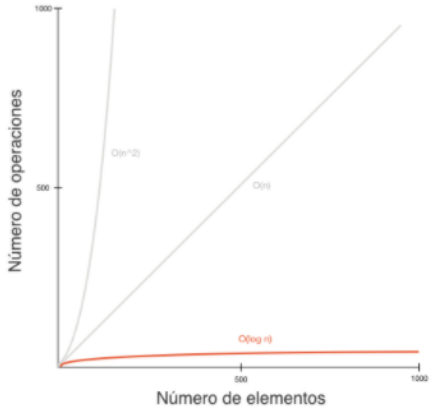
COMPLEJIDAD COMPUTACIONAL.

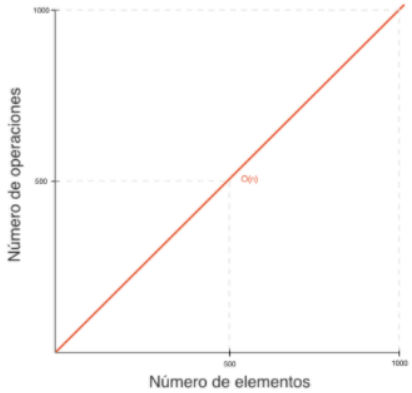
- **Definición de un algoritmo de orden constante.**
  A medida que aumenta la cantidad de datos el número de operaciones se mantiene constante. O(1) significa que sólo se ejecuta una operación, O(2) significa que se ejecutan dos operaciones, O(3) tres operaciones y así sucesivamente. Pero el número de operaciones no es importante, lo importante es que, independiente del número de datos de entrada, el rendimiento va a ser constante. Una operación puede ser una asignación de una variable, una comparación, o algún cálculo aritmético (suma, resta, multiplicación, división, etc.), entre otros.

- **Definición y ejemplo de un algoritmo de orden logarítmico.**
  A medida que aumenta la cantidad de datos, el número de operaciones aumenta, pero no de forma proporcional a los datos.



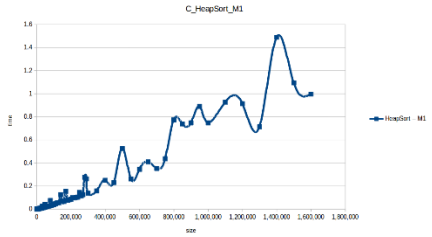
Complejidad logarítmica

- **Definición y ejemplo de un algoritmo de orden lineal.** Aumenta los datos de entrada. Es decir, que, si para una lista de 100 elementos el algoritmo tarda x segundos, para una lista de 1000 elementos (10 veces más grande) tardará 10 veces más.



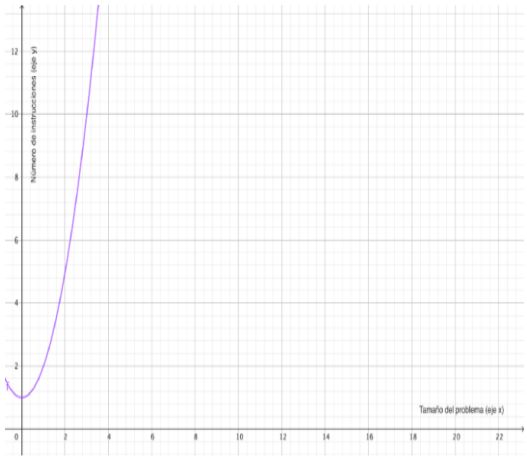
COMPLEJIDAD COMPUTACIONAL.

- **Ejemplo de un algoritmo de orden nlogn.** Aparece en algoritmo con recursión, ej: ordenamiento rápido y se considera una complejidad buena.



- ***Definición y ejemplo de un algoritmo de orden cuadrático***. *Complejidad cuadrática, aparece en bucle anidados, por ejemplo, ordenamiento directo por selección. Se puede argumentar que el orden de un algoritmo nos define cuan rápido es. Así un algoritmo con O(n) será más rápido que otro con O(n^2), por la sencilla razón de que n^2 >= n.*

```
1
2   public class ejemplo3 {
3
4       final static byte[] n = { 4, 2, 5, 1, 6, 0, 6, 9, 5, 4, 5, 0 };
5       static int resultado = 0;
6
7       public static void main(String[] args) {
8
9           for(int i=1; i<=n.length; i++){
10
11              for(int j=n.length; j>=1; j--){
12              resultado = i*7;
13              resultado /= 2;
14              }
15
16          }
17
18          System.out.println("El resultado es: "+resultado);
19
20      }
21
22  }
```



Representación gráfica de la función $f(x) = 2n^2 + 1$, que en realidad corresponde a $f(x) = n^2 + 1$, pero para

- **Definición de un algoritmo de orden polinomial.** De define como aquel con función de complejidad temporal dentro de una cota superior asintótica (denominada a veces "orden") O(p(n)) para alguna función polinómica p, donde n denota el tamaño de la entrada.

- **Definición de un algoritmo de orden exponencial.** A medida que aumenta la cantidad de datos, el número de operaciones crece de forma

exponencial. Por ejemplo, una complejidad cuadrática (O(n^2)) si para un dato necesitamos 2 operaciones, para 2 datos vamos a necesitar 4 operaciones, para 3 datos 9 operaciones y así sucesivamente.

- **Definición de un algoritmo de orden factorial.** Está definida para todos los enteros positivos, junto con el 0. ¿Qué valor debe tener 0!? Es el producto de todos los enteros mayores o iguales que 1 y menores o iguales que 0. Pero no hay tales enteros. Por lo tanto, ¡definimos que 0! equivale a la identidad multiplicativa, que es 1. (¡Definir 0! = 1 empata bien con la fórmula para escoger kkk cosas de nnn cosas. Supón que queremos saber cuántas maneras hay para escoger nnn cosas de nnn cosas. Eso es fácil, porque solo hay una manera: escoge todas las nnn cosas.