# Informe de Prototipos: Números Sociables

Antes de leer este Informe tener en cuenta el entorno de evaluación: Una computadora de escritorio con un procesador de 3 años de uso; Sistema Operativo Windows 10 con extensiones estéticas (Fondos de Pantalla y Widgets); Lenguaje de Programación Python.

## Prototipo 1:

Pensado como una serie de funciones, primero aquella que nos da el resultado de la suma de los divisores de un número; seguida la que nos devuelve un subconjunto de números distintos que cumplen con un período dado en donde la suma de los divisores del último elemento es el primer elemento. Por último, la función maestro que va a resguardar cada subconjunto distinto posible de un rango dado.

**Detalles**

* La función principal trabaja con un rango, esto vuelve al programa dependiente de la entrada N, posteriormente se mostrará que rangos que superen el millón (necesarios para encontrar conjuntos de ciertos periodos) llevan a la ejecución al rango de los minutos.
* La función sociable, llamada por principal, cuenta con un bucle mientras ejecutado N veces (N veces llamada a la función), y a su vez, este ciclo llama a divisores con otro ciclo mientras que se ejecuta M veces (M veces dado por el período).
* A esto se le suma a su vez la complejidad agregada por las diferentes restricciones y controles que debemos realizar para poder obtener siquiera una salida con sentido.
* Este prototipo falla en dar correctamente 1 solo conjunto, en el caso de un periodo 5 y un rango de 20000, le correspondería la salida: [12496, 14288.0, 15472.0, 14536.0, 14264.0]; sin embargo, el programa no tiene una restricción o forma de purgar a variantes de este conjunto con sus miembros en diferente orden, causadas por el ciclo manejado por contador. Y además devuelve conjuntos de periodos no correspondidos.
* Sumado a que el testeo de este prototipo es rudimentario ya que no se colocaron opciones de ingreso de usuario.

**TEST (Todas las aplicaciones excepto el editor cerradas)**

Probaremos el efecto del tamaño del rango y del tamaño del período

1. **Tomando una entrada de tamaño 1.300.000 y período 4:** 295.3313s (VsCode) | 290.7691s (MINGW64 Console o Git Bash)
2. **Tomando una entrada de tamaño 1.000.000 y período 7:** 359.6400s (VsCode) | 356.2855s (Git Bash)
3. **Tomando una entrada de tamaño 500.000 y período 9:** 178.6269s (VsCode) |178.4876s (Git Bash)

## Prototipo 2:

Este prototipo es la siguiente evolución para disminuir las llamadas a funciones y posible redundancia en los controles.

Aquí se trabaja en una sola función que va a trabajar con el tipo de dato Conjunto, que cuenta con la característica de no permitir elementos repetidos dentro de si mismo. Para procesar los números usamos tres ciclos, uno manejado por la entrada N, otro manejado por el periodo M y el último constante.

**Detalles**

* El primer ciclo crea una bandera para evitar bucles con los números perfectos o amigos que interfieran con periodos más grandes (Ya que los conjuntos no permiten los casos {a, b, a, b, …,a, b}). Esta bandera no evita sin embargo que estos se carguen.
* Al finalizar el segundo ciclo hace una pregunta para resguardar solo aquellos que cumplan con la característica sociables. Debemos usar cast de frozenset debido a restricciones del lenguaje, esto sin quererlo incrementa la complejidad.
* Finalizando el primer ciclo se pasará a hacer una comprobación meramente formal para devolver si existen conjuntos o no (Esto pudiendo descartarse para aumentar la velocidad).
* Para mostrar los resultados existe un analisis del conjunto para mostrar solo aquellos que cumplan la condición de if, esto debido a que se contamino con conjuntos amigos o perfectos. Esto se podría evitar si antes se realiza dicho if, evitando tener que recorrer el conjunto una vez más.

**TEST (Todas las aplicaciones excepto el editor cerradas)**

Probaremos el efecto del tamaño del rango y del tamaño del período

1. **Tomando una entrada de tamaño 1.300.000 y período 4:** 267.4142s (VsCode) | 245.2902s (Git Bash)
2. **Tomando una entrada de tamaño 1.000.000 y período 7:** 297.9873s (VsCode) | 291.7032s (Git Bash)
3. **Tomando una entrada de tamaño 500.000 y período 9:** 148.3412s (VsCode) | 145.0158 (Git Bash)

## Prototipo 3:

Este prototipo cambio el tipo de dato de subconjunto a una lista para ver si esto facilitaba el trabajo con los elementos, además de corregir los errores del Prot2. La estructura general y lógica siguen igual.

**Detalles**

* Se elimino la comprensión de conjuntos al finalizar e imprimir la salida, agregándose condiciones al if de carga. Esto generó, sin embargo, muchas condiciones y comparaciones con la compuerta and, aumentando dicha complejidad.
* Se quito la bandera para evitar el bucle debido a que subconjunto ya no era un set() sino una lista, evitando dicho bloqueo.
* La salida es incorrecta ya que muestra conjuntos del tipo {a, b, a, b, …, a, b} que debieron descartarse, sin embargo por razones de como esta escrito esta solución, arreglar ese detalle puede complejizar demasiado el problema.

**TEST (Todas las aplicaciones excepto el editor cerradas)**

Probaremos el efecto del tamaño del rango y del tamaño del período

1. **Tomando una entrada de tamaño 1.300.000 y período 4:** 251.9703s (VsCode) | 247.2063s (Git Bash)
2. **Tomando una entrada de tamaño 1.000.000 y período 7:** 299.8341s (VsCode) | 288.5187s (Git Bash)
3. **Tomando una entrada de tamaño 500.000 y período 9:** 148.6928s (VsCode) | 146.4375s (Git Bash)