**Objetivo General:**

Aplicar conceptos fundamentales de análisis de algoritmos, incluyendo notación asintótica, estructuras de datos básicas, técnicas como recursión, greedy y programación dinámica para resolver problemas computacionales.

**Ejercicios Básicos (Nivel 1)**

**1. Contar pares que suman un valor K**

**Enunciado:** Dado un arreglo de enteros y un valor K, escribir un algoritmo que cuente cuántos pares distintos de elementos suman exactamente K.

**Entrada:**



**Salida esperada:** 3

**Objetivo:** Comparar soluciones O(n^2) vs O(n).

**Solución sugerida:**

Texto

El contenido generado por IA puede ser incorrecto.

**Verificar si una cadena es un palíndromo**

**Enunciado:** Escribir una función que determine si una cadena es un palíndromo ignorando mayúsculas, espacios y signos.

**Entrada:**



**Salida esperada:** True

**Objetivo:** Trabajar limpieza de cadenas y comparación.

**Solución sugerida:**

Imagen que contiene Texto

El contenido generado por IA puede ser incorrecto.

**Ejercicios Intermedios (Nivel 2)**

**3. Problema de la mochila fraccional (Greedy)**

**Enunciado:** Dado un conjunto de objetos con peso y valor, y una mochila con capacidad máxima W, escribir un algoritmo que maximice el valor total permitiendo tomar fracciones de objetos.

**Objetivo:** Aplicar estrategia greedy ordenando por valor/peso.

**Solución sugerida:**

|  |
| --- |
| def mochila\_fraccional(objetos, capacidad):  objetos.sort(key=lambda x: x[0]/x[1], reverse=True)  total = 0  for valor, peso in objetos:  break  return total |

**Ordenamiento por mezcla (Merge Sort)**

**Enunciado:** Implementar el algoritmo Merge Sort para ordenar una lista y analizar su comportamiento.

**Objetivo:** Trabajar recursión y complejidad O(n log n).

Interfaz de usuario gráfica, Texto, Aplicación

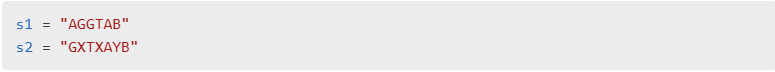
El contenido generado por IA puede ser incorrecto.

**Ejercicios Avanzados (Nivel 3)**

**5. Subsecuencia Común Más Larga (LCS - Programación Dinámica)**

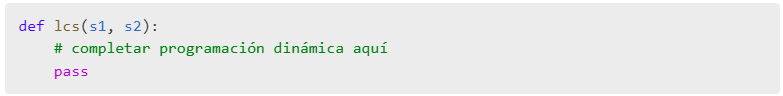
**Enunciado:** Dadas dos cadenas, encontrar la longitud de la subsecuencia común más larga (no contigua).

**Entrada:**



**Salida esperada:** 4 (subsecuencia: "GTAB")

**Objetivo:** Uso de tablas de programación dinámica. Complejidad O(n × m).



**Solución sugerida:**

|  |
| --- |
| def lcs(s1, s2):  n, m = len(s1), len(s2)  dp = [[0] \* (m + 1) for \_ in range(n + 1)]    return dp[n][m] |

**Mínimo de monedas (Bottom-Up DP)**

**Enunciado:** Dado un conjunto de monedas y un monto objetivo, encontrar el mínimo número de monedas para formar dicho monto. Retornar -1 si no es posible.

Sea dp[i] el **mínimo número de monedas necesarias para obtener la cantidad i**.

*dp = [float('inf')] \* (monto + 1)*

*dp[0] = 0 # Cero monedas para monto 0*

**Entrada:**



**Salida esperada:** 2 (por ejemplo: 3 + 3)

**Objetivo:** Diseñar solución iterativa con programación dinámica.



**Solución sugerida:**

|  |
| --- |
| def minimo\_monedas(monedas, monto):  dp = [float('inf')] \* (monto + 1)  dp[0] = 0 |