¿Qué es?

Nuestro proyecto UVS Express es un programa que determina qué productos cargar en cada camión —y cuáles dejar sin asignar—, así como la ruta óptima según criterios de tiempo o distancia, considerando la disponibilidad de vehículos y posibles retrasos en el recorrido o penalizaciones de tiempo. Además, cuenta con mecanismos de búsqueda eficiente de productos y tres modos de ordenamiento (por nombre, peso o volumen) tanto para los artículos ya asignados a un camión como para los que aún están pendientes.

Interfaz de usuario gráfica, Texto, Aplicación, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

**floyd**

Este fragmento implementa el algoritmo de Floyd y es utilizado para encontrar las distancias mínimas entre todas las parejas de localidades en una red de caminos. Primero, se copia la matriz original de distancias a una matriz llamada resultado, que servirá como base para las actualizaciones. Luego, se ejecutan tres bucles anidados: el más externo representa cada posible localidad intermedia (k), y los dos internos recorren todas las combinaciones de origen (i) y destino (j). En cada iteración, se verifica si pasar por la localidad intermedia k permite reducir la distancia total entre i y j. Si es así, se actualiza el valor en la matriz resultado. Este algoritmo tiene una complejidad de O(n³), ya que analiza todas las combinaciones posibles entre nodos

**Heap sort**

El procedimiento heap\_sort\_nombres ordena un arreglo de ProductoSeleccionado alfabéticamente construyendo primero un **min-heap**: inicializa un montículo vacío, inserta cada producto al final del arreglo interno y lo “hace flotar” mientras su nombre sea lexicográficamente menor que el de su padre, garantizando que la raíz quede siempre con el nombre más pequeño. Una vez cargado el heap, extrae repetidamente la raíz (el mínimo actual), reemplaza esa raíz con el último elemento, reduce el tamaño del montículo y aplica heapify-down para restaurar la propiedad de min-heap intercambiando con el hijo menor cuantas veces sea necesario. Cada elemento extraído se copia en la posición siguiente del arreglo de salida, de modo que, tras nnn inserciones y nnn extracciones, el arreglo original queda completamente ordenado y la memoria del montículo se libera.

Complejidad:

Mejor (n)

Promedio (n log n)

Peor (n log n)

Interfaz de usuario gráfica, Texto, Chat o mensaje de texto

El contenido generado por IA puede ser incorrecto.

Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto. Texto

El contenido generado por IA puede ser incorrecto. Captura de pantalla de un celular

El contenido generado por IA puede ser incorrecto.

**merge sort**

* complejidad n log n en todos los casos
* Eficiente con grandes volúmenes
* Ordenamiento estable en arreglos grandes (usa mucha memoria)

**Counting sort**

1. Se recorre el arreglo original para encontrar el valor máximo del volumen (escalado por 100 para evitar errores con decimales).
2. Se crea un arreglo auxiliar llamado count[] de tamaño igual al valor máximo encontrado más uno. Este arreglo se inicializa con ceros y servirá para contar cuántas veces aparece cada volumen.
3. Se recorre nuevamente el arreglo original. Por cada elemento, se incrementa el contador correspondiente en count[] según el volumen convertido a entero.
4. El arreglo count[] se transforma en una tabla acumulativa. Así, cada posición indica cuántos elementos hay con volumen menor o igual al valor actual.
5. Se crea un arreglo salida[] para construir el arreglo ordenado. Se recorre el arreglo original de derecha a izquierda (para mantener estabilidad) y se coloca cada producto en la posición indicada por count[], actualizando el contador.
6. Finalmente, el arreglo ordenado salida[] se copia de vuelta al arreglo original arr[], completando así el ordenamiento por volumen.

Su mayor ventaja es que no depende de comparaciones, por lo que ordena en tiempo lineal aunque no esta hecho para gran volumen de datos