10003 Cutting Sticks

1. ¿Su solución es BU (Bottom-Up) o TP (Top-Down)?
   La solución es Top-Down (TD). El algoritmo utiliza una función recursiva calculate_min_cut_cost, la cual descompone el problema en subproblemas más pequeños y almacena los resultados en una tabla de memoización (dp_table) para evitar calcular los mismos subproblemas varias veces. Este es un enfoque característico de la técnica top-down con memoización.
2. ¿Cuál es la complejidad espacial y temporal? con una breve explicación.
   Complejidad espacial:
   Complejidad espacial: La complejidad espacial es $O(n^2)$, donde $O(n)$ es el número de posiciones de corte. Esto se debe a que la tabla de memoización dp_table tiene un tamaño de $n \times n$, para almacenar los costos mínimos de cortar el palo entre cualquier par de posiciones posibles.
   Complejidad temporal:
   La complejidad temporal es $O(n^3)$. Esto ocurre porque:
   - Hay $O(n^2)$ pares posibles de posiciones de corte (start, end).
   - Para cada intervalo, se intenta cortar en todos los puntos intermedios posibles, lo que añade un costo de $O(n)$ por cada intervalo. Por lo tanto, la complejidad total es de $O(n^2)$ intervalos multiplicado por $O(n)$ intentos de cortes intermedios, lo que da un tiempo total de $O(n^3)$.

3. Explique brevemente la estrategia de la solución y como hace uso de la estrategia de Memoización.
   El problema se resuelve usando un enfoque de programación dinámica, descomponiendo el problema principal en subproblemas más pequeños. La función recursiva calculate_min_cut_cost se encarga de calcular el costo mínimo para cortar el palo en el intervalo entre start y end. Cada vez que se realiza un corte, se suma el costo de cortar el palo actual a la suma de los costos de cortar los palos resultantes a la izquierda y a la derecha de la posición de corte. Se exploran todas las posiciones de corte posibles dentro del intervalo, buscando el costo mínimo. El uso de memoización se da a través de la tabla dp_table, donde se almacenan los resultados de los subproblemas ya resueltos. Esto evita recalcular el costo mínimo para intervalos ya procesados, lo cual mejora considerablemente la eficiencia del algoritmo. Si el valor del intervalo ya ha sido calculado (es decir, dp_table[start][end] != -1), simplemente se devuelve el resultado almacenado en lugar de realizar cálculos adicionales.