Algoritmia Básica

CURSO DE PROGRAMACIÓN COMPETITIVA URJC - 2026

Organizadores:

- Isaac Lozano (isaac.lozano@urjc.es)
- Sergio Salazar (sergio.salazar@urjc.es)
- Adaya Ruiz (am.ruiz.2020@alumnos.urjc.es)
- **Olga Somalo** (o.somalo.2021@alumnos.urjc.es)
- Lucas Martín (lucas.martin@urjc.es)
- Iván Penedo (ivan.penedo@urjc.es)
- **Alicia Pina** (alicia.pina@urjc.es)
- Sara García (sara.garciar@urjc.es)
- Raúl Fauste (raul.fauste@urjc.es)

Contenidos

- Divide y vencerás
  -Algoritmos de Ordenación
  -Búsqueda binaria
- Algoritmos Voraces
- Técnicas algorítmicas

Contenidos

- **Divide y vencerás**

  -Algoritmos de Ordenación

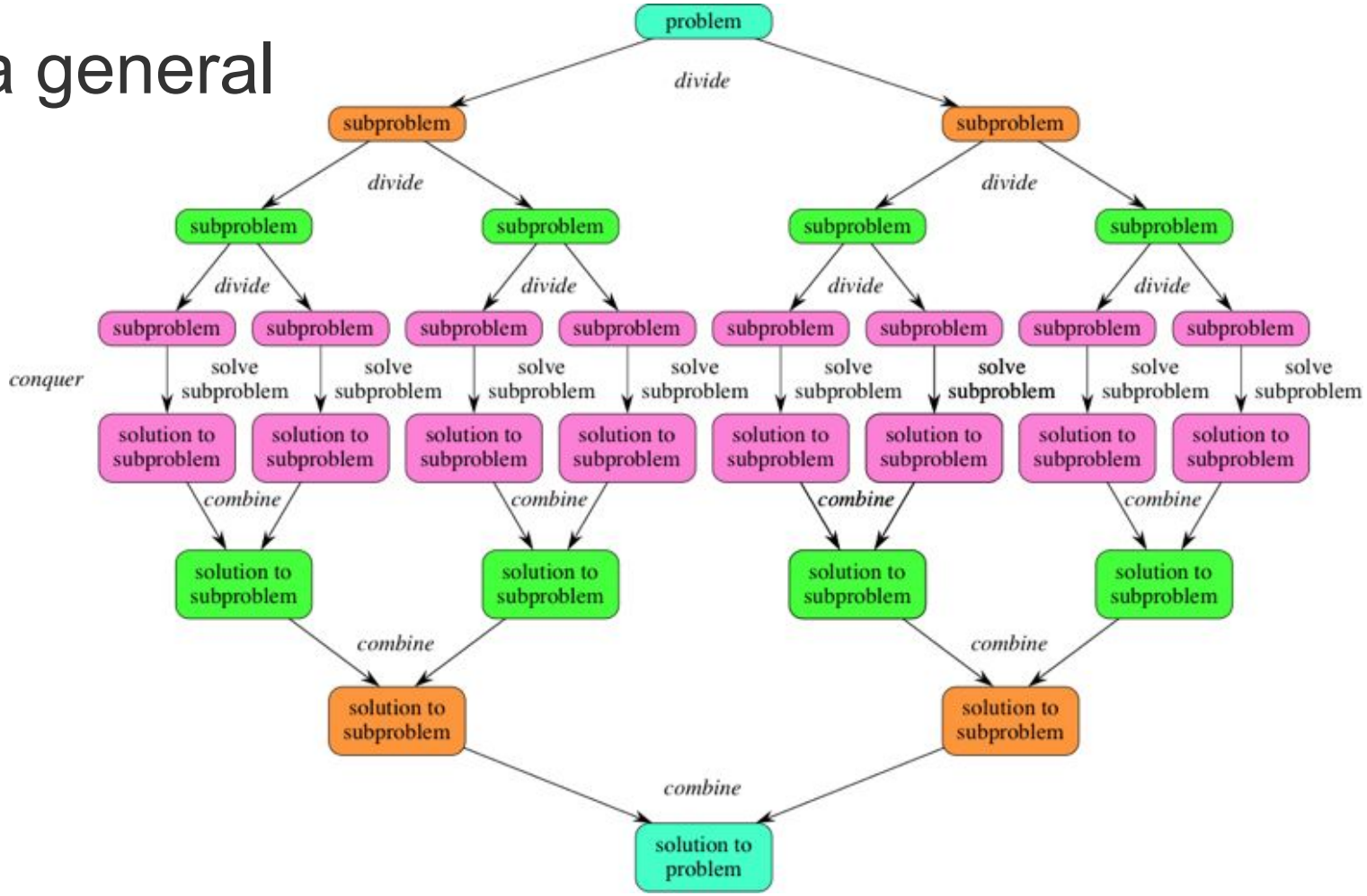
  -Búsqueda binaria
- Algoritmos Voraces
- Técnicas algorítmicas

Divide Y Vencerás

- La idea intuitiva es la separación del problema en problemas más pequeños.
- Así, la solución global está compuesta de la combinación de subsoluciones.

Divide Y Vencerás

Esquema general



Divide Y Vencerás

- Quicksort
- Mergesort
- Búsqueda binaria

Los tres son ejemplos de esta aproximación...

Algoritmos de Ordenamiento

Algunos problemas requieren tener ordenados una serie de elementos para dar una respuesta

- Algoritmos de ordenación
- Estructuras de datos ordenadas

… es importante su eficiencia

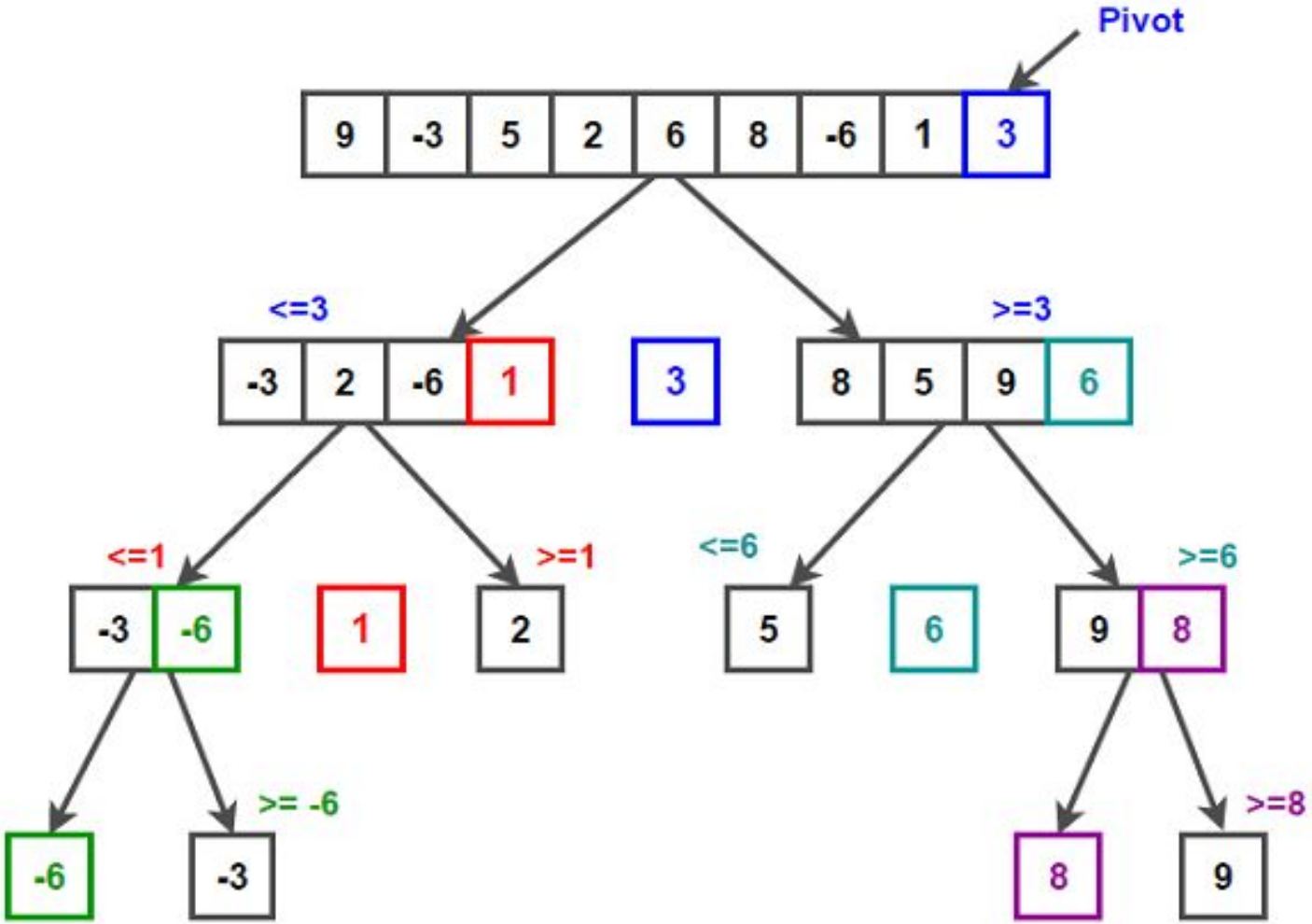
Algoritmos de Ordenamiento

- Quicksort / Mergesort
  - Complejidad: O(nLog(n))
  - Ya están implementadas en las bibliotecas básicas ¡No hay que programarlos!

Algoritmos de Ordenamiento

- Quicksort (idea básica)
  - Se toma un pivote “al azar” del array (un elemento cualquiera del array)
  - Dos arrays mantienen los elementos menores y mayores al pivote
  - Recursivamente se tratan estos dos arrays por separado y se concatena su resultado
  - Se repite el proceso hasta que quede 1

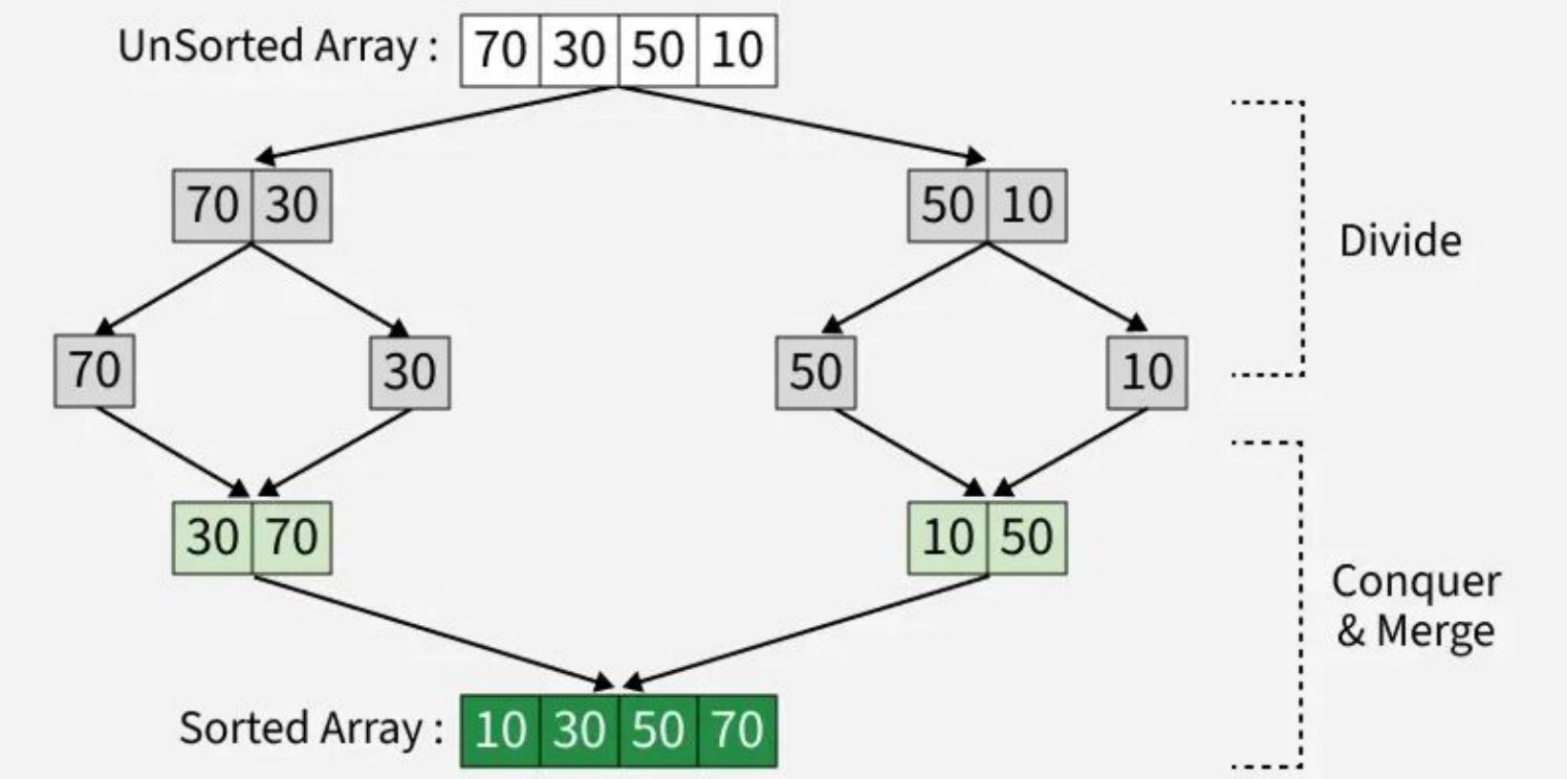
Algoritmos de Ordenamiento



Algoritmos de Ordenamiento

- MergeSort (idea básica)
  - Se llama recursivamente combinando los arrays desde 0 hasta N/2 y de N/2 hasta N
  - Si el elemento tiene 1 elemento, se asume que está ordenado
  - Formar un nuevo array tomando en cuenta los dos arrays formados

Algoritmos de Ordenamiento



Algoritmos de Ordenamiento

- C++
  - std:sort()
- Java
  - Collections.sort() -> MergeSort*
  - Arrays.sort() -> Quicksort* (para tipos primitivos)

*En realidad son variantes más eficientes

Algoritmos de Ordenamiento

- Estructuras ordenadas nos permiten hacer búsquedas “inteligentes” sobre ellas (búsqueda binaria, ternaria, etc)

- Otros algoritmos de ordenamiento

Problemas

https://open.kattis.com/problems/akcija

```
a= [5,2,8,3,1]
a.sort()
print(a)
```

```
>> a = [1,2,3,5,8]
```

```
a= [5,2,8,3,1]
a.sort(reverse=True)
print(a)
```

```
>> a = [8,5,3,2,1]
```

https://www.spoj.com/problems/SBANK/

Búsqueda Binaria

- Definición
- Donde aplicar
- Ejemplo

Búsqueda Binaria

- Se utiliza cuando un problema contiene una función f monótona creciente o decreciente
- Una función es monótona creciente si para cualquier x, y con x < y tal que f(x) <= f(y). Es monótona decreciente en el caso contrario

Búsqueda Binaria

- La idea detrás del algoritmo es ir descartando mitades donde sabemos que no podemos encontrar la respuesta
- Ej. Si queremos encontrar un valor determinado

Búsqueda Binaria

- Acotamos la función para un x mínimo y máximo dentro de f(x)
- Por cada iteración, sea mid = (Cotamin + Cotasup) / 2
- Si f(mid) > Obj, significa que nos hemos pasado, por lo tanto, Cotasup = mid
- Si f(mid) <= Obj, signifiica que hemos subestimado, por lo tanto, Cotainf = mid

Búsqueda Binaria

- La respuesta final estará en Cotainf

```
function binary_search(f, inf, sup, t):
  while sup-inf > 1
    mid = (sup+inf) / 2
    if f(mid) <= t:
      inf = mid
    else
      sup = mid
  return inf
```

Problemas

https://codeforces.com/edu/course/2/lesson/6/1/practice/contest/283911/problem/A

https://codeforces.com/edu/course/2/lesson/6/1/practice/contest/283911/problem/B

Contenidos

- Divide y vencerás
  -Algoritmos de Ordenación
  -Búsqueda binaria
- **Algoritmos Voraces**
- Técnicas algorítmicas

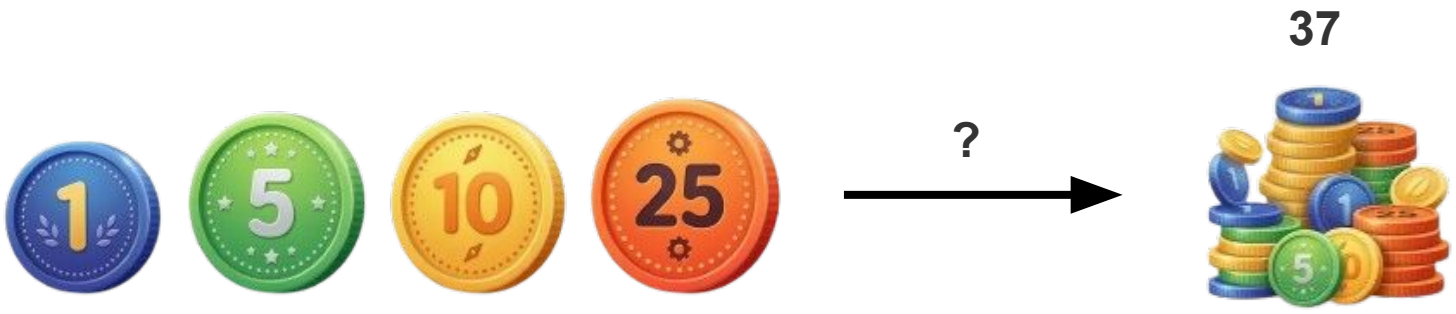
Algoritmos Voraces

Motivación: el problema de la moneda

Algoritmos Voraces

Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**



Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**

1+1+10+25

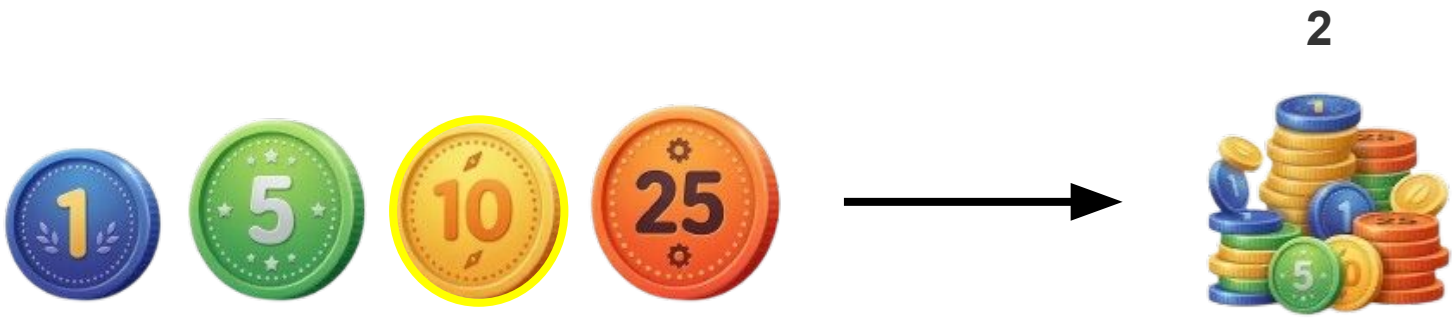
37

Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**

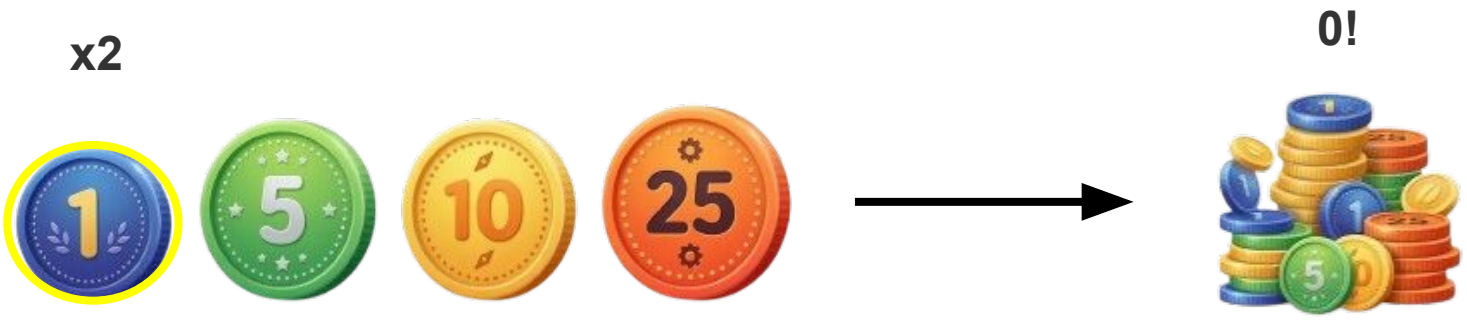
Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**



Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**



VORA

Algori[illegible]nc[illegible]V[illegible]aces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**

Algoritmos Voraces

¿Qué es un algoritmo voraz?

Algoritmos Voraces

¿Qué es un algoritmo voraz?

Un algoritmo voraz (o **greedy)** construye una solución tomando en cada paso la **mejor decisión local posible**, sin reconsiderar decisiones anteriores.

Algoritmos Voraces

Hit the Lottery

https://codeforces.com/problemset/problem/996/A

Algoritmos Voraces

Hit the Lottery

```
n = int(input())

  solu = 0
  valores = [100, 20, 10, 5, 1]

  for billete in valores:
      solu += n // billete  # Monedas que utilizamos
      n %= billete          # Dinero restantes

  print(solu)
```

Algoritmos Voraces

Shopaholic

https://open.kattis.com/contests/m4qvtu/problems/shopaholic

Algoritmos Voraces

Shopaholic

```
# Leemos toda la entrada de golpe
input_data = sys.stdin.read().split()
n = int(input_data[0])
p = [int(x) for x in input_data[1:]]

p.sort(reverse=True) # Ordenamos de mayor a menor
discount = sum(p[2::3])# Sumamos desde el índice 2, saltando de 3 en 3

print(discount)
```

Algoritmos Voraces

Cuidado!!!

Greedy no siempre funciona

Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**

Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**

Algoritmos Voraces

Objetivo: devolver el **cambio exacto** pedido con el **menor** número de **monedas**

6+6!

~~10+1+1~~?

12

Algoritmos Voraces

Elegir siempre la mejor decisión local no garantiza obtener la mejor solución global!!

Un enfoque greedy suele funcionar cuando:

- elegir ahora no empeora las opciones futuras
- el problema tiene estructura óptima
- podemos ordenar o priorizar decisiones

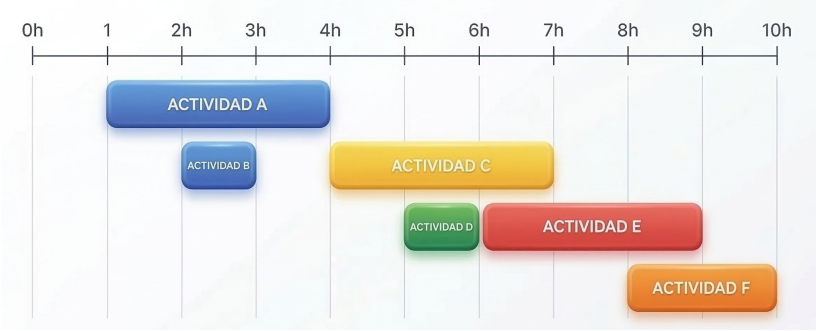
Algoritmos Voraces

Problemas clásicos con solución **voraz**

Algoritmos Voraces

Selección de tareas

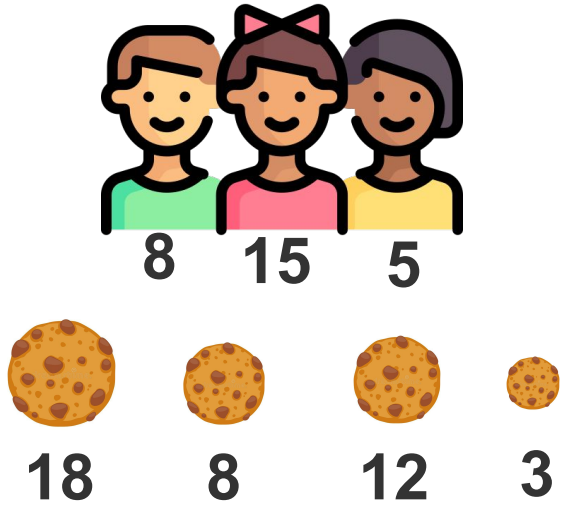
Objetivo: realizar el mayor número de tareas, sin que éstas se superpongan



https://open.kattis.com/problems/intervalscheduling

Asignar galletas

Objetivo: asignar el máximo número de galletas, sabiendo que cada niño exige un tamaño mínimo (y no repetirá galleta)



https://www.geeksforgeeks.org/problems/assign-cookies/1

Contenidos

- Divide y vencerás

  -Algoritmos de Ordenación

  -Búsqueda binaria
- Algoritmos Voraces
- **Técnicas algorítmicas**

Contenidos

El **array de acumulados**

Contenidos

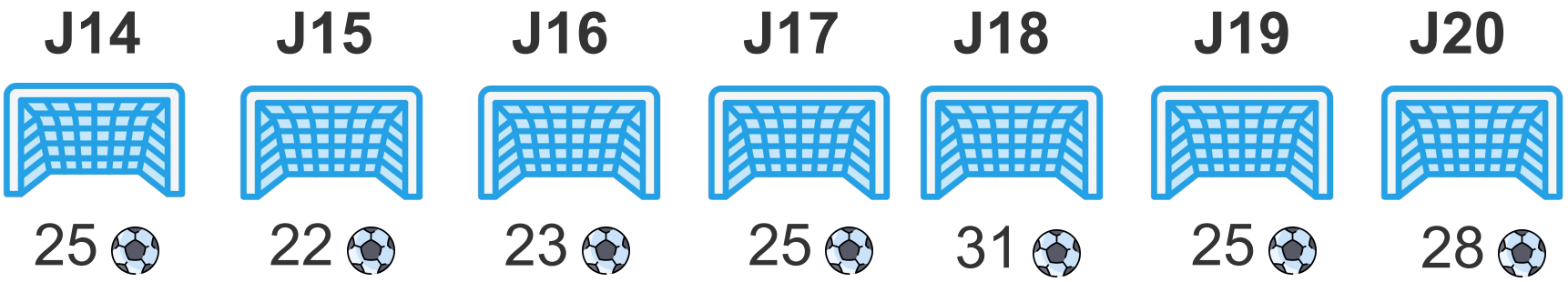
Imaginemos que queremos responder consultas del tipo:

"**suma** en el **rango** [l,r] de un array"

Contenidos

Imaginemos que queremos responder consultas del tipo:

"**suma** en el **rango** [l,r] de un array"



¿Cuántos goles se marcaron entre la jornada 16 y la 19?

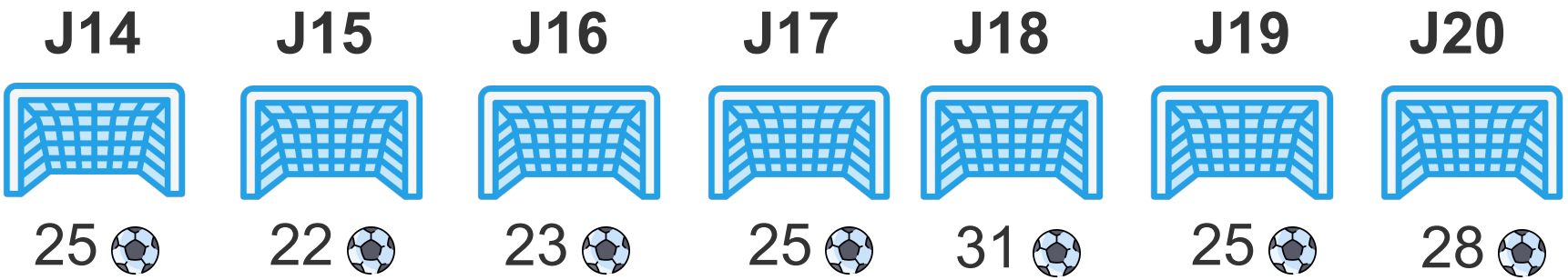
¿Y entre la jornadas 14 y la 17?

…

Contenidos

Imaginemos que queremos responder consultas del tipo:

"**suma** en el **rango** [l,r] de un array"



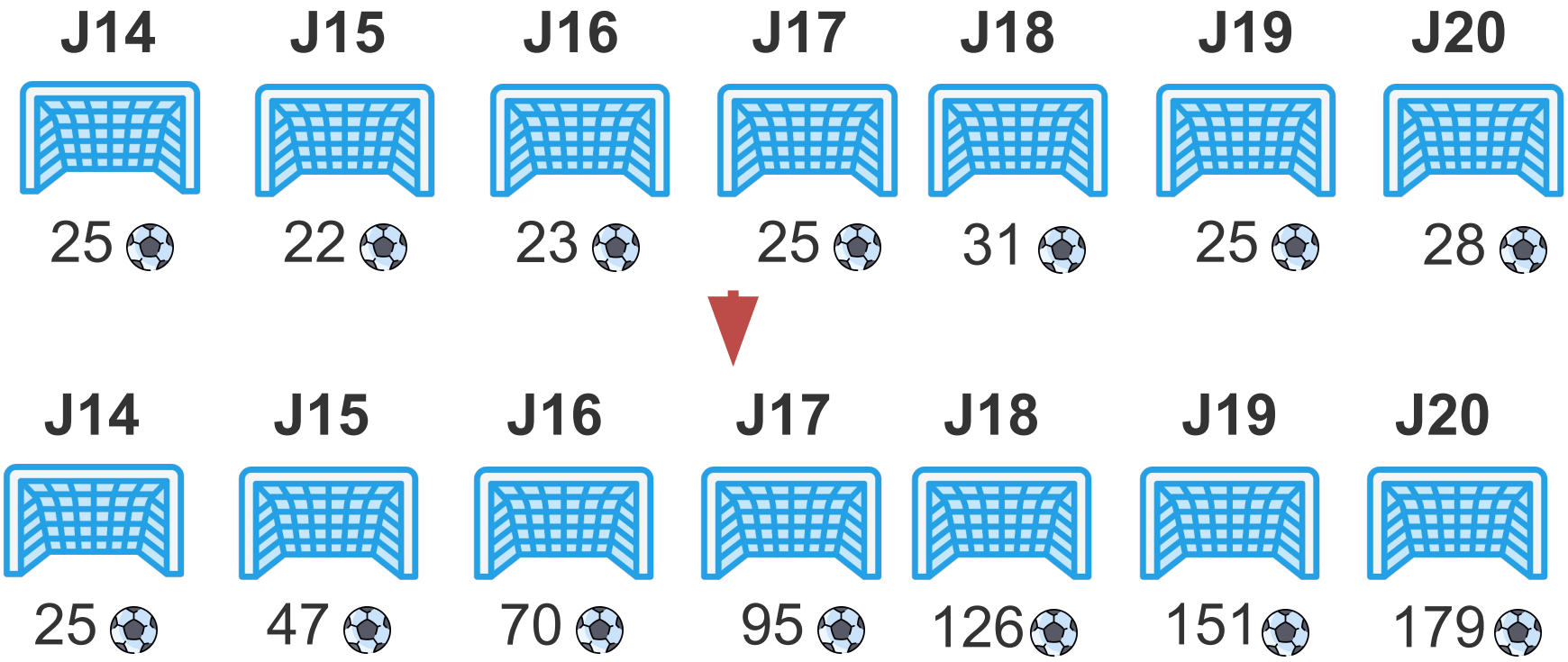
¿Fuerza bruta? → **O(n q)!!** ☠

* q = nº de consultas
* n = tamaño del array

Contenidos

Solución: array de acumulados (prefix sum)

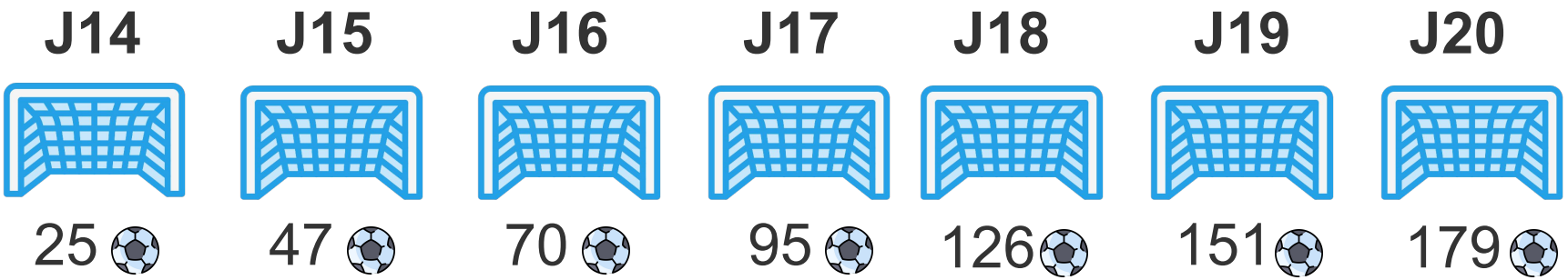
pref[i]=suma de a[0…i]



Contenidos

Solución: array de acumulados (prefix sum)

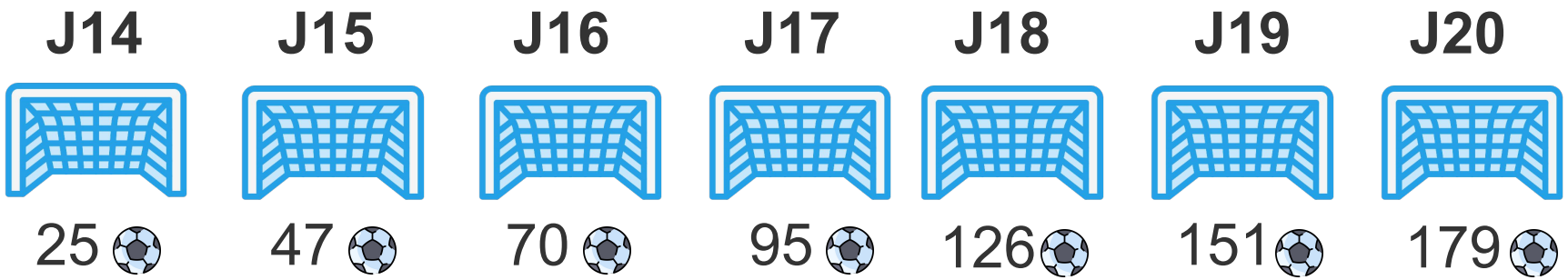
pref[i]=suma de a[0…i]



Contenidos

Solución: array de acumulados (prefix sum)

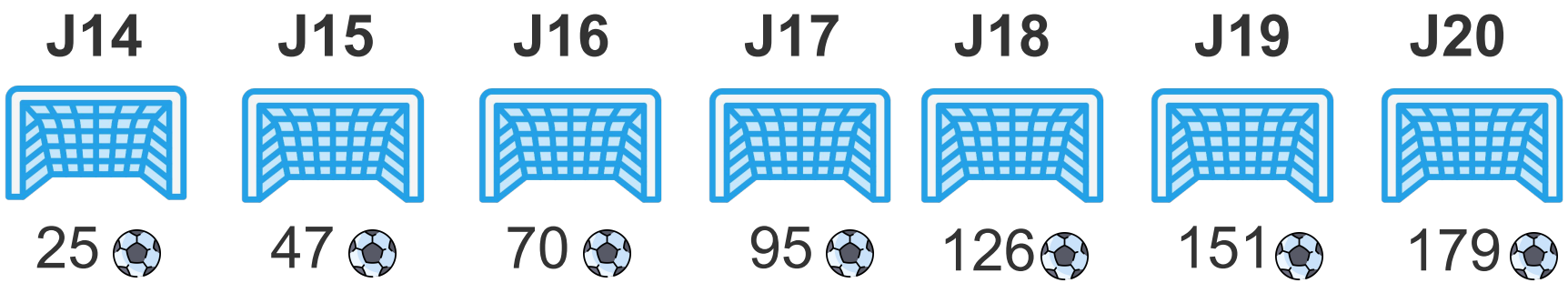
pref[i]=suma de a[0…i]



Contenidos

Solución: array de acumulados (prefix sum)

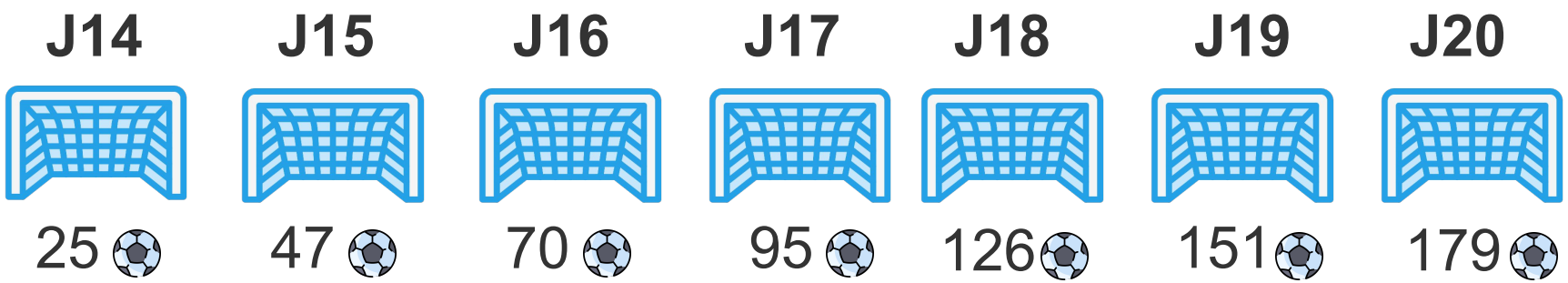
pref[i]=suma de a[0…i]



Contenidos

Solución: array de acumulados (prefix sum)

pref[i]=suma de a[0…i]

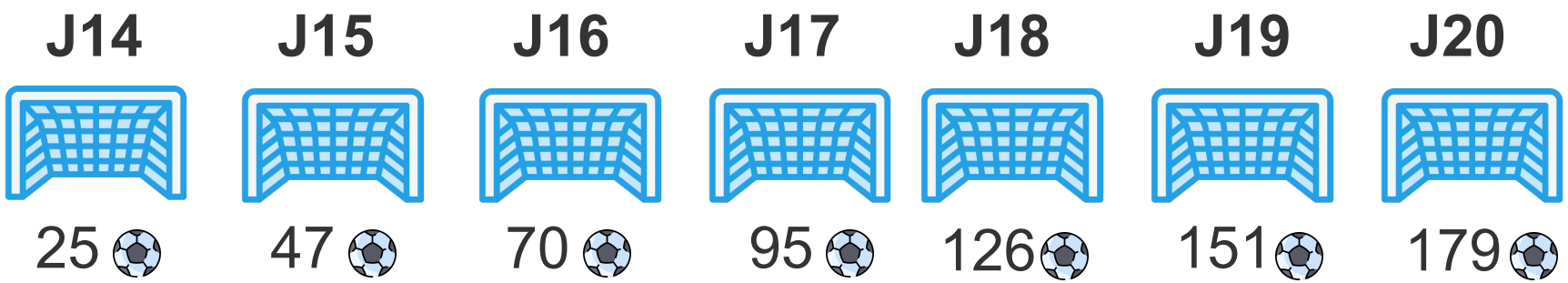


sum(l,r) = pref[r] - pref[l-1]

Contenidos

Solución: array de acumulados (prefix sum)

pref[i]=suma de a[0…i]

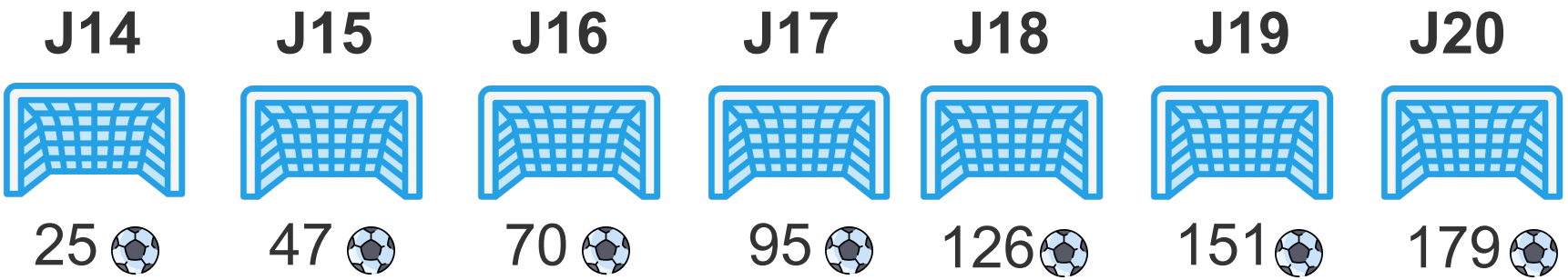


sum(16,18) = 126 – 47 =79

Contenidos

Solución: array de acumulados (prefix sum)

pref[i]=suma de a[0…i]



O(q)!

Algoritmos Voraces

Kuriyama Mirai's Stones

```
# Cálculo del array de acumulados
pref_v = [0]+ list(accumulate(v))
pref_u = [0] + list(accumulate(u))
# Responder a una consulta: tipo,l,r
if tipo == 1:
   print(pref_v[r] - pref_v[l-1])
else:
   print(pref_u[r] - pref_u[l-1])
```

Contenidos

La **ventana deslizante**

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

2 1 3 2 1 1 1

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

2 1 3 2 1 1 1

¿Fuerza bruta?

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

2 1 3 2 1 1 1

¿Fuerza bruta? ⟶ **O(n^2)!!** 

* n = tamaño del array

Contenidos

La **ventaja deslizante** es una técnica para procesar **subarrays dinámicos** de un array principal de manera eficiente

Se representa mediante 2 **índices** o **punteros:**

- 1 señalando el **inicio** de la ventana (izq)
- 1 señalando el **final** de la ventana (sin incluir) (der)

La ventana se **expande moviendo** el puntero **der** para incluir nuevos elementos, y se **contrae moviendo** el puntero **izq** cuando se deja de cumplir una condición (como superar una suma determinada)

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=0, der=1

2 1 3 2 1 1 1

acu=2

max=1

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=0, der=2

2 1 3 2 1 1 1

acu=3

max=2

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=0, der=3

2 1 3 2 1 1 1

acu=6!

max=2

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=1, der=3

2 1 3 2 1 1 1

acu=4

max=2

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=1, der=4

2 1 3 2 1 1 1

acu=6!

max=2

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=2, der=4

2 1 3 2 1 1 1

acu=5

max=2

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=2, der=5

2 1 3 2 1 1 1

acu=6!

max=2

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=3, der=5

2 1 3 2 1 1 1

acu=3

max=2

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=3, der=6

2 1 3 2 1 1 1

acu=4

max=3

Contenidos

Imaginemos que queremos saber cuál es el **subarray más largo** con **suma ≤ k**

Ejemplo: k=5

Ventana deslizante: izq=3, der=7

2 1 3 2 1 1 1

acu=5

max=4

Algoritmos Voraces

Books

https://codeforces.com/problemset/problem/279/B

Algoritmos Voraces

CODEFORCES

Books

```
left, right, current_time,max_books = 0

while right < n:
    current_time += a[right] # Abrir ventana
    right += 1

    while current_time > t:       # Cerrar ventana
        current_time -= a[left]   # (si nos pasamos)
        left += 1

    max_books = max(max_books, right - left)

print(max_books)
```

viene!

@URJC_CP

@Dijkstraidos