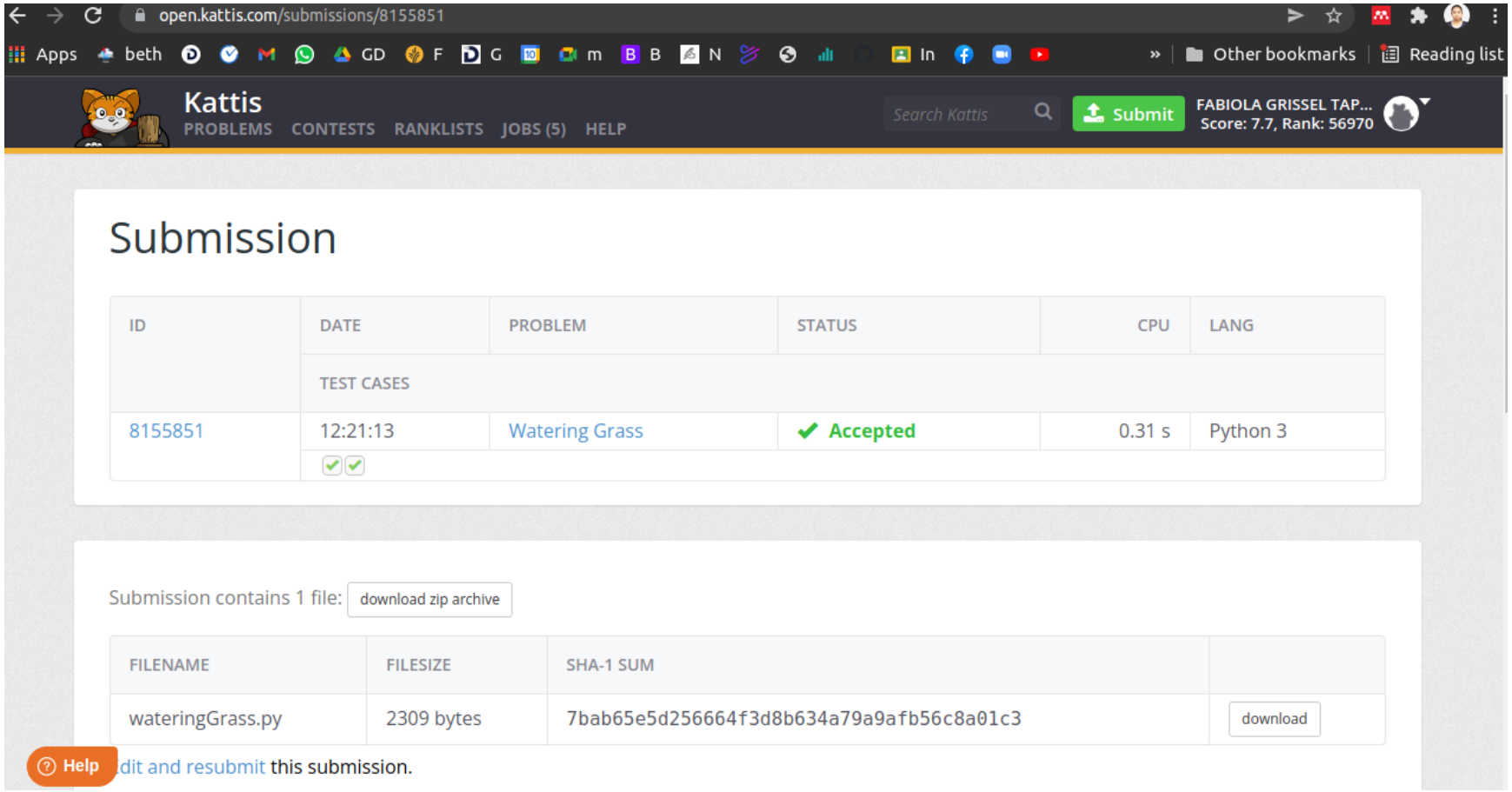

```java
        }
        System.out.println(recaudacionTotal(pq, N, T));//cantidad maximo de dinero que podemos recaudar
    }
    public static int recaudacionTotal(PriorityQueue<Persona> pq, int N, int T) {
        int totalMoney = 0;// total de dinero a recaudar
        int time = 0;//tiempo de espera
        boolean[] pos = new boolean[T];// ayuda a establecer las pos ocupadas por clientes
        Persona client;//variable auxiliar
        while(time < N && !pq.isEmpty()) {
            //itera mientras que el tiempo de espera sea menor a la cantidad de clientes y
            //mientras haya clientes en la cola
            client = pq.poll();//desencola un cliente de la cola
            int inicio = client.time;//se establece el tiempo de inicio
            while(inicio >= 0 && pos[inicio]) {// se itera desde el cliente actual
                inicio--;// pasa a una posicion anterior
            }
            if(inicio != -1) {//si es -1 cierra el Banco se acabo el tiempo de atencion
                time++;//se incrementa el tiempo
                pos[inicio] = true;//unidad de tiempo
                totalMoney += client.money;//recauda el dinero del cliente
            }
        }
        return totalMoney;//devuelve el valor maximo que ganara el banco
    }
}
class Persona implements Comparable<Persona> {//interface Comparable para la pq
    int money;//dinero del cliente
    int time;//tiempo de espera maximo del cliente
    public Persona(int money, int time) {
        this.money = money;
        this.time = time;
    }

    public int compareTo(Persona p) {//establece un orden para Persona
        int pmoney = p.money;//atributo de la persona
        int ptime = p.time;//atributo de la persona
        if (money < pmoney) {//compara la cantidad de dinero del cliente en cola y el actual
            return -1;
        }
        if (money > pmoney) {
            return 1;
        }
```

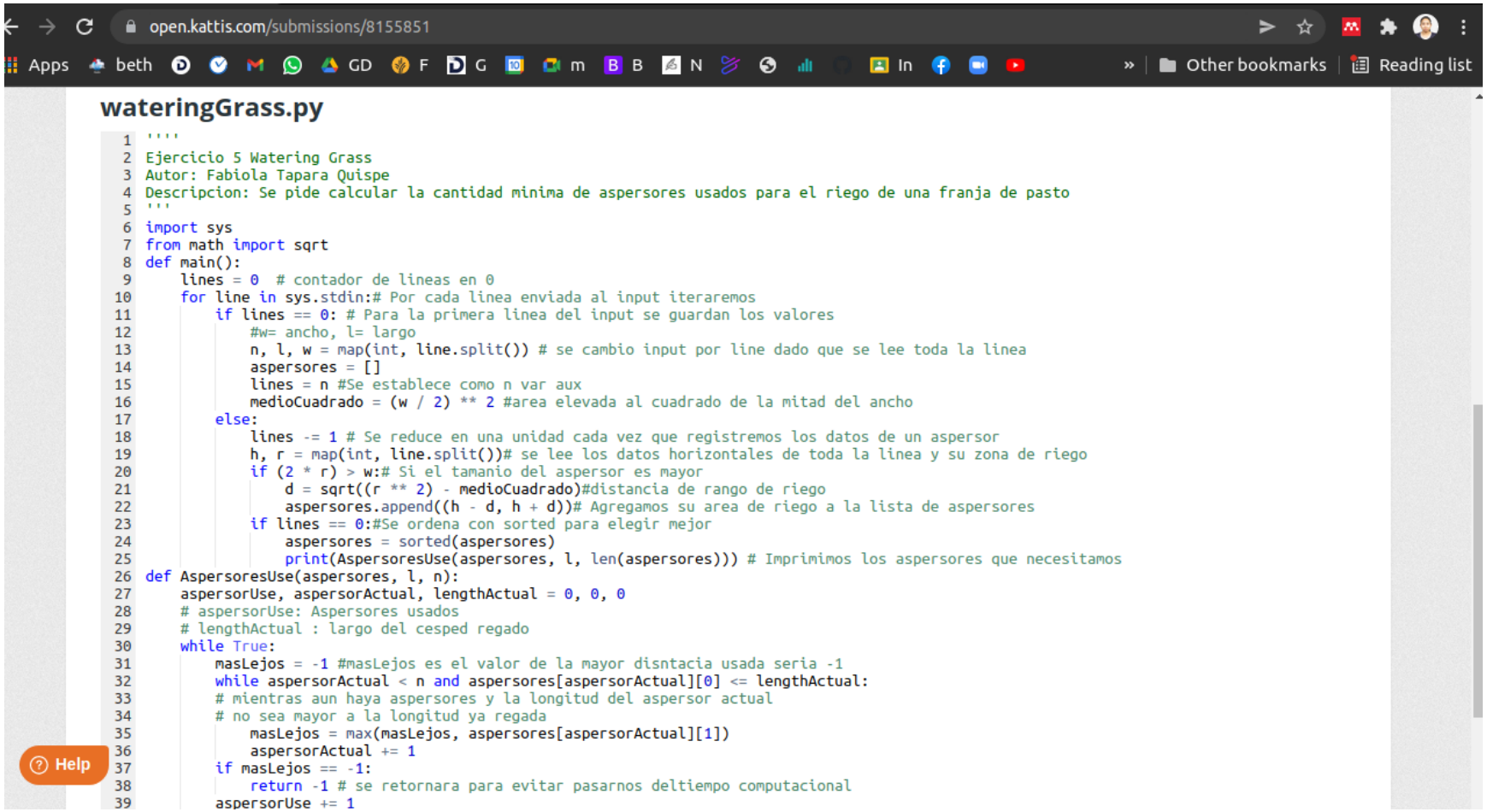
Ejercicio 4


open.kattis.com/submissions/8154922
Apps beth GD F G m B B N In Other bookmarks Reading list
Kattis
PROBLEMS CONTESTS RANKLISTS JOBS (5) HELP
Search Kattis
Submit
FABIOLA GRISSEL TAP...
Score: 7.7, Rank: 56968
Submission
ID DATE PROBLEM STATUS CPU LANG
TEST CASES
8154922 10:05:02 A Vicious Pikeman (Easy) Accepted 0.06 s Python 3
Submission contains 1 file: download zip archive
FILENAME FILESIZE SHA-1 SUM
viciousPikeman.py 1329 bytes d7665f0efe45bb25bf223455f32bdcd87190c8fb download
Help
dit and resubmit this submission.


open.kattis.com/submissions/8154922
Apps
beth
GD
F
G
m
B
N
In
Other bookmarks
Reading list
viciousPikeman.py
'''
Ejercicio 4 A Vicious Pikeman
Autor: Fabiola Tapara Quispe
Descripcion: El problema nos pide encontrar la cantidad maxima de ejercicios que tiene que resolver un concursante en menos o igual tiempo del que se le da para cada ejercicio, y la penalizacion si no lo entrega a tiempo
usar % 1000000007
'''
def viciousPikeman():
#ingreso de datos
N,T = map(int,input().split())
A,B,C,t0 = map(int,input().split()) # lista con los tiempos que tomara resolver cada ejercicio al Concursante
t = [t0] #primer valor de la lista t
for i in range(1,N):#itera llenando la lista con los tiempos de cada ejercicio
t.append(((A*t[-1]+B) % C) + 1)
t = sorted(t) # sorted ordnea el arreglo
#Se imprime la cantidad de ejercicios resueltos y su penalizacion
ex = 0 #cantidad de ejercicios resueltos
tp = 0 #tiempo de penalizacion por ejercicio
p = 0 #tiempo total de penalizacion
for i in range(len(t)):
#Se analiza si el time es mayor al tiempo maximo dado en el concurso
if tp + t[i] <= T:
tp += t[i]
p = (p + tp) % 1000000007 #nuevo valor para la penalidad recibida
ex+=1#Agrega el tiempo al total
else:
break
print(ex," ",p)#devuelve la cantidad maxima de ejercicios resueltos y la penalidad
viciousPikeman()

Ejercicio 5


open.kattis.com/submissions/8155851
Apps beth GD F G m B B N In Other bookmarks Reading list
Kattis
PROBLEMS CONTESTS RANKLISTS JOBS (5) HELP
Search Kattis
Submit
FABIOLA GRISSEL TAP...
Score: 7.7, Rank: 56970
Submission
ID DATE PROBLEM STATUS CPU LANG
TEST CASES
8155851 12:21:13 Watering Grass Accepted 0.31 s Python 3
Submission contains 1 file: download zip archive
FILENAME FILESIZE SHA-1 SUM
wateringGrass.py 2309 bytes 7bab65e5d256664f3d8b634a79a9afb56c8a01c3 download
Help
dit and resubmit this submission.




```python
'''
Ejercicio 5 Watering Grass
Autor: Fabiola Tapara Quispe
Descripcion: Se pide calcular la cantidad minima de aspersores usados para el riego de una franja de pasto
'''
import sys
from math import sqrt
def main():
    lines = 0  # contador de lineas en 0
    for line in sys.stdin:# Por cada linea enviada al input iteraremos
        if lines == 0: # Para la primera linea del input se guardan los valores
            #w= ancho, l= largo
            n, l, w = map(int, line.split()) # se cambio input por line dado que se lee toda la linea
            aspersores = []
            lines = n #Se establece como n var aux
            medioCuadrado = (w / 2) ** 2 #area elevada al cuadrado de la mitad del ancho
        else:
            lines -= 1 # Se reduce en una unidad cada vez que registremos los datos de un aspersor
            h, r = map(int, line.split())# se lee los datos horizontales de toda la linea y su zona de riego
            if (2 * r) > w:# Si el tamanio del aspersor es mayor
                d = sqrt((r ** 2) - medioCuadrado)#distancia de rango de riego
                aspersores.append((h - d, h + d))# Agregamos su area de riego a la lista de aspersores
            if lines == 0:#Se ordena con sorted para elegir mejor
                aspersores = sorted(aspersores)
                print(AspersoresUse(aspersores, l, len(aspersores))) # Imprimimos los aspersores que necesitamos
def AspersoresUse(aspersores, l, n):
    aspersorUse, aspersorActual, lengthActual = 0, 0, 0
    # aspersorUse: Aspersores usados
    # lengthActual : largo del cesped regado
    while True:
        masLejos = -1 #masLejos es el valor de la mayor disntacia usada seria -1
        while aspersorActual < n and aspersores[aspersorActual][0] <= lengthActual:
        # mientras aun haya aspersores y la longitud del aspersor actual
        # no sea mayor a la longitud ya regada
            masLejos = max(masLejos, aspersores[aspersorActual][1])
            aspersorActual += 1
        if masLejos == -1:
            return -1 # se retornara para evitar pasarnos deltiempo computacional
        aspersorUse += 1
```