Concurso Estructuras de Datos 1 2024/2025

Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

Clasificación de los problemas

Problema	Categoría
A - RomanoX	Strings, Condicionales, ¿Mapas?
B - Pillando tramposos	Listas
C - Codazos en el metro	Pilas.
D - Montayo por favor!!	Monotonic Queues, Pilas, Long.
E - Cuántas complejidades!	Condicionales.
F - Los Problemas de la Tecnología	Conjuntos, Long Long.
G - Menú Amigo	Listas/Pilas.

Estadísticas

Problema	# casos de prueba	Espacio en disco
A - RomanoX	95	1.4 KB
B - Pillando tramposos	10	9 MB
C - Codazos en el metro	5	58 MB
D - Montayo por favor!!	8	12 MB
E - Cuántas complejidades!	3	3.5 KB
F - Los Problemas de la Tecnología	9	7 MB
G - Menú Amigo	9	10 MB
- **Total**	139	(+-) 96 MB

Estadísticas*

Problema	Primer equipo en resolverlo	Tiempo
A - RomanoX	PyPals	8
B - Pillando tramposos	¿?	¿?
C - Codazos en el metro	BBC	40
D - Montayo por favor!!	¿?	¿?
E - Cuántas complejidades!	hap.py	4
F - Los Problemas de la Tecnología	Equipo webo	12
G - Menú Amigo	Equipo webo	53

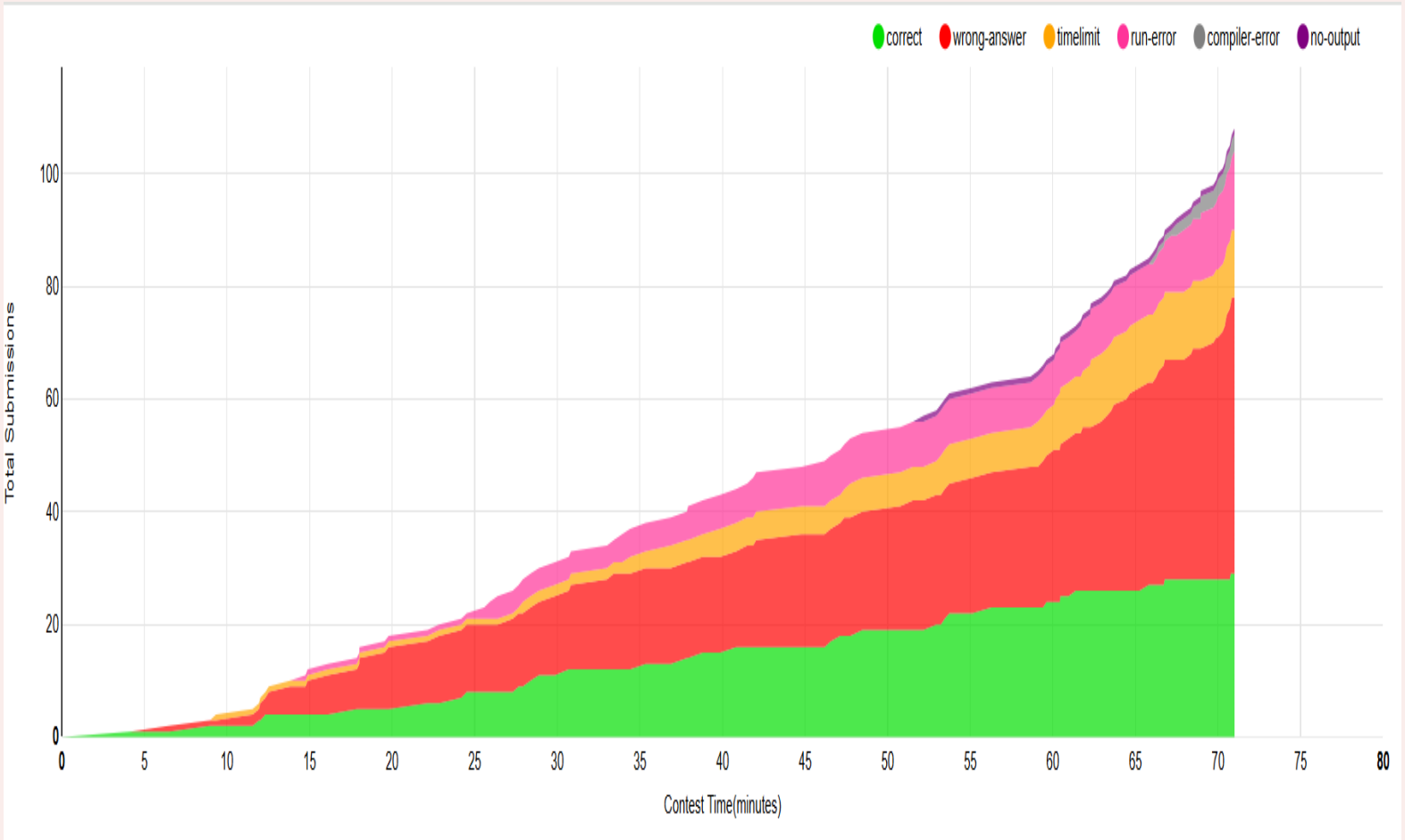
*Antes de congelar el marcador.

Estadísticas*

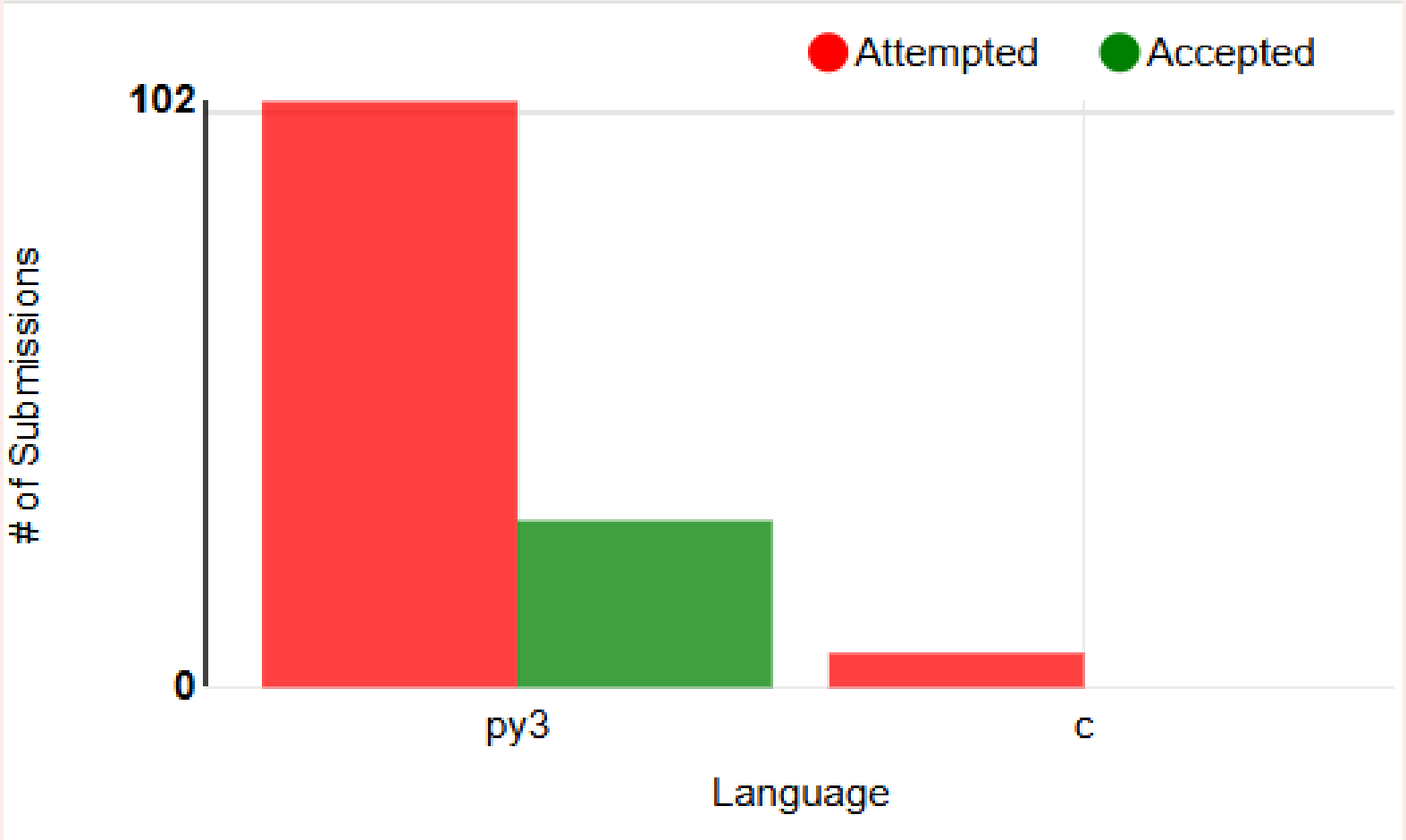
Problema	Válidos	Envíos	% éxito
A - RomanoX	9	21	43 %
B - Pillando tramposos	0	2	0 %
C - Codazos en el metro	1	2	50 %
D - Montayo por favor!!	0	3	0 %
E - Envíos prioritarios	10	16	62 %
F - Los Problemas de la Tecnología	2	11	18 %
G - Menú Amigo	2	12	16 %

*Antes de congelar el marcador.

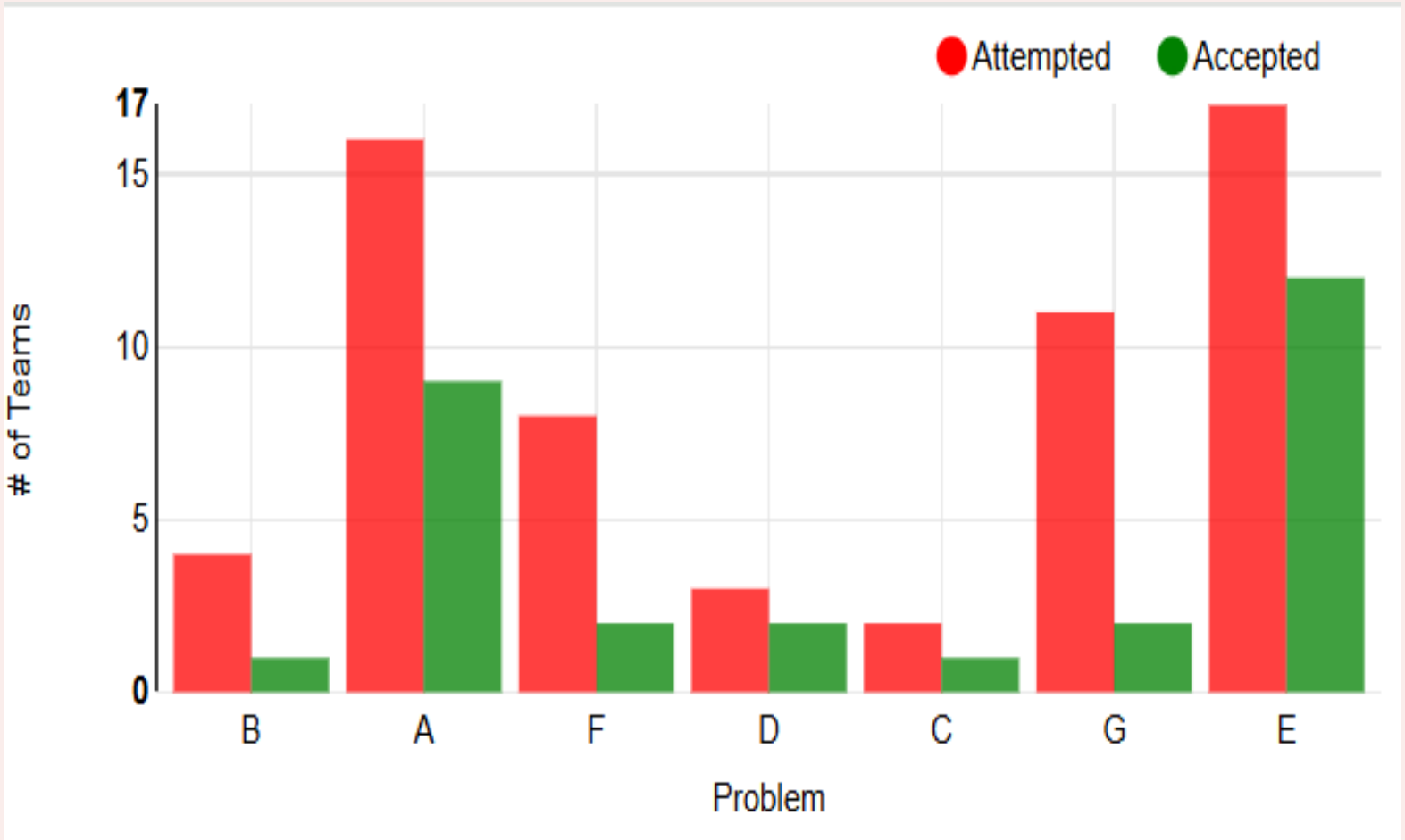
Estadísticas varias



Estadísticas varias



Estadísticas varias



A. RomanoX

Envíos	Válidos	% éxito
21	9	43 %

A. RomanoX

Nos dan una única cadena de caracteres S representando un número romano. Para traducir el número romano a arábigo, hay que ir sumando **o restando** los caracteres de esta cadena.

I	V	X	L	C	D	M
1	5	10	50	100	500	1000

Hay que tener en cuenta que si un carácter de mayor valor precede a uno de menor valor, el primero resta su valor al segundo: IX = 9.

A. RomanoX

```
X, i = 0, 0
while i < len(s):
    s1 = value(s[i])

    if i + 1 < len(s):
        s2 = value(s[i + 1])
        if s1 >= s2:
            X += s1
        else:
            X += (s2 - s1)
            i += 1
    else:
        X += s1
    i += 1

print(X)
```

A. RomanoX

```
def value(r):
    if r == 'I': return 1
    if r == 'V': return 5
    if r == 'X': return 10
    if r == 'L': return 50
    if r == 'C': return 100
    if r == 'D': return 500
    if r == 'M': return 1000
    return -1
```

A. RomanoX

¡Introducción a Estructuras de Datos II!

Mapas

```
value = {
    'I': 1,
    'V': 5,
    'X': 10,
    'L': 50,
    'C': 100,
    'D': 500,
    'M': 1000
}
```

B. Pillando tramposos

Envíos	Válidos	% éxito
2	0	0 %

B. Pillando tramposos

Tenemos que pillar a aquellos usuarios que están utilizando inteligencia artificial para hacer los problemas. Para ello, hemos estimado el tiempo que se tarda en realizar cada uno de los problemas que se han hecho durante el curso, divididos por concursos.

- Cada caso de prueba es un concurso, donde para cada uno:
  - Se da el número de problemas N del concurso y el número de alumnos M.
  - Tiempo estimado de cada problema i del concurso.
  - Tiempo de usuario j en resolver el problema i.

B. Pillando tramposos

7	10	6	15	30	47	26
1	8	3	12	23	11	24
5	20	19	32	54	53	26
20	10	13	16	24	39	37

B. Pillando tramposos

7	10	6	15	30	47	26	
1	8	3	12	23	11	24	7/7
5	20	19	32	54	53	26	
20	10	13	16	24	39	37	

B. Pillando tramposos

7	10	6	15	30	47	26	
1	8	3	12	23	11	24	7/7
5	20	19	32	54	53	26	0/7
20	10	13	16	24	39	37	

B. Pillando tramposos

7	10	6	15	30	47	26	
1	8	3	12	23	11	24	7/7
5	20	19	32	54	53	26	0/7
20	10	13	16	24	39	37	1/7

B. Pillando tramposos

7	10	6	15	30	47	26	*Salida*
1	8	3	12	23	11	24	Usa IA
5	20	19	32	54	53	26	Le sabe
20	10	13	16	24	39	37	Sospechoso

B. Pillando tramposos

Para cada caso de prueba:

```
N, M = map(int, input().split())
estimados = list(map(int, input().split()))
for _ in range(M):
    tiempos = list(map(int, input().split()))
    problemas_con_ia = 0
    for i in range(N):
        if tiempos[i] < estimados[i]: problemas_con_ia += 1
    if problemas_con_ia == N: print('Usa IA')
    elif problemas_con_ia == 0: print('Le sabe')
    else: print('Sospechoso')
print('-----')
```

C. Codazos en el metro

Envíos	Válidos	% éxito
2	1	50 %

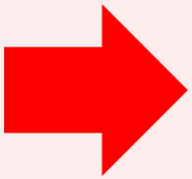
C. Codazos en el metro

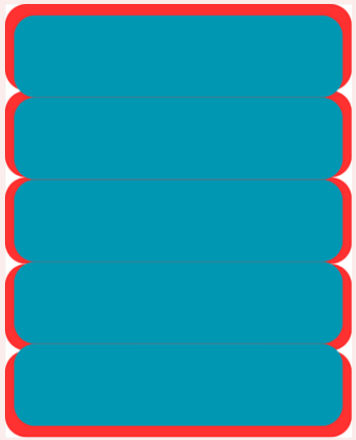
Quieres saber si ha habido algún problema en el metro. La única manera de que no haya problemas es que los pasajeros se bajen en el orden contrario al que han subido, es decir, que el último en subir sea el primero en bajar.

C. Codazos en el metro

último en subir, primero en bajar

Last In, First Out (**LIFO**)





C. Codazos en el metro

Idea:

- Al subir, apilarlos.
- Al bajar, desapilarlos y comprobar que el que baja es el que estaba en la cima.

C. Codazos en el metro

```
leer T
para cada caso:
    leer N
    pila = []
    codazo = false
    en cada parada:
        # bajada
        por p que baje:
            si p == pila.cima:
                pila.desapilar()
            si no:
                codazo = true
        # subida
        por cada p que suba:
            pila.apilar(p)
    si codazo:
        imprimir("CODAZO")
    si no:
        imprimir("BIEN")
```

D. Montayo por favor!!

Envíos	Válidos	% éxito
3	0	0 %

D. Montayo por favor!!

El enunciado nos habla de que Montayo tiene que averiguar la distancia de cada villa a la más próxima que tenga una altura mayor, siendo 0 para la más alta de ellas.

En la entrada nos indican el número de villas y la altura de cada una, estando situadas en línea recta y con una distancia de 1 kilómetro entre cada una de ellas.

Este problema se puede solucionar usando una estructura llamada cola monotónica, que se puede implementar con una pila:

- Se guardan elementos en la pila de forma que siempre se encuentren en orden ascendiente o descendiente, y se eliminan los elementos que no cumplan la condición.
- En este problema podemos almacenar los índices de los elementos con valor mayor al que estamos comprobando, de forma que el último elemento de la pila sea el más cercano.
- Para asegurarnos de coger siempre el valor óptimo debemos comprobarlos en ambos sentidos: comprobamos las distancias de izquierda a derecha y viceversa y seleccionamos el valor mínimo.

D. Montayo por favor!!

Los métodos para conseguir la lista de distancias recorriendo la villa en valor ascendiente y descendiente quedarían como:

```
def getLargestClosest(lista, range_):
    stack = deque()
    # Lista de índices del más próximo que sea mayor
    largestClosest = [-1] * len(lista)
    # Range corresponde a la longitud de la lista de villas, podemos
        hacerlo en rango ascendiente o descendiente
    for i in range_:
        # Si el último elemento de la pila no es mayor se quita
        while stack and lista[stack[-1]] <= lista[i]:
            stack.pop()
        # Si la pila no está vacía significa que el último elemento
            contiene el índice del mayor más cercano
        if stack:
            largestClosest[i] = stack[-1]
        # Metemos en la pila el elemento que se acaba de evaluar
        stack.append(i)
    return largestClosest
```

E. Cuántas complejidades!

Envíos	Válidos	% éxito
16	10	62 %

E. Cuántas complejidades!

Complejidad	Operaciones por segundo
`O(1)`	:D
`O(n)`	$1,000,000$
`O(n*logn)`	$100,000$
`O(n^2)`	1000
`O(n^3)`	100
`O(2^n)`	20
`O(n!)`	12

E. Cuántas complejidades!

Si el número de operaciones por segundo del programa N es menor o igual que el limite de operaciones por segundo para $O(f(n))$, y mayor que el siguiente nivel de complejidad en la tabla, entonces se imprime que el programa tiene complejidad $O(f(n))$.

E. Cuántas complejidades!

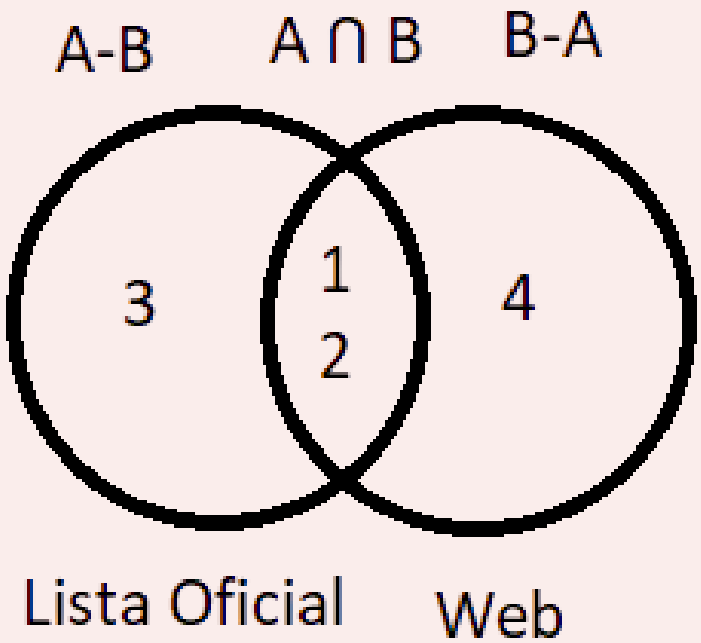
```
leer Casos
for i in Casos
    leer N
    if (N > 1000000)                    :   imprimir "O(1)"
    else if (100000 < N <= 1000000)     :   imprimir "O(n)"
    else if (1000 < N <= 100000)        :   imprimir "O(n*logn)"
    else if (100 < N <= 1000)           :   imprimir "O(n^2)"
    else if (20 < N <= 100)             :   imprimir "O(n^3)"
    else if (12 < N <= 20)              :   imprimir "O(2^n)"
    else if (N <= 12)                   :   imprimir "O(n!)"
```

F. Los problemas de la tecnología

Envíos	Válidos	% éxito
11	2	18 %

F. Los problemas de la tecnología

Dado dos listados de números A y B representado mediante conjuntos, devuelve la diferencia de A menos B ($A \setminus B$).



F. Los problemas de la tecnología

Errores comunes.

- No usar Long Long Int -> WA
- No devolver elementos ordenados -> WA
- Mostrar los elementos de B que no están en A (3 1 2 3 1 2 4 imprimir 3 4) -> WA

Posibles ideas...

Aproximación	Tiempo	Espacio	Veredicto
Array y recorrer por cada premio en web	$O(P + P^2)$	$O(P)$	TLE
2 Arrays ordenados y recorrer	$O(2 \cdot (P + PlogP) + P)$	$O(P + P)$	AC
Mapa sumando y restando	$O(P + P + P)$	$O(P)$	AC
Set añadiendo y eliminando	$O(P + P + P)$	$O(P)$	AC

Siendo P el número de premios.

F. Los problemas de la tecnología

El algoritmo quedaría como:

```
int P
long long int P_i
set<long long int> S
read P
for i in P
    read P_i
    S.insert(P_i);
for i in P
    read P_i
    S.erase(P_i);
bool flag = true,correct=true
for s in S
    if flag imprimir s
    else imprimir " " + s
    correct=false;
    flag=false;
if correct
    imprimir "OK"
```

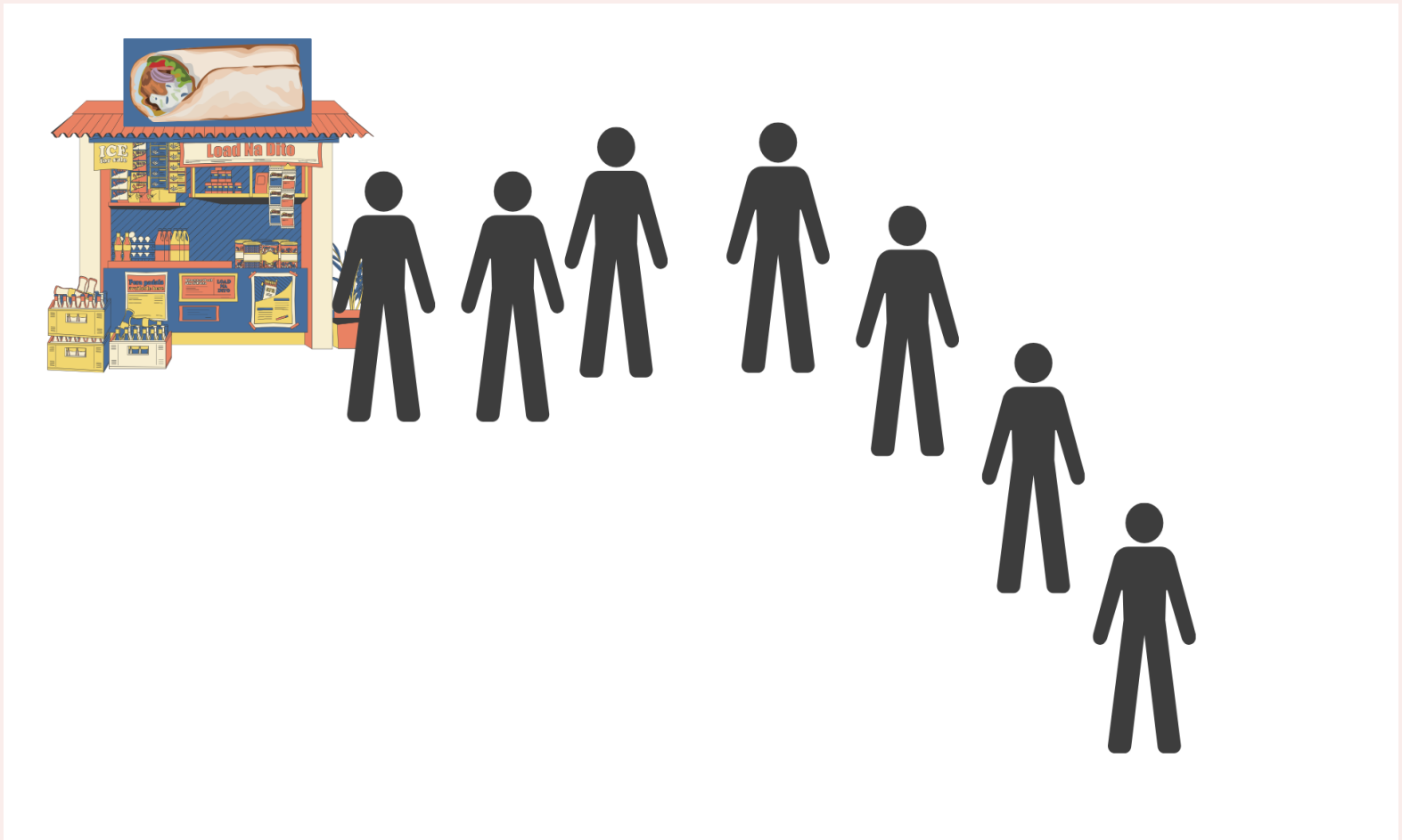
G. Menú Amigo

Envíos	Válidos	% éxito
12	2	16 %

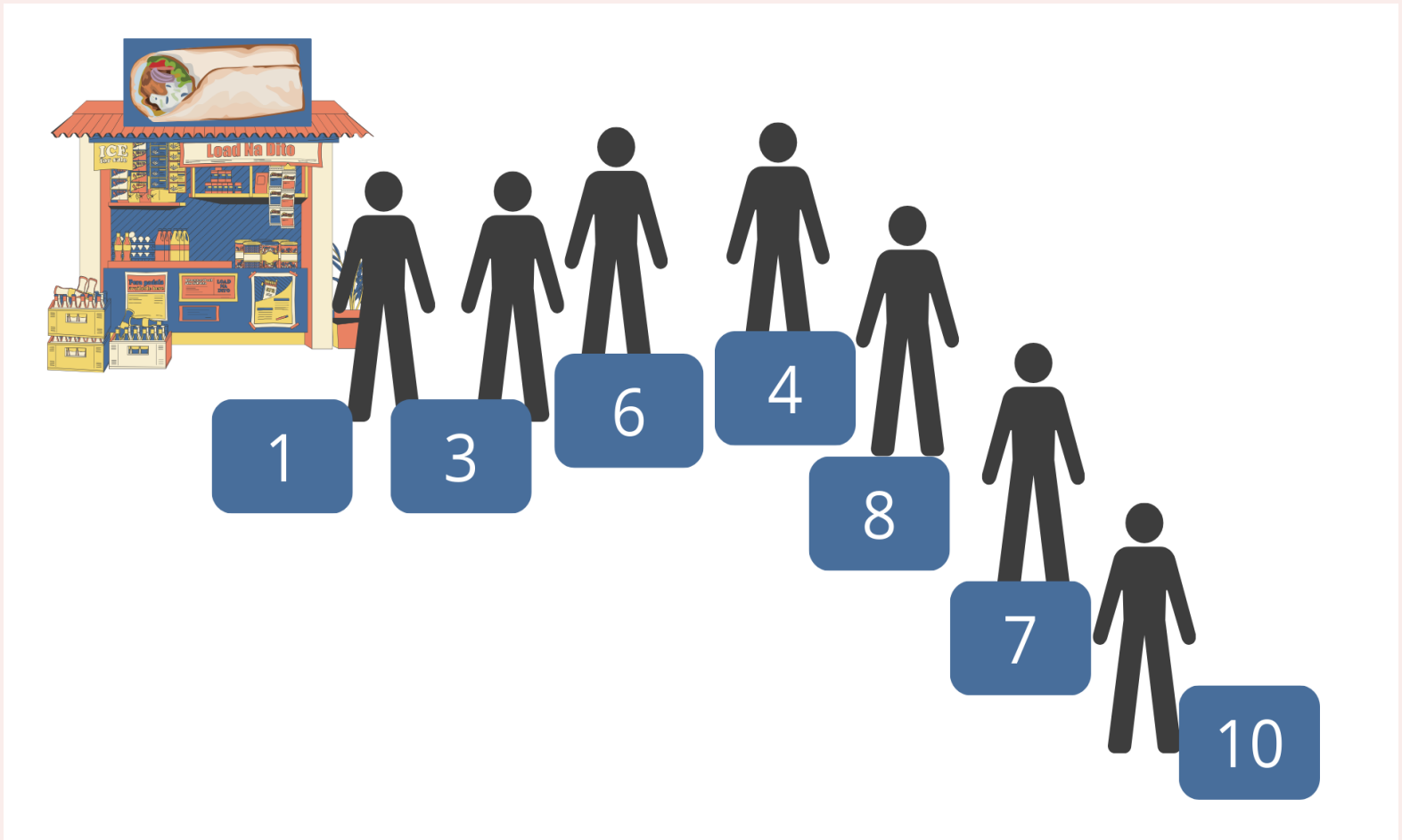
G. Menú Amigo

Cuánta gente se ha colado en la fila?

G. Menú Amigo



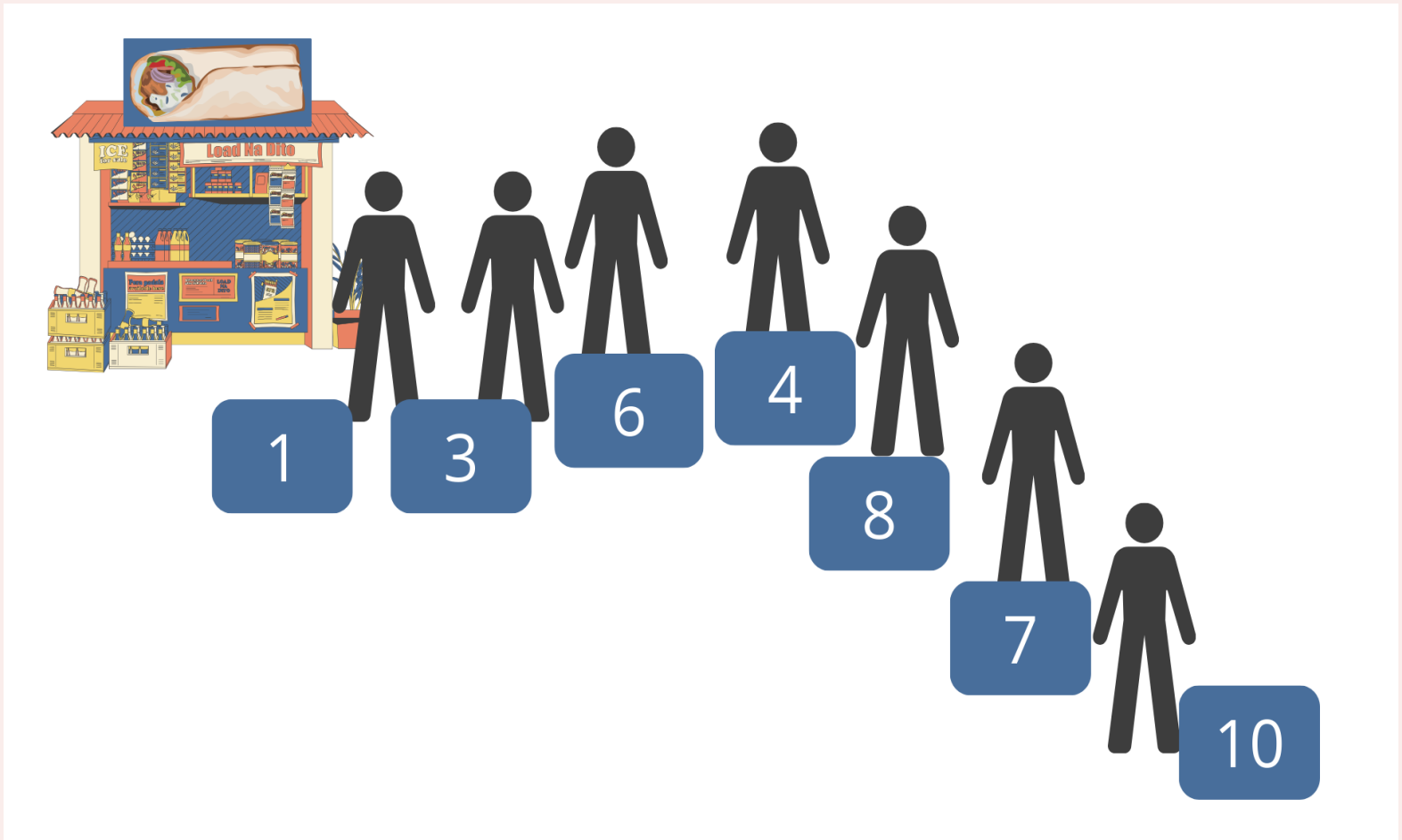
G. Menú Amigo



G. Menú Amigo

Idea: Recorrer la cola de atrás alante, comprobando si el número de cada persona es mayor que el de cualquiera detrás de ella.

G. Menú Amigo



G. Menú Amigo

Esto es lo mismo que comprobar si el número de cada persona es mayor al mínimo de los números detrás de ella.

G. Menú Amigo

```
int n;
int[] numeros;

int colaos=0;
min=numeros[n-1];
for(int i=n-2; i>=0; i--) {
    if(numeros[i]<min) min=numeros[i];
    else colaos++;
}
print(colaos)
```