Computer Society
Uniandes

Programación Competitiva

Computer Society

Wilmer Arévalo

Tabla de Contenido

Introducción a Greedy

01 **Repaso**

Repaso del tema y ejercicios anteriores

02 **Complejidades**

Complejidad temporal y espacial

03 **Introducción a Greedy**

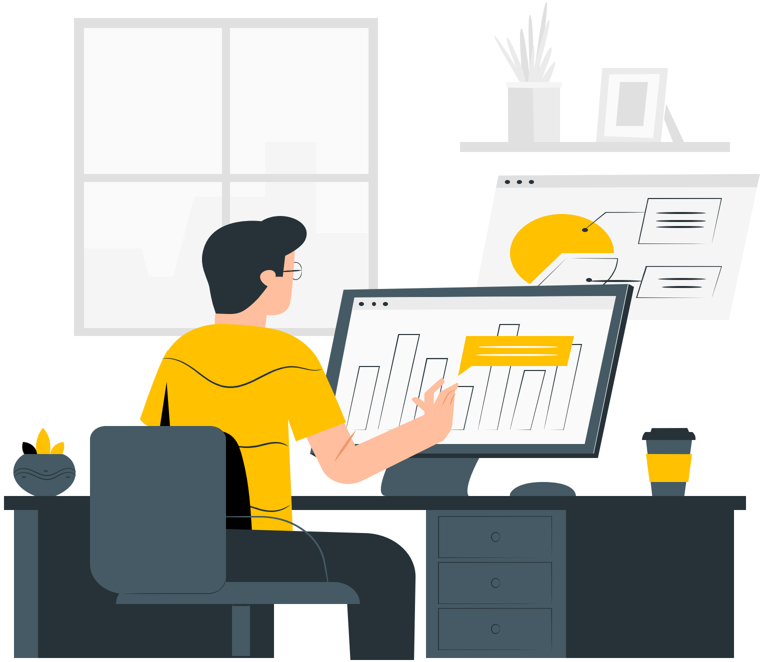
Primer acercamiento a los algoritmos voraces

04 **Problemas Introductorios**

Problemas para entender el Greedy Approach

05 **Más problemas**

Más problemas con aprovechamiento voraz



01

Repaso del Tema Anterior

¿Cómo nos fue con los problemas de la última sesión?

¡Revisemos!

Vamos a revisar un par de problemas de aquellos que les haya parecido los más interesantes

02

Complejidades

¿Cómo podemos estimar la eficiencia de un algoritmo?

Complejidades Algorítmicas

Es una métrica teórica que nos ayuda a describir el **comportamiento** de un algoritmo en términos de:

Tiempo de ejecución (complejidad temporal) y memoria requerida(complejidad espacial)

Permite comparar entre la efectividad de un algoritmo y otro, y decidir cuál es más eficiente.

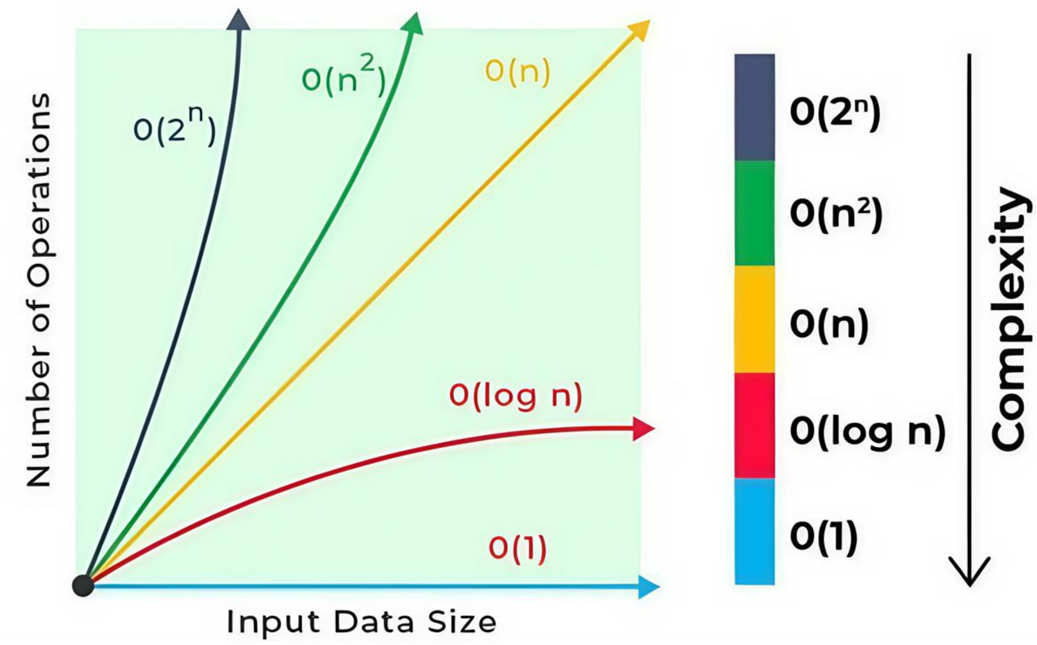
¿Cómo Interpretarla?

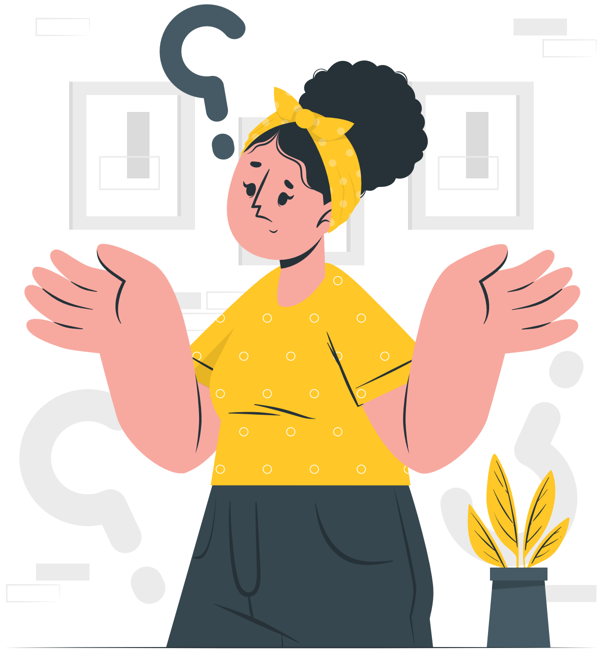
Podemos interpretar la complejidad algorítmica como la **tasa de crecimiento** en:

Tiempo o **Memoria** requerida por un algoritmo para resolver un problema en función del tamaño de su entrada

Nos olvidamos del lenguaje utilizado, el sistema en donde se corra el algoritmo y el estilo implementado.

Notación Big O





¿Cuál Elegir?

```python
def twoSum(array, target):
    for i in range(len(array)):
        for j in range(len(array)):
            if i != j and array[i] + array[j] == target:
                return i, j
    return -1
```

```python
def twoSum(array, target):
    memo = {}
    length = len(array)
    for i in range(length):
        number = array[i]
        complement = target - number
        if complement in memo:
            return memo[complement], i
        memo[number] = i
    return -1
```

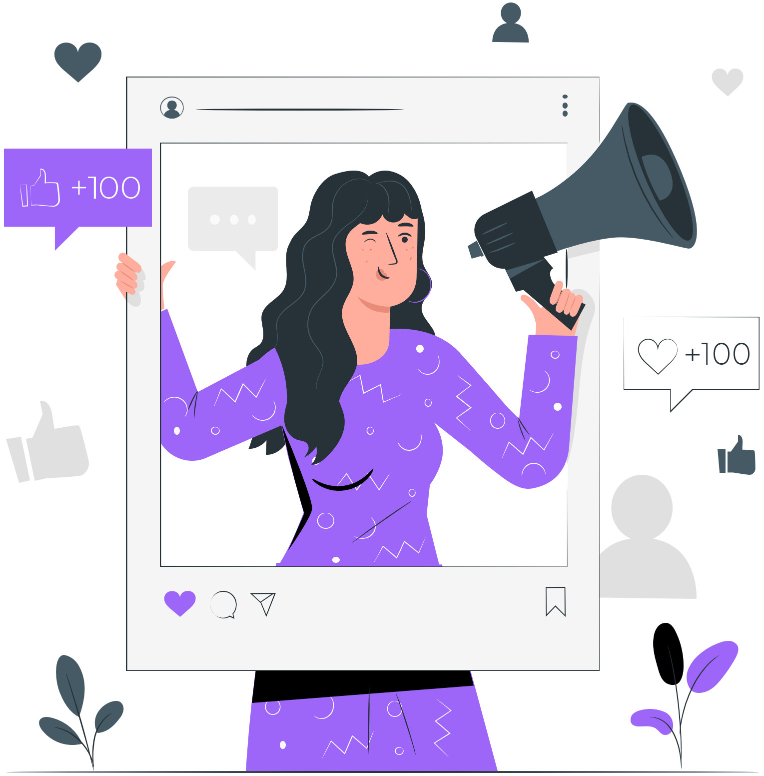
La primera opción tiene menos líneas de código...

Complejidades

Determine la complejidad algorítmica, en notación Big O, y el peor caso posible para los siguientes algoritmos:

```python
def algorithm(n, k):
    q = 0
    r = n
    while r >= k:
        q += 1
        r -= k
    return q, r
```

```python
def algorithm(n):
    b = n >= 2
    i = 2
    while i < n and b:
        b = b and n % i > 0
        i += 1
    return b
```

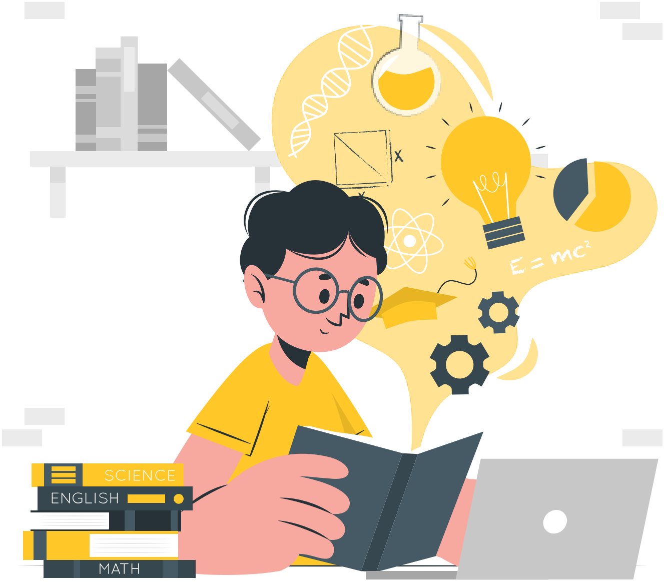


03

Introducción a Greedy

¿Qué son los algoritmos voraces?

Greedy Approach



Enfoque usado en problemas de **optimización**. Es decir, buscamos mínimos o máximos.

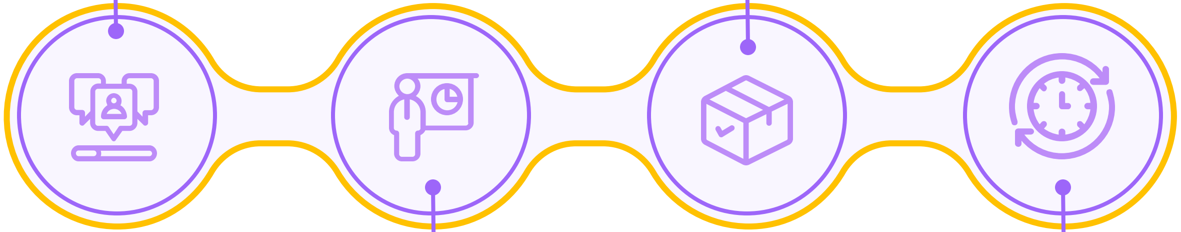
Consiste en tomar siempre la **mejor decisión local** en cada paso, con la esperanza de que esto conduzca a una solución global óptima.

El aprovechamiento Greedy **NO siempre garantiza la solución óptima**, pero se puede usar en muchas situaciones

Estrategia Greedy

Seleccionar
El mejor candidato

Agregar
Candidatos factibles



Comprobar
Si el candidato es factible

Repetir
Hasta completar

Características de los algoritmos Greedy

Rápido

Usualmente tiene complejidades bajas

Decisión local óptima

La decisión más "barata"

No vuelve atrás

Una vez se toma una decisión, no se reconsidera

Subproblemas óptimos

Generan una solución óptima

04

Problemas Introductorios

¡Ahora sí, empecemos!

Maratón de Tareas

Un estudiante universitario dejó todos sus trabajos para el último día y tiene T horas disponibles antes del cierre de entregas. Debe completar la mayor cantidad posible de tareas.

Dada una lista con los n trabajos y el tiempo estimado t_i para cada uno, determina cuántos trabajos puede terminar dentro de las T horas.

3	1	4	2	2	5		10

¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.

Greedy Approach

¿Y si somos voraces? Buscamos la tarea con menos tiempo y la realizamos. Repetir.

Greedy Ordenado

¿Y si ordenamos el arreglo? Vamos ejecutando las tareas en orden de tiempo.

¿Alguna idea?

```python
def marathonTasksBruteForce(tasks, timeLimit):
    from itertools import combinations
    maxTasks = 0
    for r in range(1, len(tasks) + 1):
        for combo in combinations(tasks, r):
            if sum(combo) <= timeLimit:
                maxTasks = max(maxTasks, len(combo))
    return maxTasks
```

Fuerza Bruta

$O(2^n)$

```python
def marathonTasksGreedy(tasks, timeLimit):
    total = 0
    accumulatedTime = 0
    remainingTasks = tasks.copy()
    while remainingTasks:
        minimumTask = min(remainingTasks)
        if accumulatedTime + minimumTask <= timeLimit:
            accumulatedTime += minimumTask
            total += 1
        remainingTasks.remove(minimumTask)
    return total
```

Greedy Approach

$O(n^2)$

```python
def marathonTasksGreedyWithSorting(tasks, timeLimit):
    tasks.sort()
    total = 0
    accumulatedTime = 0
    for task in tasks:
        if accumulatedTime + task <= timeLimit:
            accumulatedTime += task
            total += 1
        else:
            break
    return total
```

Greedy Ordenado

$O(n\log_2 n)$

Coin Change

Dado un conjunto canónico de monedas con denominaciones y un valor objetivo, se debe encontrar el número mínimo de monedas necesarias para llegar al valor exacto.

63

¿Alguna idea?

Fuerza Bruta

La vieja confiable, probar con todas las posibles configuraciones.

Greedy Approach

¿Y si voraces? Buscamos la tarea con menos tiempo y la realizamos. Después, repetimos lo mismo.

Programación Dinámica

¿Y si le damos un enfoque de programación dinámica?

Recuerda que Greedy no siempre garantiza la solución óptima

¿Alguna idea?

```python
def coinChangeBruteForce(coins, amount):
    from itertools import combinations
    minCoins = float('inf')
    for r in range(1, len(coins) + 1):
        a = coins * (amount // min(coins) + 1)
        for combo in combinations(a, r):
            if sum(combo) == amount:
                minCoins = min(minCoins, len(combo))
    return minCoins if minCoins != float('inf') else -1
```

Fuerza Bruta

$O(2^n)$

```python
def coinChangeGreedy(coins, amount):
    coins.sort(reverse=True)
    totalCoins = 0
    remaining = amount
    for coin in coins:
        while remaining >= coin:
            remaining -= coin
            totalCoins += 1
            if remaining == 0:
                return totalCoins
    return -1
```

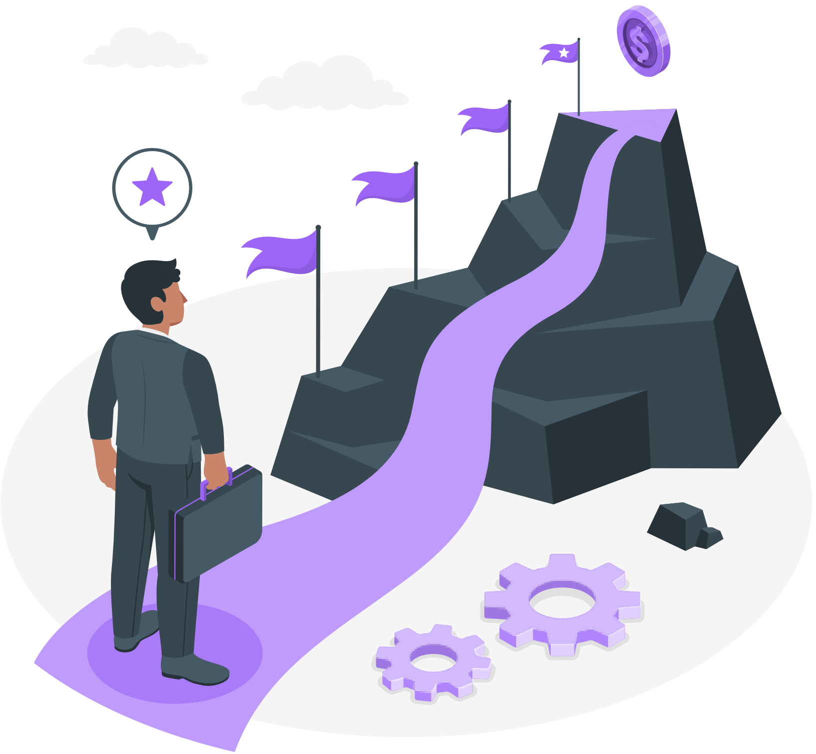
Greedy Approach

$O(n\log_2 n)$

Próximamente

Retomaremos este tema más tarde

Programación Dinámica



05

Más problemas...

¡Pon a prueba tus habilidades!

Minimum Varied Number

El ICPC presenta un reto donde se debe encontrar el número más pequeño posible cuya suma de dígitos sea igual a un valor dado y todos sus dígitos sean únicos. La entrada es un entero que representa la suma deseada de los dígitos y la salida es el número más pequeño con dígitos distintos que sume el valor dado.

20	3	8	9

Santa Claus and Candies

Santa Claus tiene n dulces y sueña con regalarlos a los niños. Quiere dar todos los dulces que tiene, asegurándose de que cada niño reciba una cantidad entera positiva distinta. El reto es encontrar el número máximo de niños a los que puede dar dulces cumpliendo estas condiciones.

Dado un número de dulces, se debe indicar cuántos niños pueden recibir dulces y la cantidad exacta para cada niño. Si existen múltiples soluciones, se puede imprimir cualquiera de ellas.

8		1	2	5

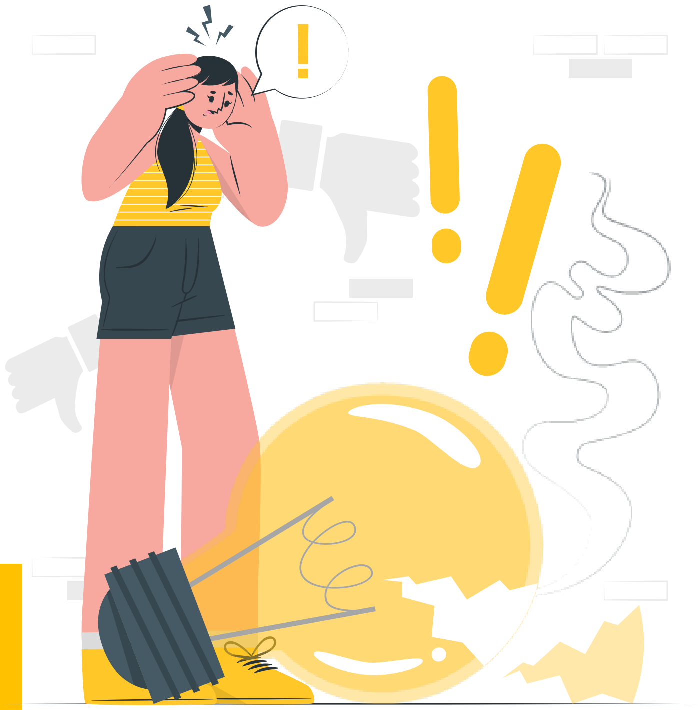
Hard Problem

El profesor Ball tiene un aula con 2 filas y m asientos en cada fila. Hay a monos que solo quieren sentarse en la fila 1, b monos que solo quieren sentarse en la fila 2, y c monos sin preferencia. Cada asiento puede ser ocupado por un solo mono siguiendo las preferencias. El reto es determinar el número máximo de monos que se pueden sentar.

m	a	b	c
10	5	5	10

20

Phone Desktop



Rosie tiene un teléfono con varias pantallas, cada una de tamaño 5 filas por 3 columnas. Quiere colocar **x** aplicaciones con íconos de 1x1 y **y** aplicaciones con íconos de 2x2, usando el menor número posible de pantallas. Cada celda puede contener solo un ícono.

x	y
1	1

1	1	
1	1	
	1	

Maximize Mex

Dado un arreglo de **n** enteros positivos y un entero **x**, se pueden realizar operaciones para incrementar los valores de ciertos elementos en **x** unidades. El objetivo es encontrar el valor máximo del MEX (mínimo entero no presente en el arreglo).

n	x
6	2

6

1	3	4	1	0	2

Recursos Adicionales

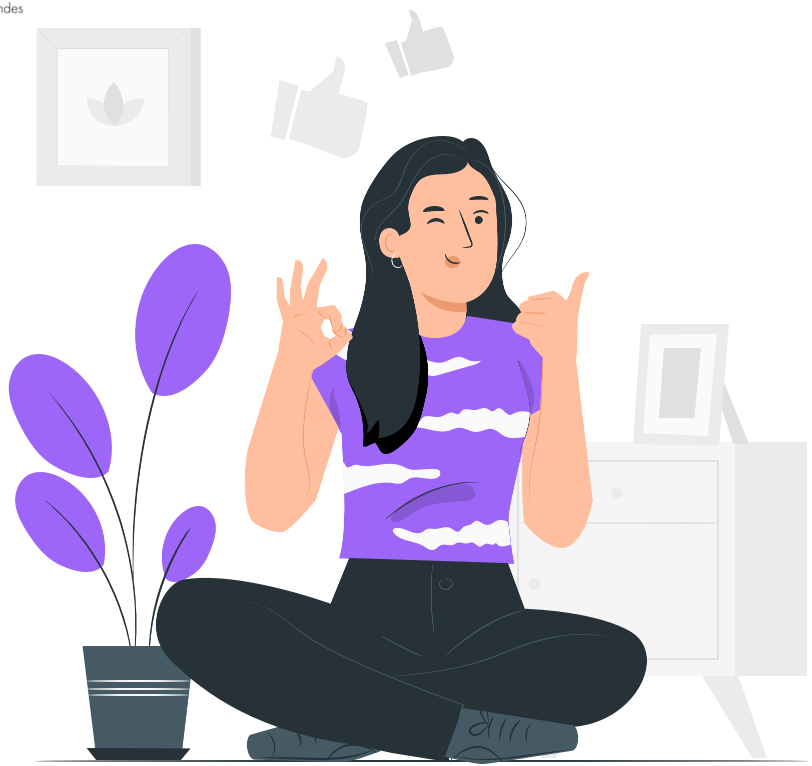
Problema

Problema

Material

Material

Computer Society
Uniandes



¡Gracias!

Computer Society

Wilmer Arévalo

CREDITS: This presentation template was created by Slidesgo, including icons by Flaticon, infographics & images by Freepik and illustrations by Stories