# Clasificatorio AdaByron URJC 2026

## Estadísticas y Soluciones

URJC Escuela Técnica Superior de Ingeniería Informática

# Clasificación de los problemas

| Problema | Categoría |
|---|---|
| A - URJC Dates | Ad-Hoc, ordenación. |
| B - Asignación de salas en congresos | Recursividad, backtracking. |
| C - ¿Cuántos números? | Matemáticas, pensar. |
| D - Raíces felices | Ad-Hoc. |
| E - Mesita de noche | Pilas. |

# Estadísticas

| Problema | # casos de prueba | Espacio en disco |
|---|---|---|
| A - URJC Dates | 5 | 1 KB |
| B - Asignación de salas en congresos | 13 | 131 KB |
| C - ¿Cuántos números? | 40023 | 750 KB |
| D - Raíces felices | 8 | 1 KB |
| E - Mesita de noche | 12 | 56 KB |
| - **Total** | 40061 | 939 KB (+-) |

# Estadísticas*

| Problema | Primer equipo en resolverlo | Tiempo |
|---|---|---|
| A - URJC Dates | 3SomeProgramming | 17 |
| B - Asignación de salas en congresos | Hap.py | 34 |
| C - ¿Cuántos números? | Hap.py | 17 |
| D - Raíces felices | HeapHunters | 5 |
| E - Mesita de noche | Hap.py | 6 |

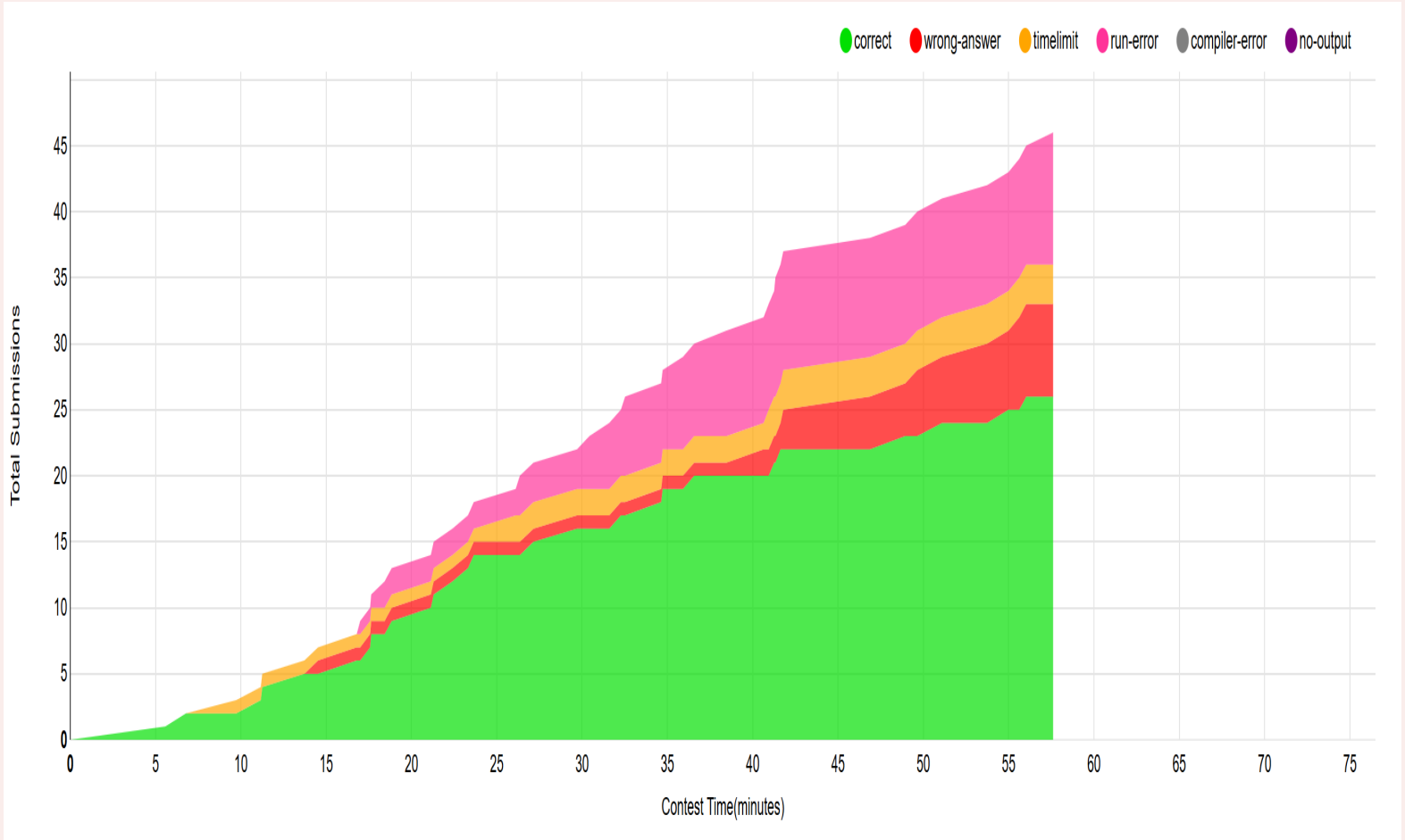
*Antes de congelar el marcador.

# Estadísticas*

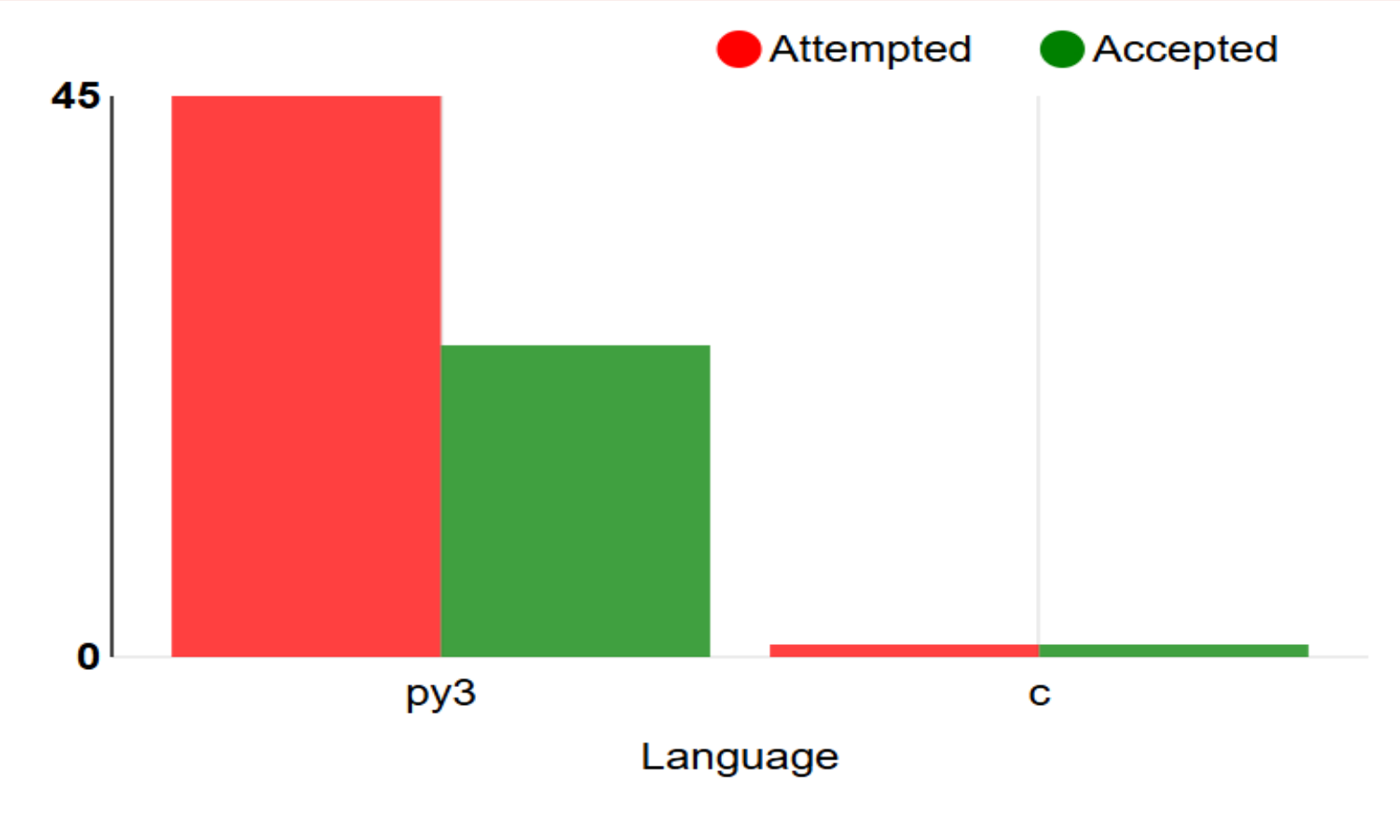
| Problema | Envíos | Válidos | % éxito |
|---|---|---|---|
| A - URJC Dates | 6 | 4 | 66 % |
| B - Asignación de salas en congresos | 7 | 2 | 28 % |
| C - ¿Cuántos números? | 10 | 3 | 30 % |
| D - Raíces felices | 7 | 7 | 100 % |
| E - Mesita de noche | 6 | 6 | 100 % |

*Antes de congelar el marcador.

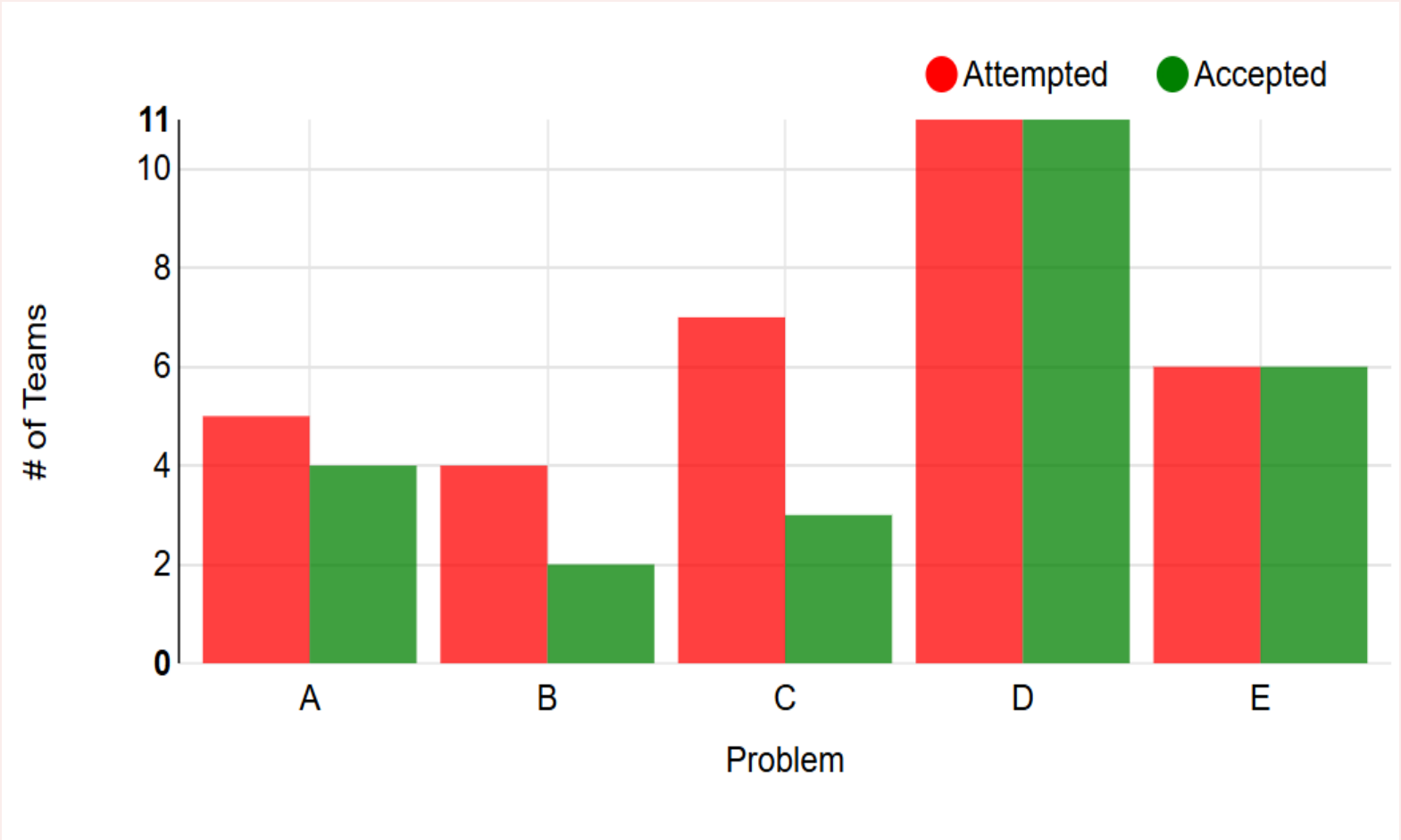
# Estadísticas varias



# Estadísticas varias



# Estadísticas varias



# A. URJC Dates

| Envíos | Válidos | % éxito |
|---|---|---|
| 5 | 4 | 80 % |

# A. URJC Dates

Nos piden minimizar la diferencia en valor absoluto entre dos listas.

# A. URJC Dates

Nos piden minimizar la diferencia en valor absoluto entre dos listas.

- Ordenar ambas listas.
- Calcular la diferencia componente a componente.

# A. URJC Dates

```
leer(N)
int[] a,b
for i in range(N)
    leer(nombre)
    leer(nota)
    a[i] = nota
for i in range(N)
    leer(nombre)
    leer(nota)
    b[i] = nota
sort(a)
sort(b)
sol = 0;
for i in range(N)
    sol += abs(a[i]-b[i]);
print(sol)
```

# B. Asignación de salas en congresos

| Envíos | Válidos | % éxito |
|---|---|---|
| 6 | 2 | 33 % |

# B. Asignación de salas en congresos

**Datos de entrada:**

- $N$ presentaciones con duración conocida
- $M$ salas disponibles
- Límite de tiempo $K$ por sala

**Objetivo:**
Asignar cada presentación a una sala sin que ninguna supere $K$ minutos.

**Tipo de problema:**
Problema de **decisión**, no buscamos la mejor asignación, solo si *existe alguna válida*.

**Ejemplo:**

- Duraciones: `[30, 60, 45, 20]`
- $M = 2$ salas, $K = 90$ min

**Asignación válida:**

- Sala 1 $\rightarrow$ `[30, 60]` $= 90$ OK
- Sala 2 $\rightarrow$ `[45, 20]` $= 65$ OK

**Restriccion:**
Todas las presentaciones deben ser asignadas.

# B. Asignación de salas en congresos

**Un algoritmo voraz puro no es suficiente:**

- Asignar cada presentación a la sala con *más tiempo libre* no garantiza solución global, aunque parezca óptimo en cada paso.

**Caso ilustrativo:**

- $N = 10$, $M = 4$, $K = 12$
- Duraciones: `[12, 4, 2, 7, 4, 4, 6, 4, 3, 2]`
- Greedy falla: asigna 12 a la sala 1 y luego no encuentra hueco para 7 + cualquier otra
- Solución válida **existe**:

| | |
|---|---|
| Sala 1 | `12` |
| Sala 2 | `7, 4` |
| Sala 3 | `6, 4, 2` |
| Sala 4 | `4, 4, 3, 2` |

# B. Asignación de salas en congresos

La solución es probar todas las combinaciones posibles de asignar N presentaciones a M salas. Esto genera un árbol de decisiones de tamaño $M^N$.

La poda clave es $t\_usado[sala] + duracion[p] <= K$ cortamos ramas inviables antes de explorarlas, reduciendo drásticamente el espacio de búsqueda frente a la fuerza bruta pura.

También había casos base que contemplar con soluciones por defecto válidas o inválidas ¿no cabe por duración de tiempo? ¿sumando todos los tiempos exceden? ¿hay pocas actividades?

# B. Asignación de salas en congresos

```
1  def backtrack(p_actual, N, M, K, duracion, t_usado,
       asignacion):
2      # Caso base: todas las presentaciones asignadas
3      if p_actual == N:
4          return True
5      # Intentar asignar a cada sala
6      for sala in range(M):
7          # Verificar que la sala tenga tiempo disponible
8          if t_usado[sala] + duracion[p_actual] <= K:
9              # Asignar sala
10             t_usado[sala] += duracion[p_actual]
11             asignacion[p_actual] = sala
12             # Recursion
13             if backtrack(p_actual + 1, N, M, K, duracion,
                   t_usado, asignacion):
14                 return True
15             # Backtrack
16             t_usado[sala] -= duracion[p_actual]
17             asignacion[p_actual] = -1
18     return False
```

## C. ¿Cuántos números?

| Envíos | Válidos | % éxito |
|---|---|---|
| 9 | 2 | 22 % |

# C. ¿Cuántos números?

¿Cuántos dígitos tiene el número $X$?

Observemos algunos casos sencillos:

| Número | Dígitos | Rango |
|---|---|---|
| $1-9$ | $1$ | $10^0 \leq X < 10^1$ |
| $10-99$ | $2$ | $10^1 \leq X < 10^2$ |
| $100-999$ | $3$ | $10^2 \leq X < 10^3$ |
| $\vdots$ | $\vdots$ | $\vdots$ |
| $10^{k-1}-10^k-1$ | $k$ | $10^{k-1} \leq X < 10^k$ |

Un número $X$ tiene $k$ dígitos cuando:

$$10^{k-1} \leq X < 10^k$$

Despejando $k$, llegamos a:

$$k = \lfloor \log_{10}(X) \rfloor + 1$$

# C. ¿Cuántos números?

Partimos de que $X$ tiene $k$ dígitos:

$$10^{k-1} \leq X < 10^k$$

Aplicamos $\log_{10}$ a toda la expresión (función creciente, conserva desigualdades):

$$k - 1 \leq \log_{10}(X) < k$$

Esto significa exactamente que:

$$\lfloor \log_{10}(X) \rfloor = k - 1$$

Despejando $k$:

$$\boxed{k = \lfloor \log_{10}(X) \rfloor + 1}$$

# C. ¿Cuántos números?

Podemos calcular el resultado de la siguiente manera:

```
1  import math
2
3  T = int(input())
4  for _ in range(T):
5      n = int(input())
6      print(math.floor(n * math.log10(2)) + 1)
```

# D. Raíces felices

| Envíos | Válidos | % éxito |
|---|---|---|
| 7 | 7 | 100 % |

# D. Raíces felices

El problema sencillo del concurso!

Solo había que imprimir los números del 1 al $n$.

Fijaos que todos los números naturales cumplían la condición: Para el 1 el 1, para el 2 el 4, para el 3 el 9...

# E. Mesita de noche

| Envíos | Válidos | % éxito |
|---|---|---|
| 6 | 6 | 100 % |

# E. Mesita de noche

Tenemos una mesita de noche.

1. En la mesita se van añadiendo libros, por encima.
2. Vamos leyendo los libros y quitándoles de la mesita de noche.

# E. Mesita de noche

Tenemos una mesita de noche.

1. En la mesita se van añadiendo libros, por encima.
2. Vamos leyendo los libros y quitándoles de la mesita de noche.

Lo que tenemos en la mesita de noche es una **pila** de libros.

# E. Mesita de noche

```
leer N
mesita = deque()

for i in range(N):
    leer ni
    si ni == 1:
        # Nuevo libro
        leer ti
        mesita.append(ti)
    si no:
        # Leemos algunos libros
        leer li
        for _ in range(li):
            mesita.pop()

M = len(mesita)
imprimir M
for _ in range(M):
    imprimir mesita.pop()
```