Ejercicio 1:

Implementar un módulo Prim.py, de forma que tras meter el nombre de un fichero, calcule y escriba por pantalla la solución óptima.

Razone convenientemente la complejidad temporal del algoritmo implementado.

La complejidad final del módulo se obtiene mediante la unión de la complejidad del algoritmo de Prim y la complejidad de la función que muestra por pantalla los resultados obtenidos.

- **Complejidad de Prim obtenida:** $O(m) + O(m) * (O(1) + O(m) * (O(1) * (O(m) * (O(1) + O(1))) + O(1)) = O(m) + O(m) * (O(1) + O(m) * O(m) + O(1)) = O(m) + O(m) * O(m^2) = O(m) + O(m^3) = O(m^3)$.
- **Complejidad de printPrim obtenida:** $O(1) + O(m) * O(1) + O(1) + O(m) * O(1) = O(1) + O(m) + O(1) + O(m) = O(m)$.

- **Complejidad final del módulo Prim.py obtenida:** $O(m^3) + O(m) = O(m^3)$

Implementar un módulo PrimTiempos.py, que vaya creando grafos aleatorios de diversos tamaños y calculando el tiempo que tarda el algoritmo hecho en el apartado anterior en resolver el problema, de forma que se pueda ir rellenando la siguiente tabla.

Prim (tiempos en milisegundos)

Pondremos "FdT" para tiempos superiores a 10 minutos

N	t Prim
256	1203
512	9065
1024	78782
2048	FdT

Razone: ¿siguen la complejidad calculada los tiempos de la tabla?

La complejidad calculada sigue aproximadamente los tiempos de la tabla debido a que en el algoritmo de Prim tenemos dos ifs internos que no se evaluarán a true cada uno de ellos un número determinado de veces y, por tanto, no se ejecutarán siempre las operaciones internas a estos.

Lo que deriva en un aumento del tiempo de ejecución y dando lugar a que, aunque la complejidad sea cúbica y que el tamaño del problema se duplique en cada iteración del bucle, los tiempos no sean incrementados en un factor de $2^3 = 8$ de forma exacta en cada iteración.