Ejercicios Análisis de Algoritmos

Ejercicio 1: Análisis de Complejidad Temporal  
Analiza la complejidad temporal del siguiente código:  
def funcion(n):  
 for i in range(n):  
 for j in range(n):  
 print(i, j)

Preguntas:

* Calcula la complejidad temporal en notación Big-O.
* Explica paso a paso cuántas operaciones realiza para una entrada de tamaño n.

Ejercicio 2: Comparación de Algoritmos de Búsqueda  
Explica y compara la complejidad de búsqueda lineal y búsqueda binaria.

Preguntas:

* Compara la complejidad temporal de ambos algoritmos.
* ¿Qué condiciones deben cumplirse para que la búsqueda binaria sea más eficiente?

Ejercicio 3: Modelamiento de Problema de Ruta Óptima  
Modela un problema real de rutas óptimas, identificando entrada, salida, restricciones y algoritmo adecuado.

Preguntas:

* ¿Qué estructura de datos usarías para representar el problema?
* ¿Qué algoritmo usarías para encontrar la ruta más barata?
* Identifica entradas, salidas, restricciones y posibles heurísticas.

Ejercicio 4: Recurrencia  
Determina la complejidad del siguiente código:  
def suma\_lista(arr):  
 if len(arr) == 1:  
 return arr[0]  
 medio = len(arr) // 2  
 return suma\_lista(arr[:medio]) + suma\_lista(arr[medio:])

Preguntas:

* Calcula la complejidad temporal utilizando el método de recurrencias.
* ¿Qué sucede con el rendimiento cuando n crece?

Ejercicio 5: Heurística Greedy en Mochila Fraccionaria  
Implementa una solución heurística para la mochila fraccionaria y analiza su complejidad.

Preguntas:

* ¿Cuál sería el criterio greedy?
* ¿Qué limitaciones tiene este enfoque?
* Analiza su complejidad y exactitud.

Ejercicio 6: Dijkstra y Complejidad  
Implementa Dijkstra utilizando heap y analiza su complejidad en grafos densos y esparcidos.

Preguntas:

* ¿Cuál es la complejidad temporal con y sin el uso de un heap?
* ¿Cómo varía el tiempo de ejecución si el grafo es denso o esparso?
* Analiza el mejor y peor caso.

Ejercicio 7: Conteo de Operaciones  
Analiza la complejidad del siguiente código:  
def mystery(n):  
 i = n  
 while i > 1:  
 i = i // 2  
 print(i)

Preguntas:

* Determina la complejidad temporal en Big-O.
* Justifica cuántas veces se ejecuta el bucle.

Ejercicio 8: Comparación de Algoritmos de Ordenamiento  
Compara la complejidad de Bubble Sort y Merge Sort.

Preguntas:

* ¿Cuál es la complejidad de cada algoritmo?
* Haz pruebas para entradas de 100, 1,000 y 10,000 elementos.
* ¿Cuál es más eficiente y por qué?

Ejercicio 9: Aplicación del Teorema Maestro  
Resuelve T(n) = 2T(n/2) + n y explica el resultado.

Preguntas:

* Aplica el **Teorema Maestro** para determinar la complejidad.
* Relaciona este resultado con un algoritmo conocido.

Ejercicio 10: TSP Heurístico  
Implementa el algoritmo del vecino más cercano para el TSP y analiza su calidad y complejidad.

Preguntas:

* Describe una solución con el método **Nearest Neighbor (Vecino más cercano)**.
* Analiza la complejidad temporal y la calidad de la solución.

Desafío 1: Árbol Binario Balanceado

Implementa una función que determine si un árbol binario está balanceado. Analiza su complejidad.

Desafío 2: Análisis de QuickSort  
Implementa QuickSort y analiza detalladamente sus casos (mejor, peor y promedio).

Desafío 3: Programación Dinámica - Fibonacci  
Implementa la serie de Fibonacci con programación dinámica y compara con la versión recursiva simple.

Desafío 4: Grafo - Detección de Ciclos  
Implementa un algoritmo para detectar ciclos en un grafo dirigido. Explica su complejidad.

Desafío 5: Problema de la Mochila 0/1  
Implementa y analiza un algoritmo para la mochila 0/1 con programación dinámica, evaluando complejidad temporal y espacial.