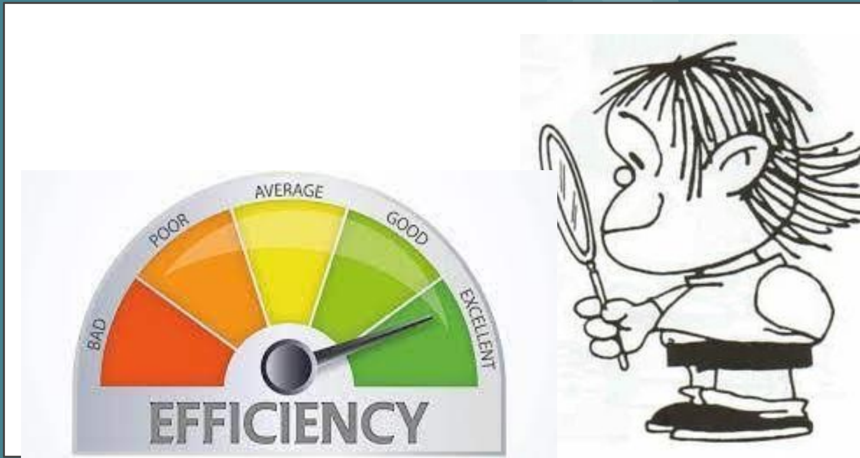
Eficiencia



Objetivos de la materia:

- Maximizar el uso de recursos

  tiempo de ejecución y utilización de memoria

Código debe ser:

- EFICAZ: Hacer lo que debe hacer y hacerlo bien
- EFICIENTE: Utilizar los recursos mínimos

- *Resolver un problema implica:*
  - *Algoritmo o método de resolución*
  - *Codificación del método.*

Analizar algoritmos

- *Puede haber muchos parámetros, pero los más usuales son:*
  - *TIEMPO DE EJECUCIÓN*
  - *MEMORIA UTILIZADA*

Seudocódigo y java

```
ALGORITMO  medirTiempo
    ENTERO tiempo, n
    n <- 100000
    tiempo = calculaDuracion (n)
    ESCRIBIR tiempo
FIN ALGORITMOS
```

```java
public static void main(String[] args) {
        long c = calculaDuracion(1000000)
        System.out.println("Duración : " + c/1e6 + " ms");
    }
public static <E> long calculaDuracion(int cantRep) {
        long comienzo  = System.nanoTime();
        //codigo con cantRep repeticiones a medir
        long tarda = System.nanoTime() - comienzo;
        return (tarda);
    }
```

*1 Milisegundos = **1.000.000** Nanosegundos*

*1 Nanosegundos = 1.0 x **10-6** Milisengundos*

*sout ("tiempo:" + **tarda** + “ ns (nanosegundos), ” **tarda/1e6** + " ms (milisegundos");*

Cómo analizar un código en base al tiempo?

- Se necesita un modelo de computación:
- las instrucciones se ejecutan de modo secuencial
- cada instrucción sencilla tarda exactamente una unidad de tiempo
    - (asignación, comparación, adición)
- no se indica la unidad utilizada
- suponemos una memoria infinita
- Se analiza el tiempo de ejecución

Tiempos de Ejecución - Ti

- Operación elemental: corresponde a un tiempo de ejecución acotado por una constante que depende de la implementación.
- Por convención se toma la unidad.
- Ejemplo de Operaciones Elementales

Tiempo de una secuencia - $T_1 + T_2$

Sentencias

diametro ← Pi * 2 * radio → 1 asignación , 2 multiplicaciones, Total 3 tiempos

diametro ← diametro+1 → 1 asignación , 1 suma, Total 2 tiempos

El tiempo de ejecución $T_s(n) = T_1(n) + T_2(n) = 5$

Tiempo de Ejecución en alternativas

- El tiempo de ejecución t_{si} nunca es mas grande que el tiempo empleado por la condición mas el mayor de los tiempos de S_1 y S_2.
- La evaluación de la condición se llama t_{cond}
- La evaluación del máximo de los tiempos de S_1 y S_2 se la muestra como $max(t_1 , t_2)$

Requiere: Tiempo Si $t_{SI} = t_{cond} + t_1$

SI **cond** HACER
S1
FIN SI

SI **cond** HACER
S1
SINO
S2
FIN SI

El tiempo de ejecución del peor caso $T_w(n)$ es el mayor tiempo de ejecución de S1 y S2

Requiere: Tiempo Si/Sino $t_{SINO} = t_{cond} + max(t_1, t_2)$

Tiempo de Ejecución en Sentencias FOR

```
PARA cond = 1 HASTA n
HACER
        S1
FIN PARA
```

Se debe considerar

- La inicialización de la variable = t_{ini}
- La evaluación de la condición = t_{cond}
- La evaluación de las condiciones para decidir la finalización se realiza una cantidad de iteraciones cantIt de veces en true y la última
- Los incrementos se evalúan con el tiempo t_{inc}
- El tiempo interno del ciclo (Serie de entencias internas -

Serie S_1 de sentencias (T_1)

```
for (int i= 0; i < N; i++) S1;
```

Inicialización

condición

incremento

$$\text{Tiempo para } t_{PARA} = t_{ini} + cantIt*(t_{cond} + t_{INTERNO} + t_{inc}) + t_{cond}$$

Ejemplos de sentencias FOR

- Tiempo de inicialización t_{ini} (j = 0) → 1 asignación = 1 tiempo
- Tiempo de condición t_{cond} (j <=n) → 1 operación = 1 tiempo
- Los incrementos t_{inc} (i++) → 1 operación matem = 2 tiempoa
- Tiempo interno $t_{INTERNO}$:
  - 1 asignación + 1 suma + 2 acceso al vector = 4 tiempos
- Cantidad de Itenaciones = n veces

PARA **j = 0** HASTA **n-1**
HACER
a[j] ← a[j] + 10
FIN PARA

```
for (j= 0; j < N; j++) S1;
```

Tiempo para = t_{ini} + cantIt*(t_{cond} + t_{int} + t_{inc}) + t_{cond}

- *Tiempo Para* $t_{PARA} = t_{ini} + cantIt*(t_{cond} + t_{INTERNO} + t_{inc}) + t_{cond}$

$$= 1 + n* \quad (1 + 4 + 2) \quad + 1 = 1 + 7\,n + 1 = 7\,n + 2$$

Cálculo de tiempos

```
MODULO suma (ENTERO n) RETORNA ENTERO
    ENTERO sumaParcial
    PARA j = 1 HASTA n HACER
        sumaParcial ← sumaParcial + j*j*j
    FIN PARA
    RETORNAR sumaParcial
FIN MODULO
```

```
int suma (int n){
    int sumaParcial = 0;        //1
    for (int j= 1; j<=N; j++)   //2
    sumaParcial += j*j*j ;      //3
    return sumaParcial;         //4
}
```

- Las instrucciones [1] y [4] valen una unidad.
- La línea [3] cuenta 4 unidades (dos *, una + y una asignación) y se repite n veces.
- La línea [2] tiene el costo de inicialización de i, testeo de i<=n y el incremento. Costo total: 1 para inicializar, n+1 para comprobar, 2n para el incremento. Resultado: 3n+2.
- Resultando un total de $1+(3n+2)+4n+1=7n+4$.

Tiempo de ejecución

n es la cantidad de datos y se analiza en función de ese

El tiempo de ejecución de un programa en función de **n**, se denomina **t(n).**

- *Si utilizamos arreglos o matrices, n es el nro. de elementos que la componen*

Se puede medir:

- *Ejecutando el programa, reloj en mano,*
- *Contando instrucciones a ejecutar sobre el código y multiplicando por el tiempo requerido por cada instrucción*

Otras repetitivas

- Tiempo de inicialización t_{ini} (j = 0) → 1 asignación = 1 tiempo
- Tiempo de condición t_{cond} (j <=n) → 1 operación = 1 tiempo
- Los incrementos t_{inc} (i++) → 1 operación matem = 2 tiempoa
- Tiempo interno $t_{INTERNO}$:
  - 1 asignación + 1 suma + 2 acceso al vector = 4 tiempos
- Cantidad de Itenaciones = n veces

PARA **j = 0** HASTA **n-1**
HACER
a[j] ← a[j] + 10
FIN PARA

```
for (j= 0; j < N; j++) S1;
```

$$\text{Tiempo para} = t_{ini} + cantIt*(t_{cond} + t_{int} + t_{inc}) + t_{cond}$$

- *Tiempo Para* $t_{PARA} = t_{ini} + cantIt*(t_{cond} + t_{INTERNO} + t_{inc}) + t_{cond}$

$$= 1 + n* \quad (1 + 4 + 2) \quad + 1 = 1 + 7\,n + 1 = 7\,n + 2$$

Otras repetitivas

Ciclos REPETIR HASTA (while y repeat).

- Se calcula el tiempo de ejecucion de las instrucciones internas, mas el costo de evaluar la condicion por el numero maximo de iteraciones (peor caso)

$$\text{Tiempo RepetirHasta} = maxIt(T_1 + t_{cond})$$

REPETIR
S1
HASTA **cond**

Ciclos MIENTRAS

- Se calcula el tiempo de ejecucion de las instrucciones internas, mas el costo de evaluar la condicion por el numero maximo de iteraciones (peor caso), y una evaluacion mas de

$$\text{Tiempo Mientras} = maxIt(T_1 + t_{cond}) + t_{cond}$$

MIENTRAS **cond** HACER
S1
FIN MIENTRAS

Concepto

- El análisis de algoritmos es una estimación teórica de la cantidad de recursos necesarios para ejecutarlos.
- El tiempo de funcionamiento de un algoritmo se expresa como una función de la longitud de entrada en relación con un número de pasos o ubicaciones de almacenamiento.
- Estas estimaciones dan una idea de instrucciones razonables de búsqueda de algoritmos eficientes.
- Se estima su complejidad en sentido asintótico, para un n muy grande

Tipos de análisis....

- El tiempo de ejecución del peor caso, $T_w(n)$ - worst case - $T_{peor}(n)$
  - El máximo tiempo de ejecución sobre todas las entradas de tamaño n
  - Puede no ser muy fiel
- El tiempo promedio de ejecución: $T_a(n)$ - average - $T_{promedio}(n)$
  - promedio de tiempos sobre todas las entradas de tamaño n
  - Puede ser más fiel
  - En algunas ocasiones puede ser difícil de determinar
- El tiempo de ejecución del mejor caso: $T_b(n)$ - best case - $T_{mejor}(n)$
  - El menor de los tiempos sobre todas las entradas de tamaño n
  - Puede ser engañoso en un algoritmo lento que trabaja rápido sobre algunas entradas

Independiente de la computadora ....ASINTOTICO

Tipos de análisis

Peor Caso: se considera el máximo uso de recursos

Caso Promedio: se considera un promedio de uso de recursos.

Analisis probabilistico: se considera el uso de recursos de cada instancia en función de su probabilidad de ser ejecutada.

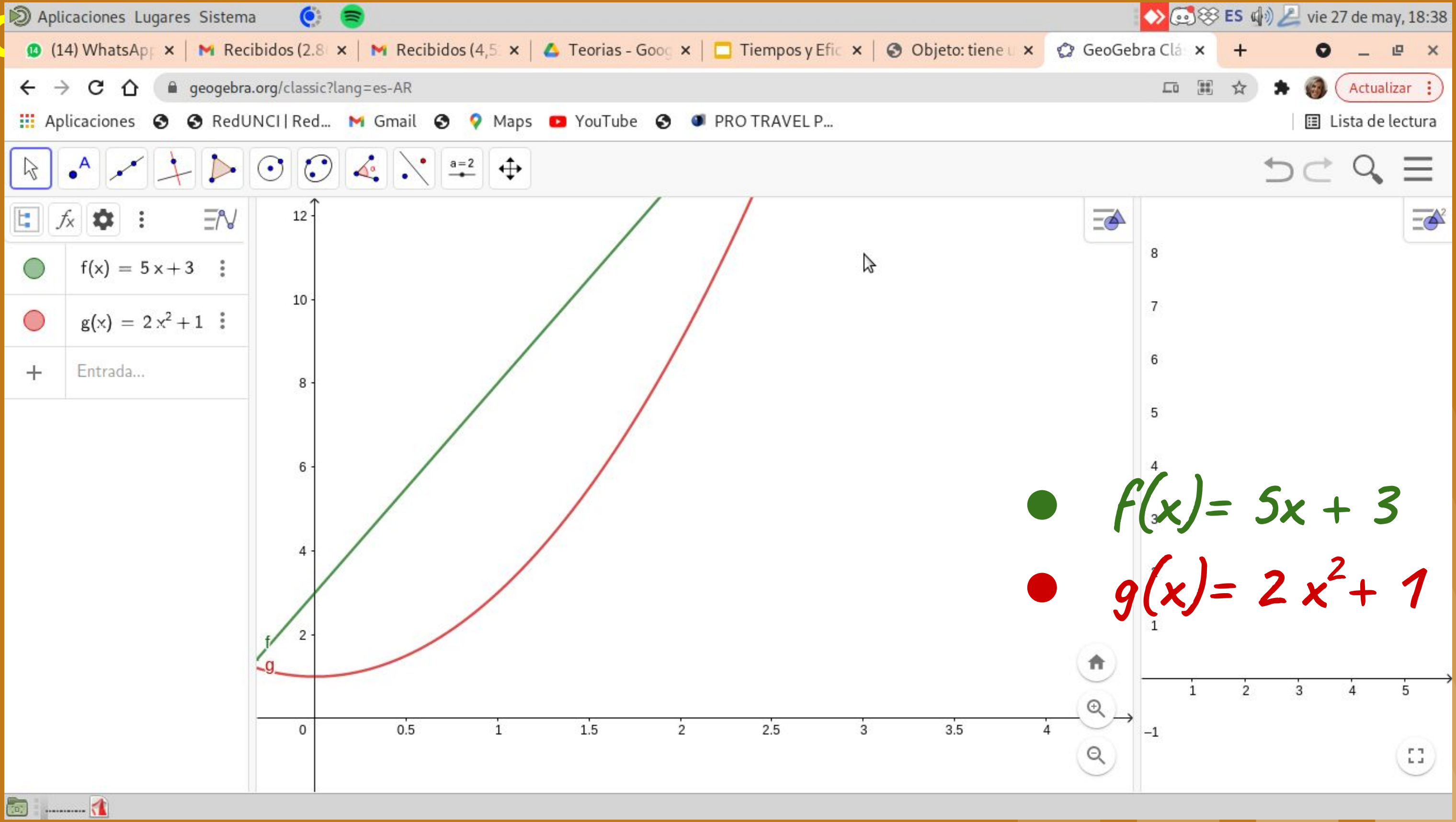
Mejor Caso: se considera el mínimo uso de recursos

- *Cómo sabemos qué algoritmo es más eficiente?*

Se utilizan representaciones gráficas del tiempo en función del tamaño de los datos de entrada.

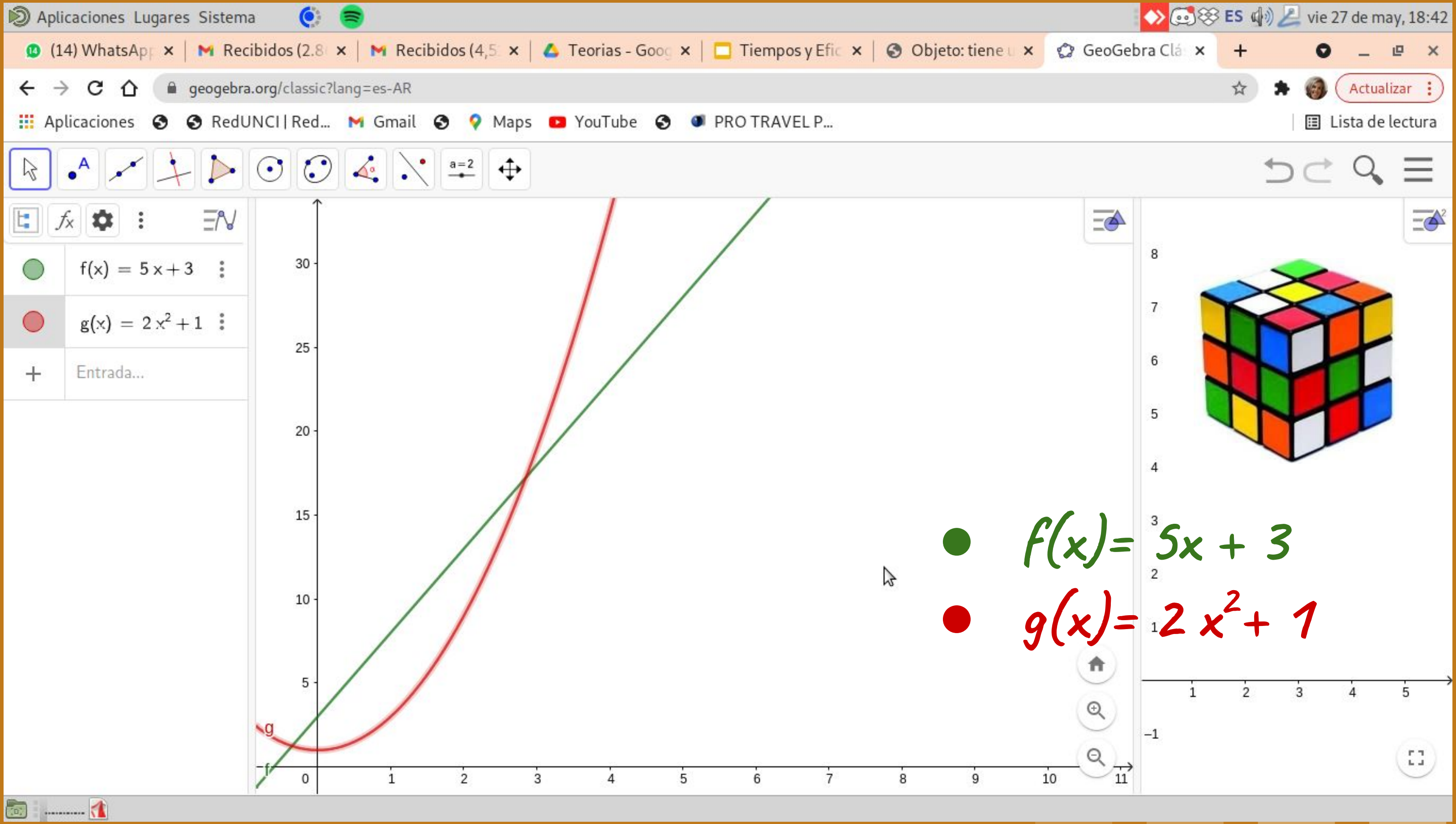
Geogebra

- *Cómo sabemos qué algoritmo es más eficiente?*



- $f(x) = 5x + 3$
- $g(x) = 2x^2 + 1$

- *Cómo sabemos qué algoritmo es más eficiente?*

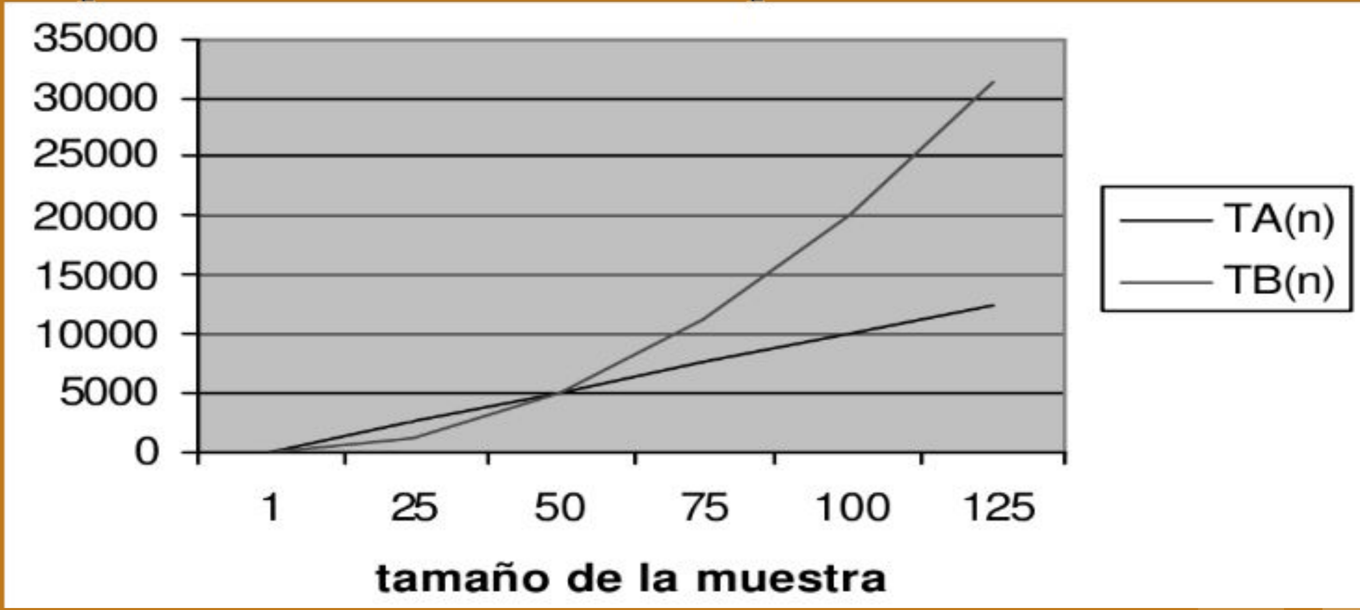


- $f(x) = 5x + 3$
- $g(x) = 2x^2 + 1$

- *Qué algoritmo es más eficiente?*

- Ejemplo Si consideramos una cantidad n = 10 y n = 100
  - $T_A(10) = 100 * n = 1000$; $T_A(100) = 10000$
  - $T_B(10) = 2 * n^2 = 200$; $T_B(100) = 20000$

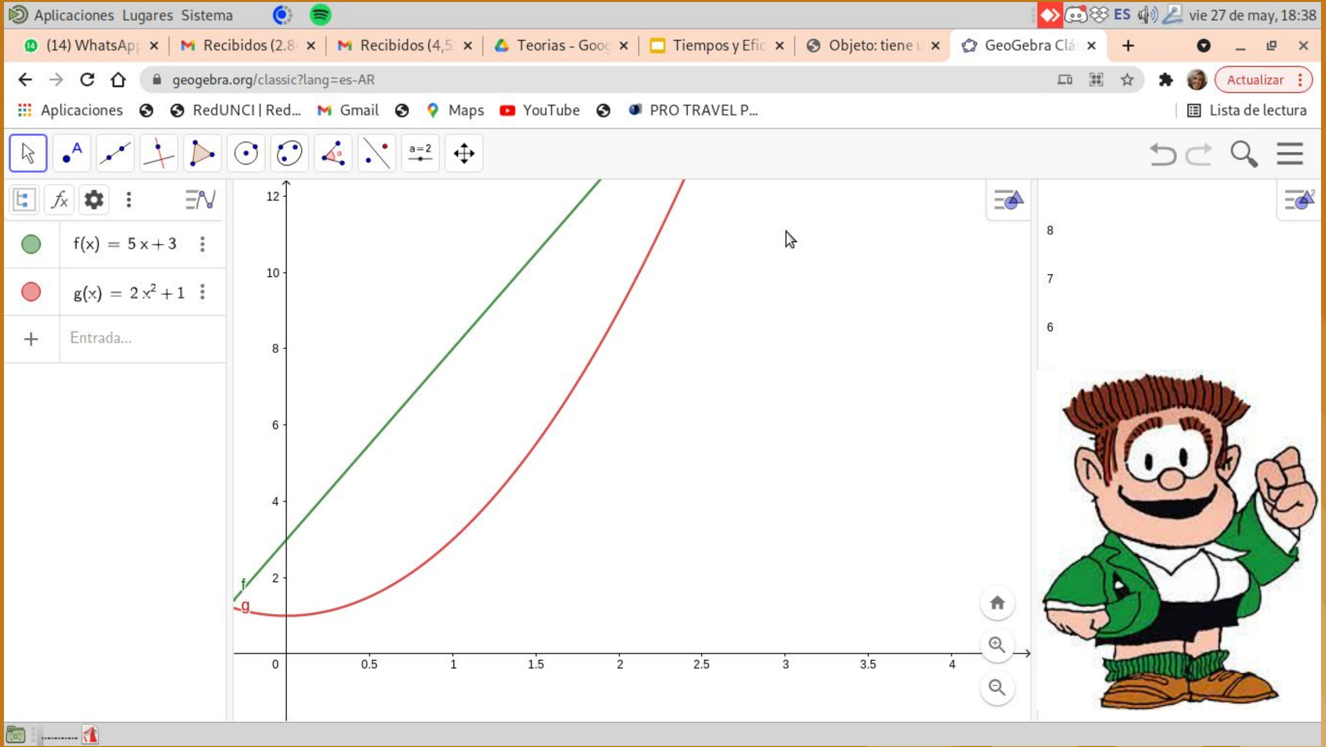
¿Para qué valores de n lo pienso?



Cómo sabemos qué algoritmo es más eficiente?

Se utilizan representaciones gráficas del tiempo en función del tamaño de los datos de entrada.

Geogebra

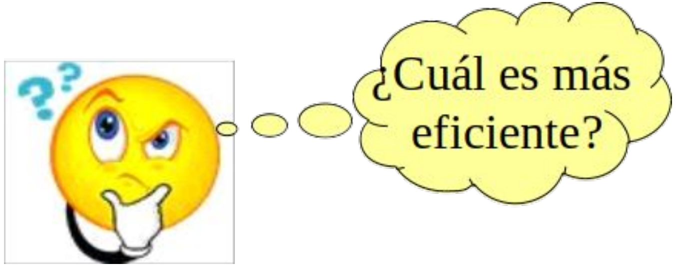


Menor tiempo de ejecución?

1. Supongamos un problema y dos algoritmos A y B para resolverlo.

   $T_A(n) = 100 * n$

   $T_B(n) = 2 * n^2$



Si consideramos una cantidad n = 10

$T_A(10) = 100 * 10 = 1000$

$T_B(10) = 2 * 10^2 \quad = 200$

Eficiencia?

Comparemos gráfico

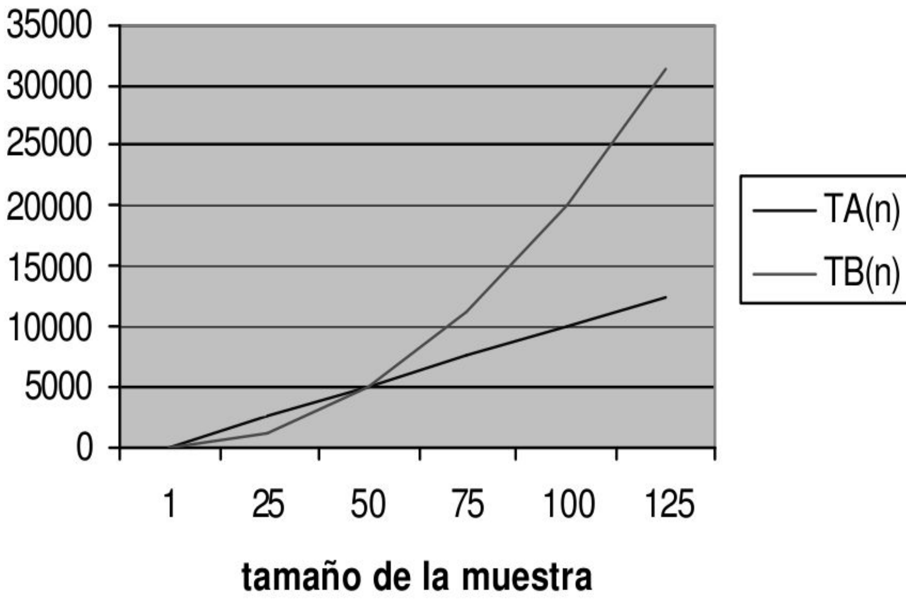
Si consideramos una cantidad n = 10 y n = 100

$T_A(10) = 100 * n = 1000;$

$T_A(100) = 10000$

$T_B(10) = 2 * n2 \quad = 200 ;$

$T_B(100) = 20000$



Notación Asintótica O

- Los algoritmos estiman su complejidad en el sentido asintótico, (para un cantidad muy grande de datos de entrada) .
- El análisis de algoritmos proporciona estimaciones teóricas de recursos necesarios .
- Sirve para dar una idea de instrucciones razonables en la búsqueda de algoritmos eficientes.
- A veces se requiere de ciertas suposiciones acerca de la implementación particular del algoritmo (llamado modelo de computación).

Notación **asintótica O**,

La notación asintótica superior, es una función que sirve de cota superior o techo de un conjunto de funciones y es de gran utilidad para clasificar la eficiencia de los algoritmos.