Propuesta de Método de Selección para Algoritmos Evolutivos basada en ponderación con la Escala Logarítmica

Por Alejandro Ramos @arhcoder

"Logarithmic Random Ponderated Selector" **(LRPS)** es un algoritmo cuyo propósito es **recibir una lista de objetos y seleccionar uno aleatoriamente,** con el sesgo de que mientras más al inicio de la lista esté cada uno, más probable será que sea seleccionado; con un **factor de ponderación dado por la forma de la curva logarítmica-exponencial.**

¿Para qué?

Cuando se trabaja con modelos que requieren mecanismos de selección, es común utilizar algoritmos basados en el azar, en los que –de manera metafórica– se hace girar una ruleta que define cuál de los ***n*** objetos será escogido en cada selección. Estos métodos son útiles cuando se prefiere que exista la misma probabilidad de selección para todos los elementos del conjunto, sin embargo, puede no ser esta la intención.

La idea tras esta propuesta nace del análisis de fenómenos para los que existen eventos con mayor frecuencia de estar presentes, pero de los cuales no se conoce el factor que determina dicha magnitud. Siendo concreto, ***LRPS*** se origina de mi intento de replicar la composición de música a través de algoritmos genéticos con aleatoriedad, siendo claro el hecho de que según el estilo de música, género, época, etc., ciertos patrones de composición son más comunes; por ejemplo: un compás rítmico de 4/4, notas largas o cortas, armonías simples o complejas, etc. La música popular suele tener más presente el compás 4/4, y acordes sólo mayores-menores, melodías de voz no veloces, ritmos relajados, entre otros; es entonces que para intentar replicar la composición de música popular, podríamos considerar estas características como las más comunes, pero sin dejar de lado que es posible tener más variedad de decisión.

Nota: Dicho proyecto de composición musical con algoritmos genéticos se puede encontrar en: ***https://github.com/arhcoder/M.I.A***.

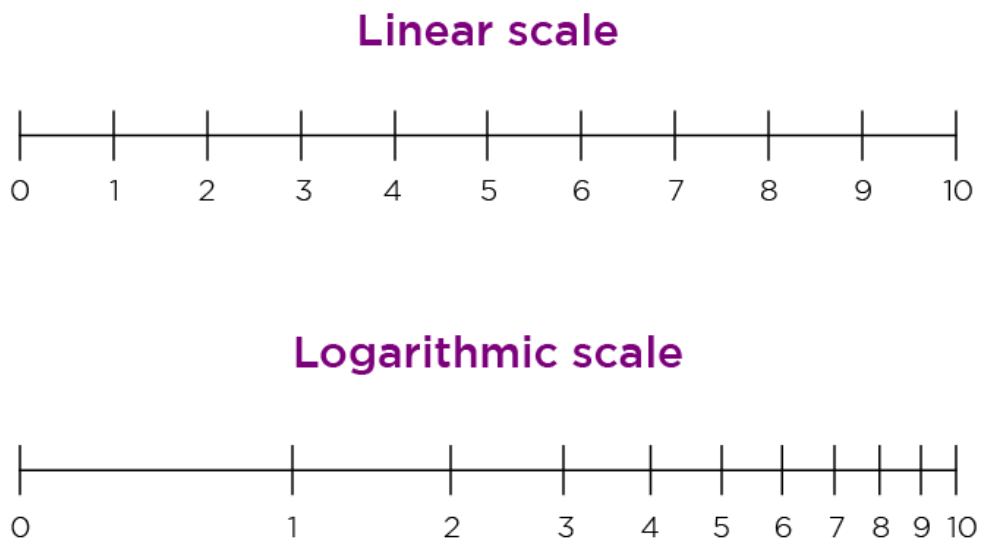
¿Cómo?

Si para la toma de decisión en un modelo existen sesgos y ponderaciones sin un número que represente su magnitud; esta propuesta de algoritmo puede ser de utilidad.

Por ejemplo: Imaginando que se quiere decidir el sabor de un helado para comer, basados en una lista que contiene los sabores favoritos:

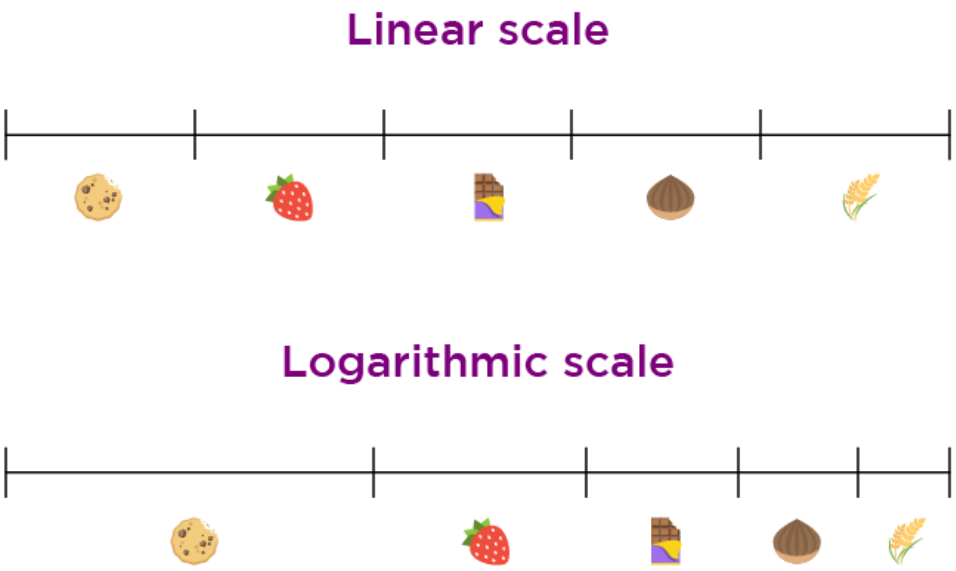
- **1ro: Galleta.**
- **2do: Fresa.**
- **3ro: Chocolate.**
- **4to: Nuez.**
- **5to: Pistache.**

En el modelo se ha decidido que el sabor galleta es el favorito, por lo que debería ser más frecuente su selección, enseguida la fresa, el chocolate, etc. La cuestión importante es que no se tiene un número tangible que permita determinar ese sesgo de importancia con respecto a los demás sabores, simplemente se sabe que se prefieren más con forme más altos estén en el top. La escala logarítmica puede proporcionar una relación de crecimiento con base en su naturaleza matemática exponencial:

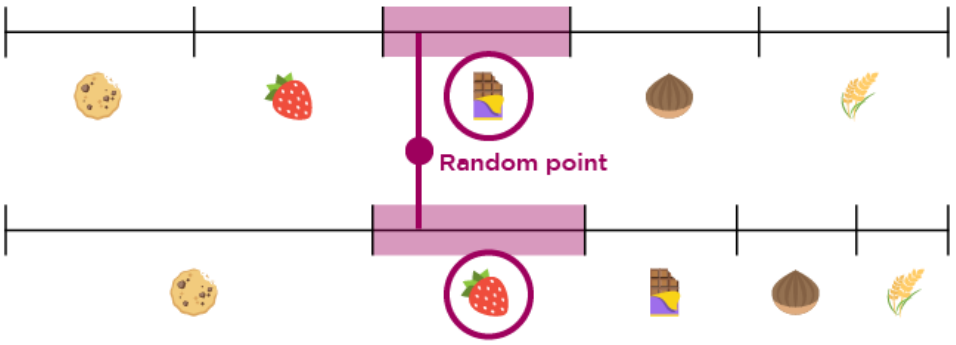


Una vaga representación de cómo luce la diferencia entre una escala lineal y una logarítmica, está mostrada en la imagen anterior; si la tomamos como ejemplo, podríamos entender este algoritmo de decisión como colocar un punto de manera aleatoria en cualquiera de las dos escalas, en donde cada espacio entre dos divisiones representa un objeto a seleccionar. Para el caso de la escala lineal, la probabilidad de encontrarse con el objeto entre ***0 y 1,*** es la misma que del objeto entre ***1 y 2,*** o ***2 y 3.*** En cambio, con la escala logarítmica, la probabilidad de caer en el objeto entre ***0 y 1*** es muy distinta que la del objeto entre ***5 y 6.***

Utilizando datos reales para el ejemplo de escoger un sabor de helado, tendríamos la siguiente comparación de escalas:



Esta es la forma que tiene la escala del algoritmo ***LRPS*** comparada a un algoritmo de ruleta. Si lanzáramos un punto aleatorio entre este espacio; obtendríamos las decisiones para ambos tipos de algoritmos, que serían:



- ***Chocolate*: para un algoritmo de ruleta.**
- ***Fresa*: para el algoritmo LRPS.**

Al ser más grande el espacio de la fresa con respecto al chocolate, es más probable que nos topemos con comer fresa en lugar de chocolate.

Matemáticas

La manera de construir una escala logarítmica adaptada a la ***n*** cantidad de objetos a seleccionar es posible a través de un **conjunto *P* de puntos sobre un eje cartesiano** en los que cada punto delimita el espacio de selección de cada uno de los ***n*** elementos de la lista, utilizando la siguiente fórmula:

$$\mathbb{P} = \{\ \boldsymbol{p_i} : \boldsymbol{p_i} = (n \log_{n+1}(i+1), 0),$$

$$n \in \mathbb{Z},\ n > 1,$$

$$i\ \in \mathbb{Z},\ \ 1 \leq i \leq n\ \ \};$$

$$\boldsymbol{p_i}\ es\ \boldsymbol{coordenada}\ (x, y);$$

Es decir, dado el mismo caso de construir la escala sobre un eje cartesiano, con ***n*** cantidad de espacios (objetos) y por ende ***n*** cantidad de puntos de división, la distancia entre cada punto ***i*** y el ***origen*** (coordenada (0, 0)) está dada por la fórmula:

$$\boldsymbol{d_i = n \log_{n+1}(i+1)};$$

Si se desea encontrar la coordenada del punto 1, para el ejemplo de los cinco sabores de helados (n = 5), la coordenada sería:

$$p_1 = (5 \log_{5+1}(1+1),\ 0);$$

$$p_1 = (5 \log_6(2),\ 0);$$

$$p_1 = (1.934,\ 0);$$

Entonces, para encontrar todos los puntos necesarios para construir la escala logarítmica de cinco objetos:

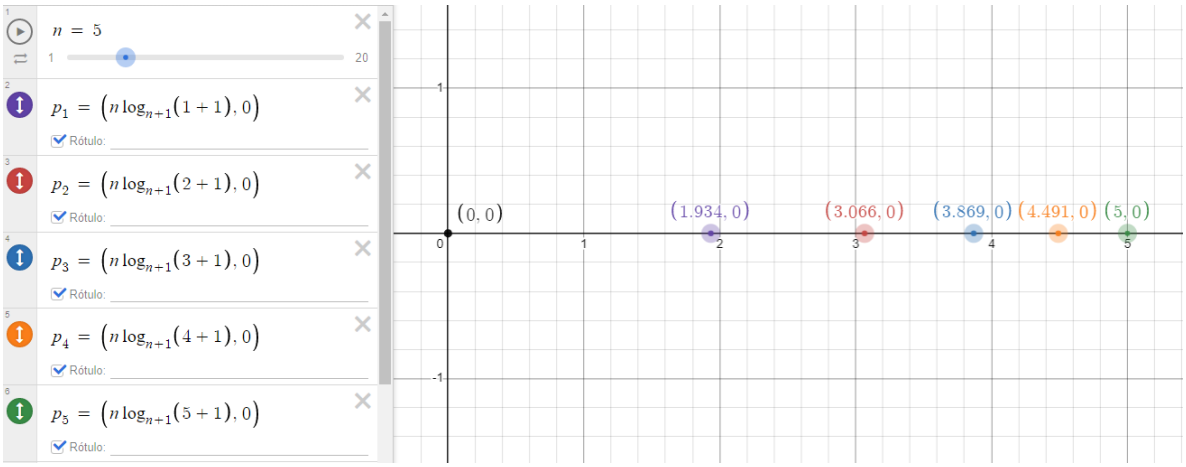
$$p_1 = (5 \log_{5+1}(1+1),\ 0);$$

$$p_2 = (5 \log_{5+1}(2+1),\ 0);$$

$$p_3 = (5 \log_{5+1}(3+1),\ 0);$$

$$p_4 = (5 \log_{5+1}(4+1),\ 0);$$

$$p_5 = (5 \log_{5+1}(5+1),\ 0);$$



Nota: El uso de eje cartesiano es innecesario para el algoritmo, puesto que sólo se necesita saber la distancia entre el origen y cualquier otro punto de la escala; por lo que sólo será necesaria la fórmula de distancia: $d_i = n\log_{n+1}(i+1)$.

Algoritmo

El pseudocódigo del algoritmo de selección se muestra a continuación:

```
lrps(list objects = [obj1, obj2, obj3, obj4, obj5]):

    # n = amount of objects:
    n = objects.length

    # If there are not objects:
    if n == 0:
        return "Objects list empty :c"

    # If list has only 1 object, returns the obj1:
    if n == 1:
        return obj1

    # Gets a random decimal number between [0, n]:
    randomPoint = random_between(0, n)

    # Loop between 1 and n:
    for i = 1 to n:
        if randomPoint <= (n * log_n+1(i+1)):
            return obj_i
                                        @arhcoder
```

Se recibe de entrada una lista de objetos de la cuál se obtiene el valor de ***n*** como la cantidad de objetos en la selección. Un par de casos de paro se dan cuando la lista está vacía o contiene sólo un elemento, en casos contrarios se procede con el resto del algoritmo, en donde se obtiene un número decimal aleatorio entre 0 y ***n***, y se inicia un ciclo de 1 hasta ***n***.

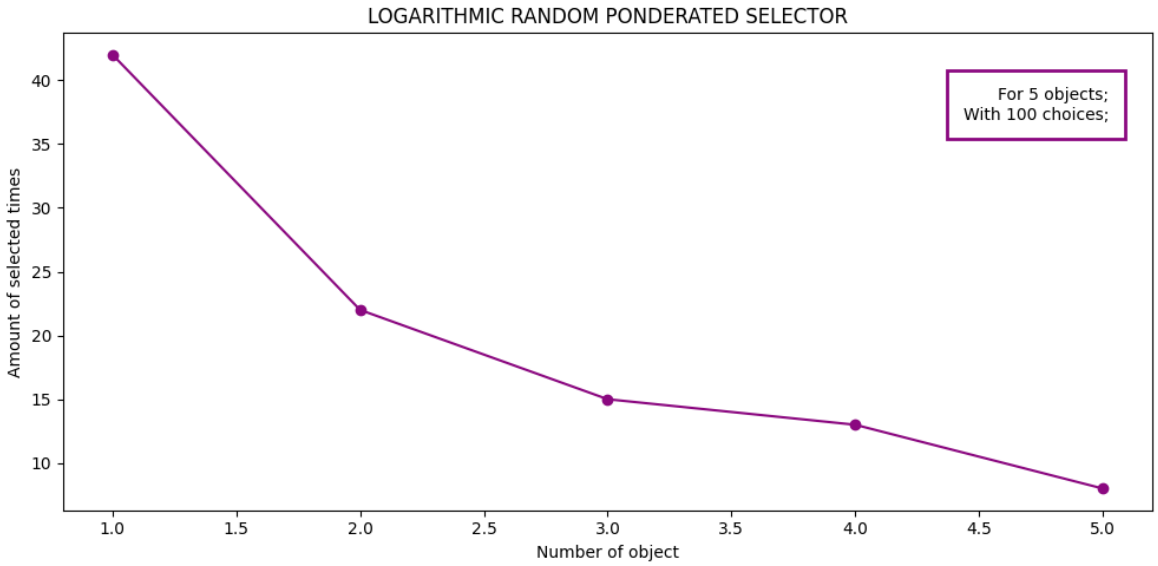
Dentro del ciclo se calcula –usando la misma analogía de un plano cartesiano– la distancia del ***"origen"*** hasta el primer punto de la escala logarítmica **(p_1),** utilizando la fórmula de distancia antes descrita. Si el punto aleatorio está entre 0 y la distancia del primer punto de la escala; es decir, si el punto aleatorio quedó antes de p_1 **(punto aleatorio <= distancia a p_1),** entonces se toma la decisión de tomar el objeto de dicho espacio (el objeto número 1), en caso de que el punto aleatorio esté por encima de este primer punto de la escala, se continúa con el ciclo hasta encontrar el espacio en que cayó el punto.

El algoritmo tiene una **complejidad lineal O(n)** –con notación ***"Big O"***–, recordando que ***n*** es la cantidad de objetos en la lista inicial de decisión.

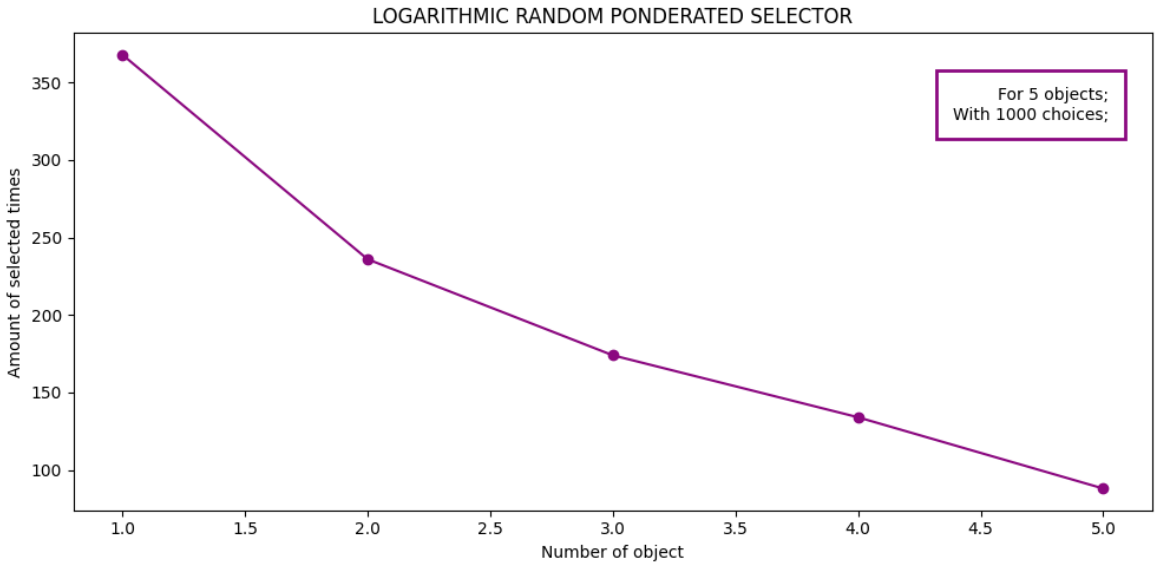
Experimentos

Haciendo pruebas de conteo para las decisiones tomadas por el algoritmo, a fin de comprobar el sesgo en las elecciones, **repitiendo la selección una cierta cantidad de veces y contando cuántas veces se eligió cada uno de los objetos, se graficó:**

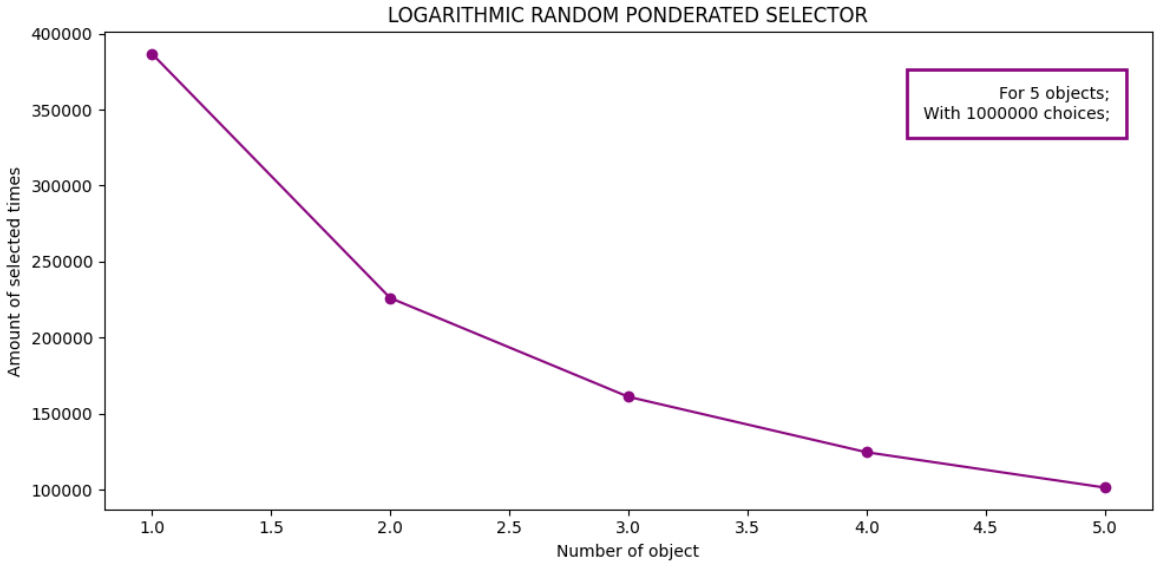
1. Con 5 objetos y 100 repeticiones:



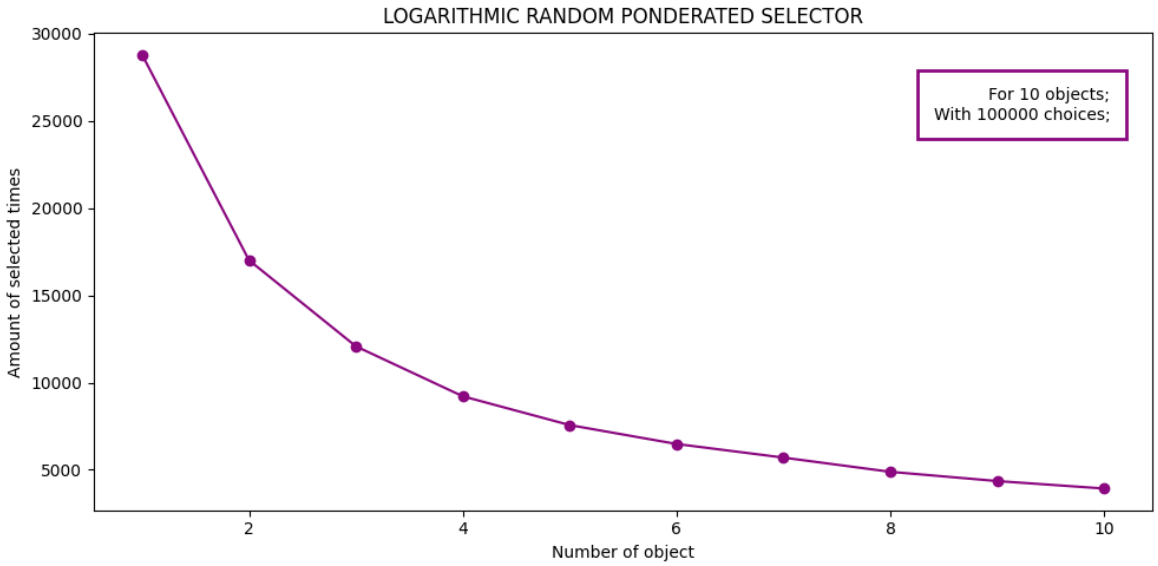
2. Con 5 objetos y 1,000 repeticiones:



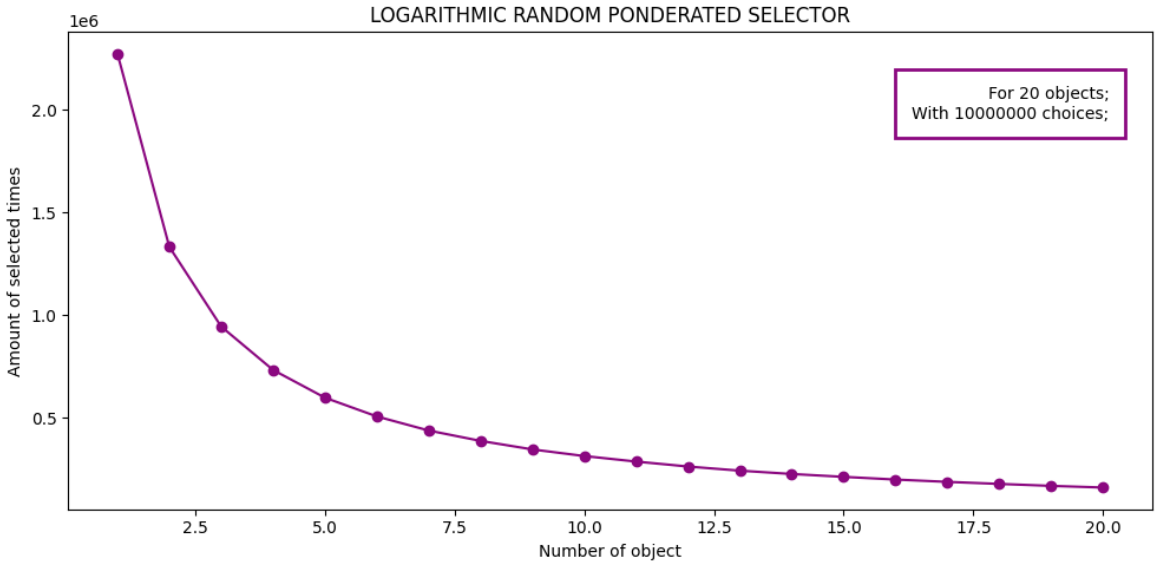
3. Con 5 objetos y 1,000,000 repeticiones:



4. Con 10 objetos y 100,000 repeticiones:



5. Con 20 objetos y 10,000,000 repeticiones:



Se puede comprobar que se obtiene la forma de la curva de un logaritmo.

Nota: Para la generación de números aleatorios se utilizó la función ***random*** del **core de Python v.3.11 de 64 Bits.** El script de medición-graficación así como el algoritmo LRPS se encuentran en ***este repositorio***.

Conclusiones

Esta es sólo una propuesta de algoritmo, puede ser de utilidad en; por ejemplo, las decisiones tomadas en **algoritmos genéticos con elitismo.** Si tenemos una lista de individuos de una población, ordenados por su desempeño, y queremos tomar decisiones que afecten con mayor frecuencia a los mejores individuos, pero no queremos perder la probabilidad de tomar en consideración a los individuos menos aptos, esta puede ser una buena opción de método de selección. Puede ser también una propuesta interesante para encontrar el punto óptimo entre la **exploración y explotación en metaheurísticas.**

En caso de que para un problema específico se quiera que dos o más objetos tengan la misma probabilidad de selección, se puede considerar –para esta implementación– que dichos objetos estén incluidos en una estructura de datos que les contenga como **UN** solo objeto, de modo que si el algoritmo los escoge, arrojará a los objetos en conjunto, posteriormente y mediante una decisión del tipo ruleta (como la escala lineal) se puede escoger alguno de estos.

Consulta el código del algoritmo desde el **repositorio principal** en ***https://github.com/arhcoder/LRPS-Algorithm***, cualquier **colaboración y/o propuesta de mejora** será bien recibida, al igual que una **estrella de GitHub.**

Escríbeme a **arhcoder@gmail.com** para cualquier inquietud, o si te resultó útil este aporte en alguna aplicación y búscame en redes sociales como **@arhcoder**.

Muchas gracias por leer 😀

Alejandro Ramos, @arhcoder

❤