ANALISIS EFICIENCIA PRÁCTICA 1.

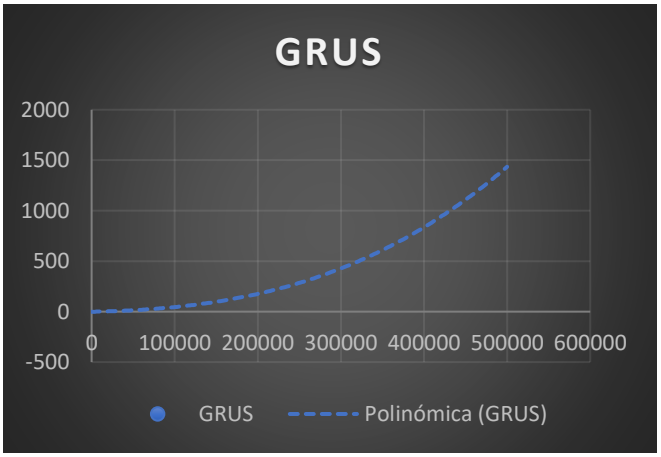
Evaluación de la eficiencia respecto al tiempo, mediante medidas de tiempo utilizando distintos tamaños del fichero de entrada.

Se analizará el análisis correspondiente a las etapas 2, 3 y 4 del algoritmo, las cuales son: creación de la lista de usuarios, creación de la lista de grumos y ordenación y selección de los grumos.

Tomadas las respectivas medidas, mediante Excel, se obtienen estos resultados con y podemos corroborar el orden de magnitud que tienen las siguientes etapas.

Creación lista grumos.

ejemplo.txt	0	15
test100.txt	0,002	100
test500.txt	0,014	500
test1000.txt	0,035	1000
test2000.txt	0,084	2000
test5000.txt	0,162	5000
test12000.txt	0,811	12000
test20000.txt	1,614	20000
test30000.txt	3,6	30000
test40000.txt	8,528	40000
test50000.txt	10,033	50000
test100000.txt	50,539	100000
test250000.txt	283,563	250000
test500000.txt	1435,471	500000




```
public static void listaGrus(ArrayList<Integer> usr, ArrayList<listaUsuarios> red, ArrayList<Integer> asig, ArrayList<ArrayList<Integer>> grus) {
    for (int i=0; i<usr.size();i++) {
        int numeroGrumo=usr.get(i);
        ArrayList<Integer> grumo = new ArrayList<>();
        if ((!asig.contains(numeroGrumo))) {
            grus.add(uber_amigos(numeroGrumo,red,grumo));
            asig.addAll(grumo);
        }

    }
}
```

Como los add y son O(1) y los contains O(i).
$\sum_{i=0}^{n} O(1) + O(1) + O(i) = \sum_{i=0}^{n} O(i)$

Pero en este caso también tenemos la llamada a uber_amigos() que es una función recursiva.

```java
public static ArrayList<Integer> uber_amigos(int usr_ini, ArrayList<listaUsuarios> red, ArrayList<Integer> grumo) {
    grumo.add(usr_ini);
    for (int i=0; i<red.size();i++) {
            int usr1 = red.get(i).getUsr1();
            int usr2 = red.get(i).getUsr2();

            if ((usr_ini==usr1) && (!grumo.contains(usr2))) {
                uber_amigos(usr2,red,grumo);
            }
            if ((usr_ini==usr2) && (!grumo.contains(usr1))) {
                uber_amigos(usr1,red,grumo);
            }
    }
    return grumo;
}
```

Ya que no es sencillo calcular su coste mediante relaciones de recurrencia, se analizará de manera global.
Los get y las comparaciones son O(1), los contains O(n), como no solo hay una única llamada recursiva también podemos ver que el orden puede pasar a ser $O(n^2)$, y nos quedaremos con que el orden de uber_amigos() es $O(n^2)$.

Como tenemos i operaciones con dos recursiones llegamos a la conclusión de que a = 2:

$$\sum_{i=c1}^{n\pm c2} O(i^a) = O(n^{a+1}) => \sum_{i=0}^{n} O(i^2) = O(n^{2+1}) = O(n^3)$$

A su vez se corresponderá tanto con el peor como con el mejor de los casos.