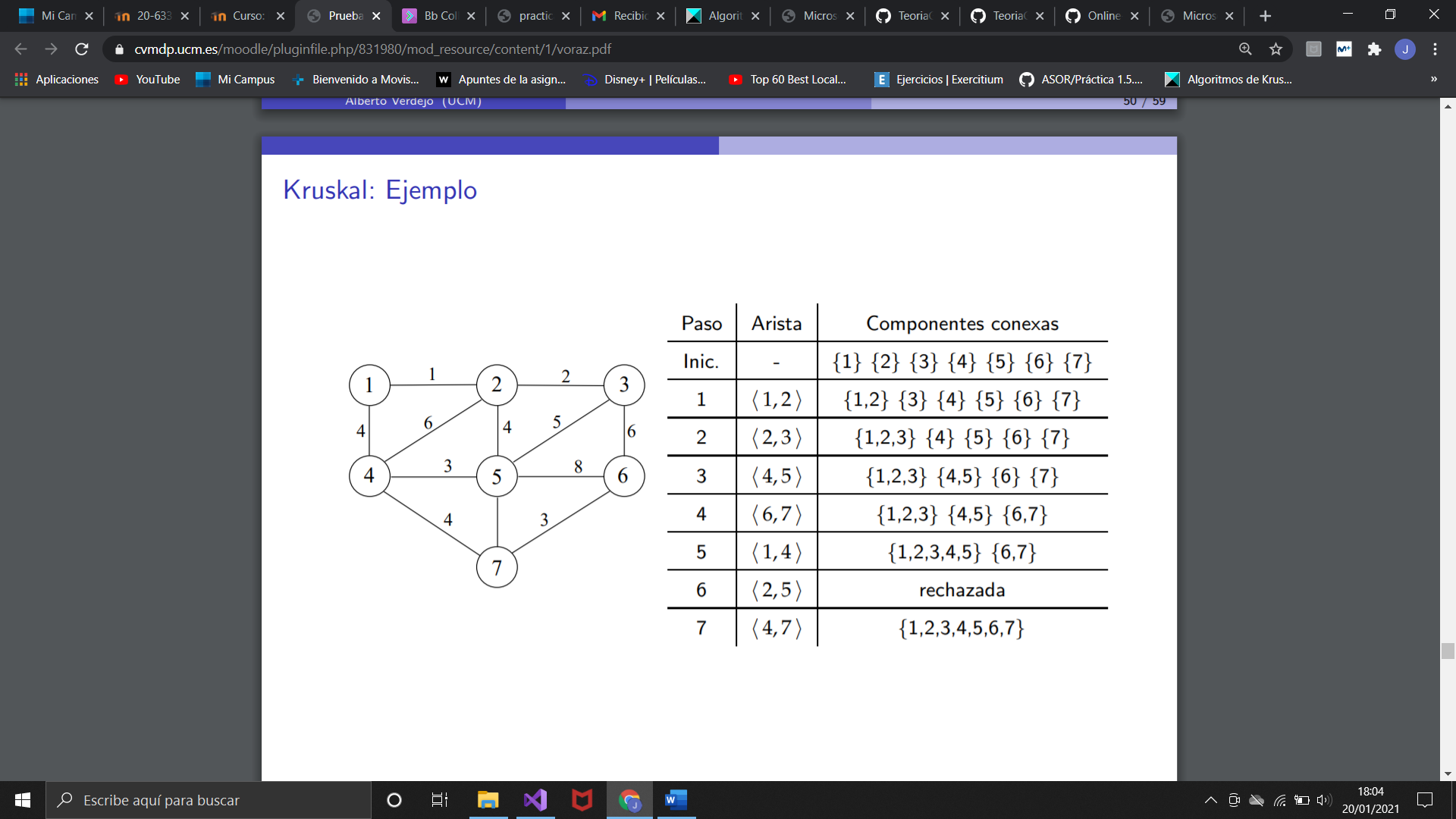
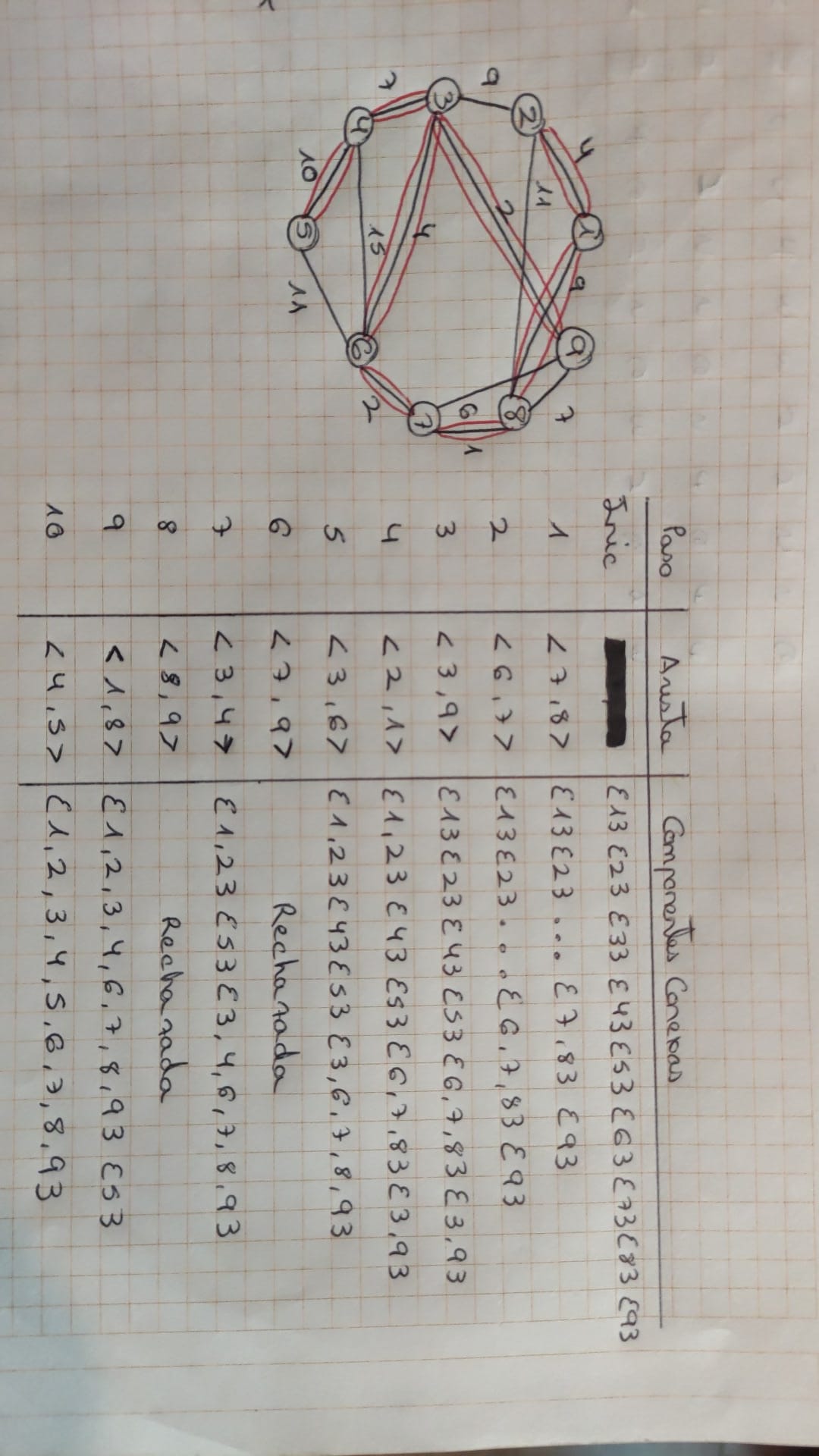
**Algoritmo de Kruskal con estructura de partición con compresión de caminos**

En esta memoria podremos ver los costes y las diferencias entre una entrada con un numero de casos de prueba sencillo y un numero de casos de prueba voluminosos, en los que nos apoyaremos en graficas para poder detallarlos mejor.

* **Casos de prueba sencillos:**
* Para el primer caso de prueba he utilizado el ejemplo de las transparencias, habilitado por el profesor, con el que he comprobado que nuestro algoritmo funcionaba de forma correcta (entrada en archivo “ ejemploProfe.txt ”).



* Segundo caso de prueba sencillo (entrada en archivo “ ejemplo.txt ”).



* **Coste Previsto:**

**Ordenar las aristas** tiene un coste *O(a log a)* que es el típico de ordenar un array de longitud *a*. Como el grafo es conexo *a ≥ n-1* ,siendo n el número de vértices. El máximo número de aristas posibles es *a ≤ C(n, 2) = ½ n (n-1)2*. Con el mínimo número de aristas posibles tendremos un coste de *O(a log(n-1)) ⇒ O(a log n)*. Con el máximo número de aristas posibles el coste será *O(a log(½ n (n-1)2)) ⇒ O(a log n2) ⇒ O(a 2 log n) ⇒ O(a log n)*.

El coste de **construir la partición** es *O(n)* puesto que hay que inicializar un array listaAristas[n] y MST[n].

El coste del bucle voraz se analiza por medio del **coste de la partición**. Partiendo de *n* conjuntos disjuntos, el coste para fusionar todos los disjuntos en un único conjunto es *O(c α(c, n))*, donde *c=b+f*, siendo *b* el total de llamadas a buscar y *f≤n-1* el total de llamadas a fusionar.

El bucle itera por *a* aristas como máximo, por lo que se producen *b = 2a* llamadas a buscar en el peor caso. Las operaciones fusionar siempre serán *f=n-1*, que son las necesarias para fusionar *n* conjuntos disjuntos. Con *c=b+f=2a+n-1* el coste de nuestra partición será esencialmente lineal en este *c*, es decir, *O(c) ⇒ O(2a+n-1) ⇒ O(2a)* dado que *a≥n-1*.

De los tres costes en secuencia *O(a log n)*, *O(n)* y *O(2a)* el máximo es *O(a log n)* usando la versión simplificada del algoritmo de Kruskal.

* **Graficas con tiempos:**

Se ha ejecutado 1000 veces el algoritmo, después se ha dividido el tiempo obtenido por esa cantidad además de realizar más de una ejecución por archivo, obteniendo la media y teniendo mejores resultados.

* El primer punto de la gráfica es el archivo “ ejemploProfe.txt ” en el que tenemos ***7 vértices y 12 aristas***, y tiene un tiempo de ejecución de ***2 ms***.
* El segundo punto de la gráfica es el archivo “ ejemploIni.txt ” en el que tenemos ***9 vértices y 14 aristas***, y tiene un tiempo de ejecución de ***3 ms***.

El primer y segundo punto casi no se distinguen en la grafica ya q tiene datos muy parecidos.

* El tercer punto de la gráfica es el archivo “ ejemploMedio.txt ” en el que tenemos ***100 vértices y 150 aristas***, y tiene un tiempo de ejecución de ***15 ms.***
* El cuarto punto de la gráfica es el archivo “ ejemploGrande.txt ” en el que tenemos ***400 vértices y 500 aristas***, y tiene un tiempo de ejecución de ***47 ms.***
* El quinto punto de la gráfica es el archivo “ ejemploVoluminoso.txt ” en el que tenemos ***1000 vértices y 1500 aristas***, y tiene un tiempo de ejecución de ***114 ms.***
* El sexto punto de la gráfica es el archivo “ ejemploMaximo.txt ” en el que tenemos ***2500 vértices y 3000 aristas***, y tiene un tiempo de ejecución de ***292 ms***.

*El eje X comprende las Aristas, y el eje Y los milisegundo(ms)*