# Análisis de Algoritmos 2024

Trabajo Práctico 2 - Notación Asintótica 2

*Fecha de entrega: 6 de Septiembre de 2024*

El objetivo de este trabajo es realizar los ejercicios propuestos de forma colaborativa. Para los ejercicios compartidos entre grupos, pueden compartir el desarrollo del ejercicio o realizar dos resoluciones diferentes.
En particular para este práctico se pide en algunos casos tener código funcionando, verifica tener una computadora que ejecute tu código si te tocó algún ejercicio de este tipo. Se pedirá ejecución en clase. Además completa la justificación de tu respuesta con gráficas de las funciones. Podés usar GeoGebra para esto.
Todos los ejercicios deben ser resueltos por todos los estudiantes, pero en este trabajo sólo deben ser publicados los que te tocaron según la asignación de la tabla siguiente. Dicha asignación corresponde a la presentación oral del TP. Si el día de la práctica hay varias resoluciones logradas, elegiremos 1 grupo para exponer ese ejercicio.
Por supuesto, si querés compartir la resolución de otros ejercicios en este proyecto Overleaf, también son bienvenidas pero no es obligatorio.

**Nota:** Colocá el nombre del/de los estudiante/s que resolvió/resolvieron el ejercicio al iniciar la resolución del ejercicio que te tocó.

Las respuestas deben quedar escritas en este mismo .tex, en Overleaf.

| **Grupos (Estudiantes por grupo)** | **Ej. asignado** |
|---|---|
| Grupo 1 (Sthefany, Victoria, Paula, Adriano, Facundo) | 1.3 — 2.5a — 3.1 |
| Grupo 2 (Sebastián, Bautista, Antonio, Albany, Luis) | 2.4.a— 2.5b — 3.2 |
| Grupo 3 (Lucas, Valentina, Nicanor, Ignacio, Franco) | 2.1 — 2.4.b — 3.3 |
| Grupo 4 (Manuel, Jan, Ulises, Demian , Luciano) | 2.2 — 2.4.c — 3.1 |
| Grupo 6 (Roman, Ulises, Guillermo, Jamiro, Juan) | 1.2 —2.6 — 3.2 |
| Grupo 7 (Christopher, Tomás, Joaquín, Leonard, Facundo) | 1.4 — 2.5c — 3.3 |
| Grupo 8 (Facundo, Belén, Jeremías, Bruno, Kevin) | 1.1 — 2.3 — 3.2 |

## 1. Repaso de $O$

1. Supongamos un algoritmo con un costo de $O(n)$ tarda 20 segundos en realizar un determinado procesamiento sobre una computadora de 3GHz. Responda:

a) ¿Cuánto se tardaría en hacer el mismo procesamiento en una máquina de 1 GHz?

b) ¿Como se calcularía el tiempo de ejecución si la cantidad de datos a procesar se duplica?

Ejercicio 1.1 - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin

$a)$ Sabiendo que el costo de un algoritmo de $f(n)$ tarda 20 segundos en una computadora de 3 GHz, entonces para calcular el tiempo en una computadora de 1 GHz, se tiene que:

- Una computadora de 1 GHz es 3 veces más lenta que una de 3 GHz
- Entonces el tiempo que se tarda en procesar el algoritmo una maquina de 1GHz es 3 veces mayor

$\therefore$ Tiempo en 1 GHz = 20 segundos x 3 = 60 segundos

$b)$ Dado que la complejidad es $O(n)$, el tiempo de ejecución es directamente proporcional a la cantidad de datos n. Esto significa que si se duplica la cantidad de datos, el tiempo de ejecución también se duplica.

- Si procesar $n$ datos toma 20 segundos en una maquina de 3 GHz, entonces procesar $2n$ datos va a tomar el doble de tiempo

$\therefore$ Tiempo de ejecucion = 20 segundos x 2 = 40 segundos

Falta justificar por el principio de invariancia y asumir las condiciones de que no hay más variables que el cambio de velocidad de procesamiento. Luego se define la función de tiempo . esto para el inciso a). para el inciso b) se debe considerar que cuando cambia la muestra se coloca la nueva magnitud donde está n inicialmente. Luego se resuelve la función según sea la función de tiempo.

2. ¿Qué tipo de crecimiento caracteriza mejor a estas funciones?

| Función | Constante | Lineal | Polinomial | Exponencial |
|---|---|---|---|---|
| $3n$ | | X | | |
| $1$ | X | | | |
| $3/2 * n$ | | X | | |
| $2n^3$ | | | X | |
| $2n$ | | X | | |
| $3n^2$ | | | X | |
| $1000$ | X | | | |
| $(3/2)^n$ | | | | X |

**Ejercicio 1.2 - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani**

Hecho en la tabla ↑

3. Dadas dos clases de complejidad $O(f(n))$ y $O(g(n))$, decimos que $O(f(n)) \subseteq O(g(n))$ si y sólo si para toda función $h(n) \in O(f(n))$ sucede que $h \in O(g(n))$

   a) ¿Qué significa que $O(f(n)) \subseteq O(g(n))$? ¿Qué se puede concluir cuando simultáneamente $O(f(n)) \subseteq O(g(n))$ y $O(f(n)) \supseteq O(g(n))$? Pruebe su afirmación por contradicción.

   b) Ordene y grafique utilizando $\subseteq$ las siguientes clases de complejidad:

   - $O(1)$
   - $O(\sqrt{x})$
   - $O(\sqrt{2})$
   - $O(log\ x)$
   - $O(log\ x!)$
   - $O(x+1)$
   - $O(1/x)$
   - $O(log\ x)$
   - $O(log\ log\ x)$
   - $O(2^x)$
   - $O(x)$
   - $O(x^x)$
   - $O((log\ x))$
   - $O(x!)$
   - $O(x\ log\ x)$

Resolución ejercicio 1.3 (a) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

**Definición:**

$$O(f(n)) \subseteq O(g(n))$$

Expresa que el orden de crecimiento $f(n)$ es menor o igual al de $g(n)$. Gráficamente, la función $f(n)$ crece igual o más lentamente que $g(n)$ y la clase de funciones que crecen a lo sumo como $f(n)$, está incluída dentro de la clase de funciones que crecen a lo sumo $g(n)$.

**¿Qué se puede concluir cuando simultáneamente $O(f(n)) \subseteq O(g(n))$ y $O(f(n)) \supseteq O(g(n))$?**
Cuando esto sucede, se puede concluir que $O(f(n)) = O(g(n))$, es decir, f(n) y g(n) poseen el mismo orden de crecimiento asintótico.

**Prueba de la afirmación por contradicción:**

Suposición inicial: $O(f(n)) \subseteq O(g(n))$ **y** $O(f(n)) \supseteq O(g(n))$, pero $O(f(n)) \neq O(g(n))$

Si se cumple que $O(f(n)) \neq O(g(n))$, implica que existe una función $h(n)$ tal que:

- $h(n) \in O(f(n)) \wedge h(n) \notin O(g(n))$

o bien

- $h(n) \in O(g(n)) \wedge h(n) \notin O(f(n))$

Caso 1) Sea $O(f(n)) \subseteq O(g(n))$ :

$$\forall h(n) \in O(f(n)) \implies h(n) \in O(g(n))$$

Luego, si hayamos una función $h(n) \in O(f(n))$ pero $h(n) \notin O(g(n))$, se contradice la suposición inicial: $O(f(n)) \subseteq O(g(n))$.

Caso 2) Sea $O(f(n)) \supseteq O(g(n))$ :

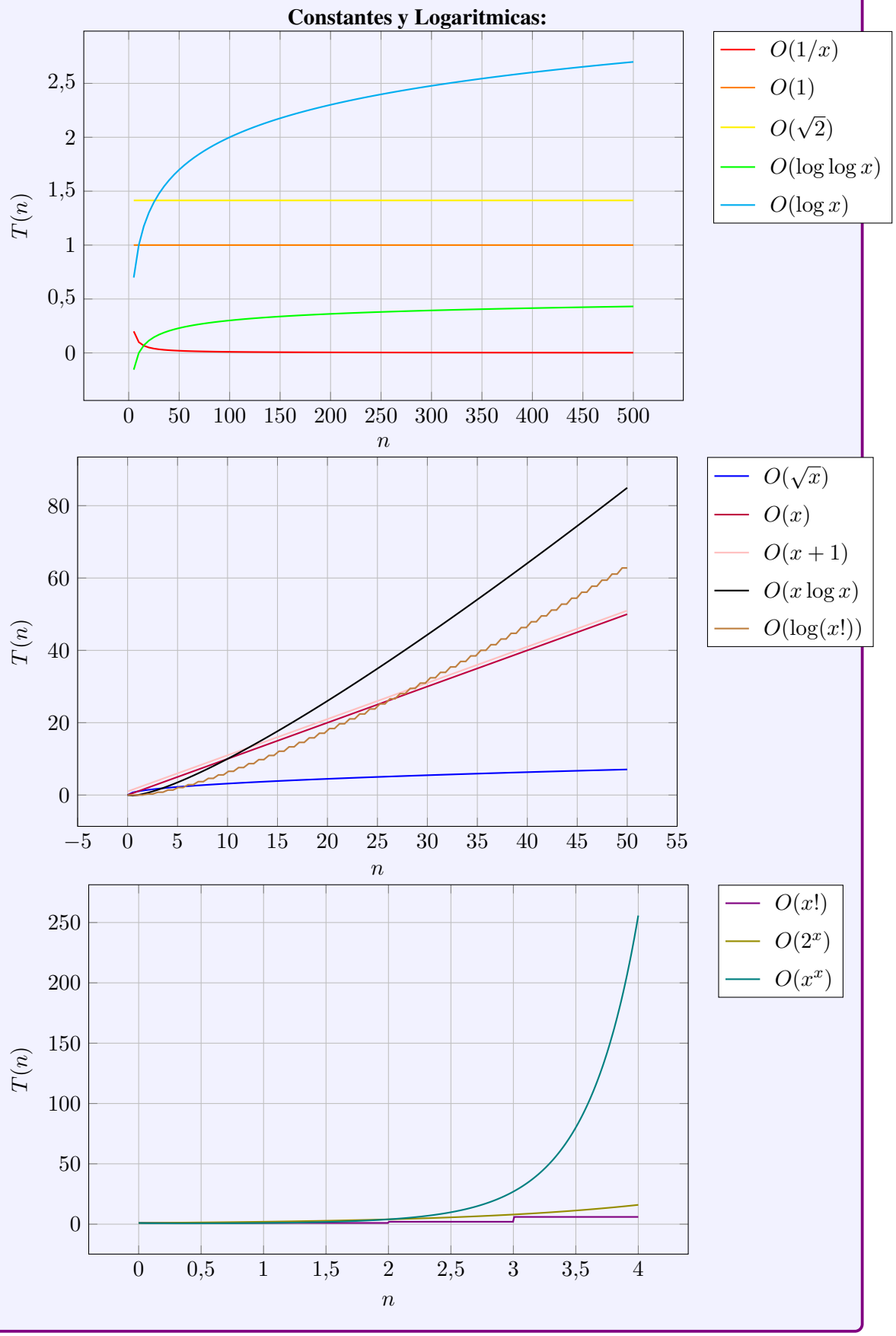
$$\forall h(n) \in O(g(n)) \implies h(n) \in O(f(n))$$

Luego, si hayamos una función $h(n) \in O(g(n))$ pero $h(n) \notin O(f(n))$, se contradice la suposición inicial: $O(f(n)) \supseteq O(g(n))$.

$\therefore$ Dado que ambos casos conducen a una contradicción, entonces $O(f(n)) \subseteq O(g(n)) \wedge O(f(n)) \supseteq O(g(n))$ pero $O(f(n)) \neq O(g(n))$ es **falso**. Por lo tanto, necesariamente debe cumplirse que $O(f(n)) = O(g(n))$.

en el caso 2 podría decirse directamente ”de forma análoga”para no repetir la explicacion

Resolución ejercicio 1.3 (b) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido



++

Resolución ejercicio 1.3 (b) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

**Simbólicamente:**

$$O(1/x) \subseteq O(\log\log x) \subseteq O(1) \subseteq O(\sqrt{2}) \subseteq O(\log x) \subseteq O(\sqrt{x}) \subseteq O(x) \subseteq O(x+1) \subseteq O(\log(x!)) \subseteq O(x\log x) \subseteq O(\sqrt{x}) \subseteq O(x!) \subseteq O(2^x) \subseteq O(x^x)$$

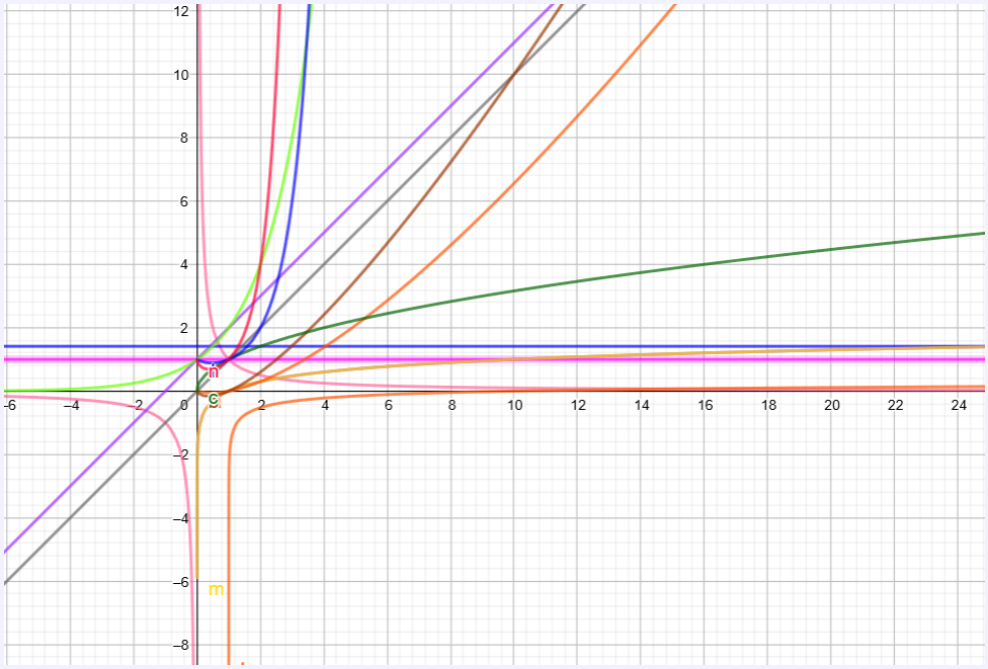



Figura 0.1: Representación gráfica de las funciones

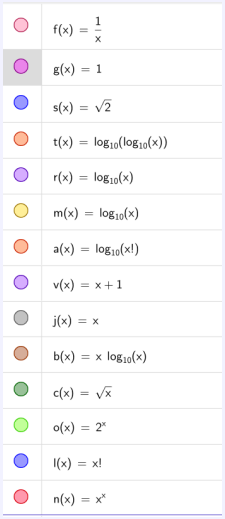



Figura 0.2: Referencias

se recomienda graficar sólo valores positivos. Las agrupaciones de gráfica están buenas, la tercer agrupación es interesante, ver si otra tiene interés en algún rango de valores para x. El orden de las funciones es correcto. Dar el rango de valores para los cuales pudieron graficar y por qué no pudieron graficar en otro rango, o por qué eligieron no graficar en otro rango (por ejemplo no tiene sentido por la forma de la funciòn que ya se sabe que es mayor qu eotra por definiciòn o teorema alguno.). Muy bien presentado y explicado.

4. ¿Por qué hay un error en la siguiente expresión? $O(f(n)) - O(f(n)) = 0$ Además, ¿cómo debería ser realmente el miembro derecho de la igualdad anterior?

Ejercicio 1.4- Grupo 7

**Definición de Orden:**

Sea $g : \mathbb{N} \rightarrow [0, \infty)$ . Se define el conjunto de funciones de orden O (Omicron) de g como: $O(g(n)) = \{t : \mathbb{N} \rightarrow [0, \infty) | \exists c \in \mathbb{R}, c > 0, \exists n_0 \in \mathbb{N} : t(n) \leq c \cdot g(n), \forall n \geq n_0\}$

De la definición se desprende que el Orden $O(g(n))$ representa un conjunto de funciones que cumplen con la condición tener una cota superior en común. Debido a esto, en la expresión en análisis tenemos la diferencia entre dos conjuntos iguales, cuyo resultado no es 0, sino el conjunto vacío, valor que debe ocupar el miembro derecho de la igualdad.

- Expresión original: $O(f(n)) - O(f(n)) = 0$

- Expresión correcta: $O(f(n)) - O(f(n)) = \varnothing$

ok. explicado y resuelto

## 2. Orden Exacto: Θ

1. ¿Qué significa que el tiempo de ejecución de un algoritmo está *en el orden exacto de* $f(n)$?

Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris

**Definición:**
Que esté en el orden exacto de f(n), significa que está acotada tanto superior como inferiormente por esa función. Las funciones cota inferior y superior las denominamos Θ(g(n)).

revisar lo que explicaron. Porque hablan de una función y otra función per ono se entiende bien a cuál refiere. Podrían por ejemplo dar la definición de orden exacto y citar una función que definen como la resultante del algoritmo y ver que pasa con esa función segun el enunciado.

2. Demostrar que:

a) $T(n) = 5 \cdot 2^n + n^2$ está en el orden exacto de $2^n$ .

**Resolución ejercicio 2.2.a - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda**

Demostrar que:
$T(n) = 5 \cdot 2^n + n^2$ está en el orden exacto de $2^n$ .

Sean $f(n) = 5 \cdot 2^n + n^2$ y $g(n) = 2^n$

Entonces
$\lim_{x\to\infty} \frac{f(n)}{g(n)}$

Reemplazamos $f(n)$ y $g(n)$

$= \lim_{n\to\infty} \frac{5\cdot 2^n + n^2}{2^n}$

$= \lim_{n\to\infty} \frac{2^n \cdot 5 + n^2}{2^n}$

$= \lim_{n\to\infty} \frac{2^n \cdot (5 + \frac{n^2}{2^n})}{2^n}$ *(Simplifico $2^n$)*

$= \lim_{n\to\infty} 5 + \frac{n^2}{2^n}$

$= \lim_{n\to\infty} 5 + \lim_{n\to\infty} \frac{n^2}{2^n}$

$= 5 + \lim_{n\to\infty} \frac{2\cdot n}{2^n \cdot ln2}$ *(Aplico L'Hopital)*

$= 5 + \lim_{n\to\infty} \frac{2}{2^n \cdot ln2 \cdot ln2}$ *(Agrupa terminos semejantes)*

$= 5 + \lim_{n\to\infty} \frac{2}{2^n \cdot (ln2)^2}$ *(limite tiende a 0)*

$= 5 + 0 = 5$

Por propiedad 8 de Orden tenemos que si el valor $k = 5 \neq 0$ y $k < \infty$ entonces podemos decir que $\Theta(5 \cdot 2^n + n^2) = \Theta(2^n)$.

revisar el objetivo de la prueba, formalizar. Cuando dice Entonces está incompleto. Cuando justifican pro propiedad 8 debería ser de orden exacto o Theta. Revisar la conclusión en función de la hipótesis / tesis de la prueba

b) $T(n) = n^3 + 9 * n^2 * log(n)$ está en el orden exacto de $n^3$.

**Resolución ejercicio 2.2.b - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda**

Para saber si $T(n)$ esta en el orden exacto de $O(n^3)$:

$$\lim_{n\to\infty} \frac{T(n)}{n^3} = k, k \neq 0 \wedge k < \infty$$

$$\begin{aligned}
\lim_{n\to\infty} \frac{T(n)}{n^3} &= \lim_{n\to\infty} \frac{n^3 + 9 \cdot n^2 \cdot \log n}{n^3} && \textit{(Reemplazamos T(n))} \\
&= \lim_{n\to\infty} \frac{n^2(n + 9 \cdot \log n)}{n^3} && \textit{(Factor común)} \\
&= \lim_{n\to\infty} \frac{n + 9 \cdot \log n}{n} && \textit{(Cancelar } n^2\textit{)} \\
&= \lim_{n\to\infty} \frac{n}{n} + \frac{9 \cdot \log n}{n} && \textit{(Distribución de la suma)} \\
&= 1 + \lim_{n\to\infty} \frac{9 \cdot \log n}{n} && \textit{(Propiedad de limite de una constante)} \\
&= 1 + 9 \cdot \lim_{n\to\infty} \frac{\log n}{n} && \textit{(Propiedad de limite de una constante)} \\
&= 1 + 9 \cdot 0 && \textit{Propiedad: } \lim_{x\to\infty} \frac{\log_a(x)}{x^n} = 0, n \in N \\
&= 1
\end{aligned}$$

$$\lim_{n\to\infty} \frac{T(n)}{n^3} = 1, 1 \neq 0 \wedge 1 < \infty$$

$\therefore T(n)$ esta en el orden exacto de $n^3$

definir las funcines sobre las cuales aplican el límite. Definir bien el objetivo de la prueba. Bien la conclusión.

3. Indicar, respondiendo si/no, para cada par de expresiones *(A, B)* de la siguiente tabla, si *B* es $O, \Omega$ o $\Theta$ de *A*. Suponer que $k \geq 1, \epsilon > 0$ y $c > 1$ son constantes.

**Ejercicio 2.3 - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin**

Utilizamos la regla del límite:

Sea $f, g : \mathbb{N} \to \mathbb{R}^+$ se cumple que:

a) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} \in \mathbb{R}$, entonces $f(n) \in O(g(n)) \wedge f(n) \in \Omega(g(n)) \wedge f(n) \in \Theta(g(n))$

b) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, entonces $f(n) \notin O(g(n)) \wedge f(n) \in \Omega(g(n)) \wedge f(n) \notin \Theta(g(n))$

| item | A | B | $O(B)$ | $\Omega(B)$ | $\Theta(B)$ |
|---|---|---|---|---|---|
| 1 | $log^k n$ | $n^e$ | | | |
| 2 | $n^k$ | $c^n$ | | | |
| 3 | $\sqrt{n}$ | $n^{sin\ n}$ | | | |
| 4 | $2^n$ | $2^{n/2}$ | | | |
| 5 | $n^{log\ c}$ | $c^{log\ n}$ | | | |
| 6 | $log\ n$ | $lg\ n^n$ | | | |

Cuadro 0.1: Completar con *si* o *no* según *A* sea de *B*

c) Si $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, entonces $f(n) \in O(g(n)) \wedge f(n) \notin \Omega(g(n)) \wedge f(n) \notin \Theta(g(n))$

Suponiendo $k \geq 1, \epsilon > 0$ y $c > 1$ constantes

1) $\lim_{n\to\infty} \frac{(\log n)^k}{n^e} = \lim_{n\to\infty} \frac{(\log n)^k}{e^{e \log n}} = 0$ entonces por $c)$

$\therefore log^k n \in O(n^e) \wedge log^k n \notin \Omega(n^e) \wedge log^k n \notin \Theta(n^e)$

2) $\lim_{n\to\infty} \frac{n^k}{c^n} = 0$ entonces por $c)$

$\therefore n^k \in O(c^n) \wedge n^k \notin \Omega(c^n) \wedge n^k \notin \Theta(c^n)$

3) $\lim_{n\to\infty} \frac{\sqrt{n}}{n^{\sin n}} = \infty$ entonces por $c)$

$\therefore \sqrt{n} \notin O(n^{\sin n}) \wedge \sqrt{n} \in \Omega(n^{\sin n}) \wedge \sqrt{n} \notin \Theta(n^{\sin n})$

4) $\lim_{n\to\infty} \frac{2^n}{2^{n/2}} = \lim_{n\to\infty} \frac{2^n}{\sqrt{2^n}} = \infty$ entonces por $b)$

$\therefore 2^n \notin O(2^{n/2}) \wedge 2^n \in \Omega(2^{n/2}) \wedge 2^n \notin \Theta(2^{n/2})$

5) $\lim_{n\to\infty} \frac{n^{\log c}}{c^{\log n}} = \lim_{n\to\infty} \frac{n^{\log c}}{n^{\log c}} = 1$ entonces por $a)$

$\therefore n^{\log c} \in O(c^{\log n}) \wedge n^{\log c} \in \Omega(c^{\log n}) \wedge n^{\log c} \in \Theta(c^{\log n})$

6) $\lim_{n\to\infty} \frac{\log n}{\log n^n} = \lim_{n\to\infty} \frac{\log n}{n \log n} = 0$ entonces por $c)$

$\therefore \log n \in O(\log n^n) \wedge \log n \notin \Omega(\log n^n) \wedge \log n \notin \Theta(\log n^n)$

completar la tabla con los resultados obtenidos. Yo (Naty) les completo numerada la fila de la tabla para que puedan referir de forma enumerada a la demostración de cada respuesta

4. Demostrar las siguientes propiedades:

a) (transitiva) $f(n) \in \Omega(g(n))\ y\ g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$

Resolución 2.4.a - Grupo 2: Sarmiento, Fernandez Gramajo, Reibold, Petit, Keller

**Demostración:** Propiedad de transitividad

Para esto asumimos que $f(n) \in \Omega(g(n))\ y\ g(n) \in \Omega(h(n))$ es Verdadero.

Se tiene que demostrar que $f(n) \in \Omega(h(n))$ también lo es.

Planteamos Hipótesis con *Definición de* $\Omega$ :

- $f(n) \in \Omega(g(n)) \Leftrightarrow \exists c_0 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n_0,\ f(n) \geq c_0 \cdot g(n)$ ***(1)***
- $g(n) \in \Omega(h(n)) \Leftrightarrow \exists c_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n_1,\ g(n) \geq c_1 \cdot h(n)$ ***(2)***

*Planteamos Tesis con Definición de* $\Omega$ :

- $f(n) \in \Omega(h(n)) \Leftrightarrow \exists c_2 \in \mathbb{R}^+, \exists n_2 \in \mathbb{N}, \forall n \in \mathbb{N}, n \geq n_2,\ f(n) \geq c_2 \cdot h(n)$ **(3)**

Partimos de **(1)**:

- $f(n) \geq c_0 \cdot g(n)$

Reemplazo $g(n)$ por **(2)**:

- $f(n) \geq c_0 \cdot c_1 \cdot h(n)$

Por lo tanto:
$\exists c_2 = c_0 \cdot c_1 \in \mathbb{R}^+,\ \exists n_2 = max(n_0, n_1) \in \mathbb{N} \mid \forall n \in \mathbb{N},\ n \geq n_2, f(n) \geq c_2 \cdot h(n)$

Llegando a **(3)** y quedó demostrado que $f(n) \in \Omega(h(n))$ es Verdadero.

$\therefore$ f(n) $\in \Omega(g(n))$ y $g(n) \in \Omega(h(n)) \Rightarrow f(n) \in \Omega(h(n))$

b) (simetría)$f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$

Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris

Para probar la propiedad de simetría la hacemos en dos pasos:

- ($\Rightarrow$) Si $f(n) \in \Theta(g(n))$ entonces $g(n) \in \Theta(f(n))$ **(I)**
- ($\Leftarrow$) Si $g(n) \in \Theta(f(n))$ entonces $f(n) \in \Theta(g(n))$ **(II)**

---

**Empezaremos probando (I)**

Asumimos a $f(n) \in \Theta(g(n))$ como verdadera y lo reescribimos utilizando la definición de orden exacto

- $f(n) \in \Theta(g(n)) \Leftrightarrow \exists c_0, d_0 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : c_0 \cdot g(n) \leq f(n) \leq d_0 \cdot g(n)$ **(1)**

Luego queremos llegar a que se cumpla que $g(n) \in \Theta(f(n))$, es decir

- $g(n) \in \Theta(f(n)) \Leftrightarrow \exists c_1, d_1 \in \mathbb{R}^+, \exists n_1 \in \mathbb{N}, \forall n \geq n_1 : c_1 \cdot f(n) \leq g(n) \leq d_1 \cdot f(n)$ **(2)**

En (1) por un lado tenemos que

- $c_0 \cdot g(n) \leq f(n)$
- $g(n) \leq \frac{1}{c_0} \cdot f(n)$
- $g(n) \leq d_1 \cdot f(n)$ **(3)** (Con $d_1 = \frac{1}{c_0} \in \mathbb{R}^+$)

Y por otro lado, en (1)

- $f(n) \leq d_0 \cdot g(n)$
- $\frac{1}{d_0} \cdot f(n) \leq g(n)$
- $c_1 \cdot f(n) \leq g(n)$ **(4)** (Con $c_1 = \frac{1}{d_0} \in \mathbb{R}^+$)

$\therefore$ Usando (3) y (4) llegamos a que $c_1 \cdot f(n) \leq g(n) \leq d_1 \cdot f(n)$ y así queda probado que (2) se cumple (es decir, $g(n) \in \Theta(f(n))$)
Por consecuencia (I) es verdadero

---

**Ahora probaremos (II)**

La prueba de esto es exactamente idéntica a la que realizamos en (I), con la única salvedad de que tenemos que reemplazar cada aparición de $f(n)$ por $g(n)$ y viceversa

$\therefore$ (II) es verdadero

---

Luego como (I) y (II) se cumplen, queda probada la propiedad de simetría
Es decir, $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$

c) (regla de dualidad) $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega(f(n))$

Resolución ejercicio 2.4.c - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda

## Punto 2.4.c

Demostrar la regla de dualidad:
$f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega f(n)$
**Primera parte:** Demostrar que $f(n) \in O(g(n)) \Rightarrow g(n) \in \Omega f(n)$

1. $f(n) \in O(g(n))$
2. $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} | \forall n \in \mathbb{N}, n > n_0, \mathbf{f(n)} \leq \mathbf{c} \cdot \mathbf{g(n)}$ Definición de la relación pertenece a Omicron
3. $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} | \forall n \in \mathbb{N}, n > n_0, \mathbf{\frac{1}{c}} \cdot \mathbf{f(n)} \leq \mathbf{g(n)}$ multiplicar por 1/c en ambos miembros
4. $\exists d \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} | \forall n \in \mathbb{N}, n > n_0, \mathbf{d} \cdot \mathbf{f(n)} \leq \mathbf{g(n)}, d = \frac{1}{c}$ Redefinir como d a 1/c
5. $g(n) \in \Omega f(n)$ Definición de pertenece a Omega

**Segunda parte:** Demostrar que $g(n) \in \Omega f(n) \Rightarrow f(n) \in O(g(n))$

1. $g(n) \in \Omega f(n)$
2. $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} | \forall n \in \mathbb{N}, n > n_0, \mathbf{g(n)} \geq \mathbf{c} \cdot \mathbf{f(n)}$ Definición de la relación pertenece a Omega
3. $\exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} | \forall n \in \mathbb{N}, n > n_0, \mathbf{\frac{1}{c}} \cdot \mathbf{g(n)} \geq \mathbf{f(n)}$ multiplicar por 1/c en ambos miembros
4. $\exists d \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} | \forall n \in \mathbb{N}, n > n_0, \mathbf{d} \cdot \mathbf{g(n)} \geq \mathbf{f(n)}, \mathbf{d} = \mathbf{\frac{1}{c}}$ Redefinir a d como 1/c
5. $f(n) \in O(g(n))$ Definición de la relación pertenece a Omicron

$\therefore$ como se cumple (5) en ambos casos entonces se cumple que $f(n) \in O(g(n)) \Leftrightarrow g(n) \in \Omega f(n)$

5. Sean $P_1$ y $P_2$ dos programas cuyos tiempos de ejecución son $T_1(n)$ y $T_2(n)$ respectivamente donde $n$ es el tamaño de la entrada. Determine para los siguientes casos en qué condiciones $P_2$ se ejecuta más rápido que $P_1$:

   a) $T_1(n) = 2n^2 \qquad T_2(n) = 1000n$

**Resolución ejercicio 2.5 (a) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido**

Debemos determinar el valor de n donde T1 y T2 son iguales para poder analizar cómo se comportan con entradas menores o mayores a ese valor

$$\begin{aligned} T_1(n) &= T_2(n) \\ 2n^2 &= 1000n \\ 2n^2 \cdot \frac{1}{2} &= 1000n \cdot \frac{1}{2} \\ n^2 &= 500n \\ n^2 - 500n &= 0 \\ n(n-500) &= 0 \\ n &= 0 \vee n = 500 \end{aligned}$$

Gráficamente:

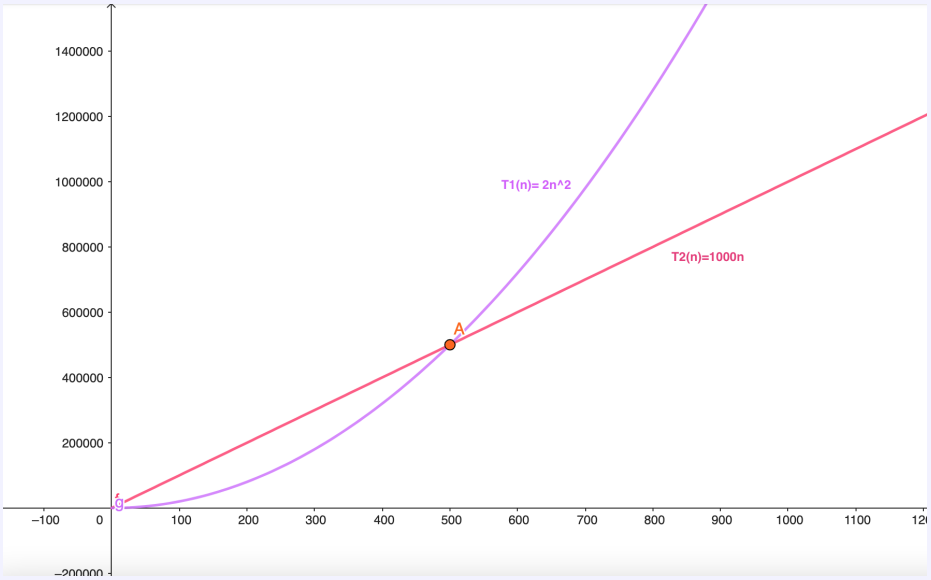



Figura 0.3: Representación gráfica de la intersección entre $T_1$ y $T_2$

$\therefore$ Cuando $n > 500$ $P_2$ se ejecuta más rápido que $P_1$

b) $T_1(n) = 3n^4 \qquad T_2(n) = 3n^3$

**Resolución 2.5.b - Grupo 2: Sarmiento, Fernandez Gramajo, Reibold, Petit, Keller**

Para determinar esto debemos evaluar el $n_0 \in \mathbb{N}$ que verifica $T_1(n) = T_2(n)$

$$3n^4 = 3n^3$$
$$n^4 = n^3$$
$$n^3 \cdot (n-1) = 0$$

$$n^3 = 0 \quad \vee \quad n - 1 = 0 \implies n = 0 \quad \vee \quad n = 1$$

Por el gráfico, podemos decir que $\exists n_0 = 1 \in \mathbb{N}$ tal que se ejecuta P2 más rápido que P1

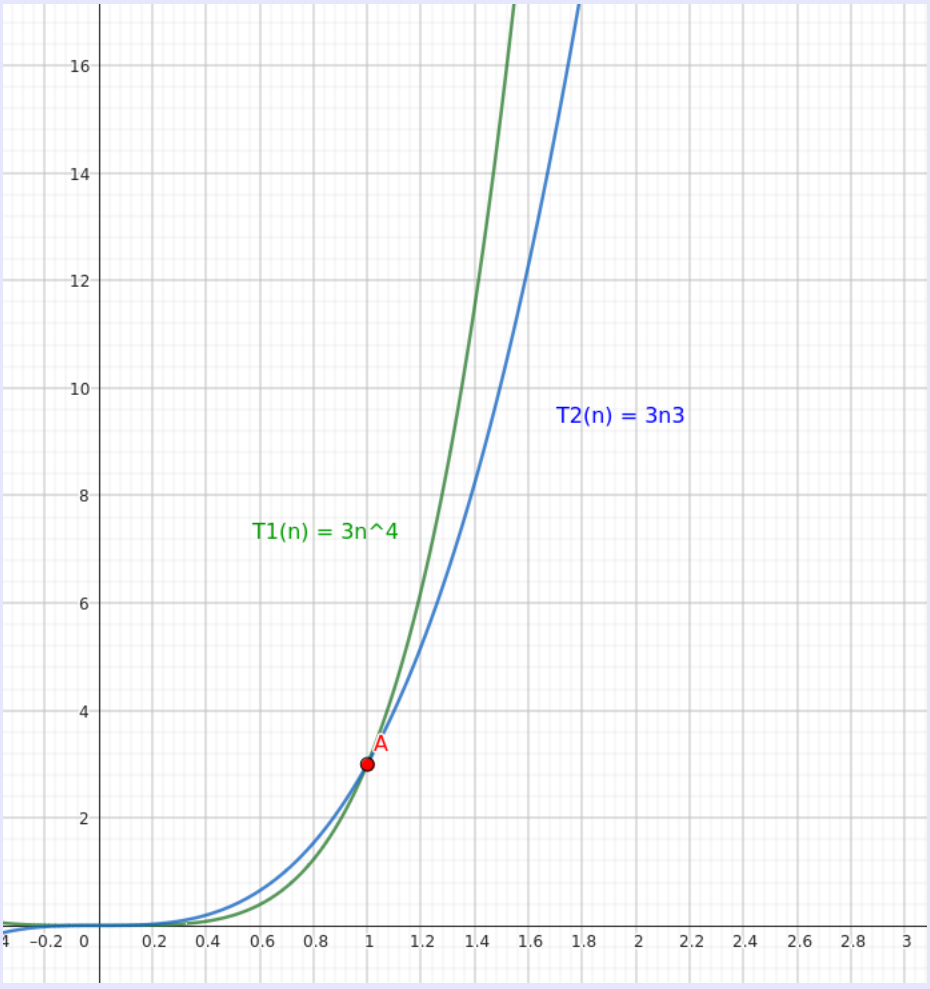



Figura 0.4: Representación gráfica de la intersección entre $T_1$ y $T_2$

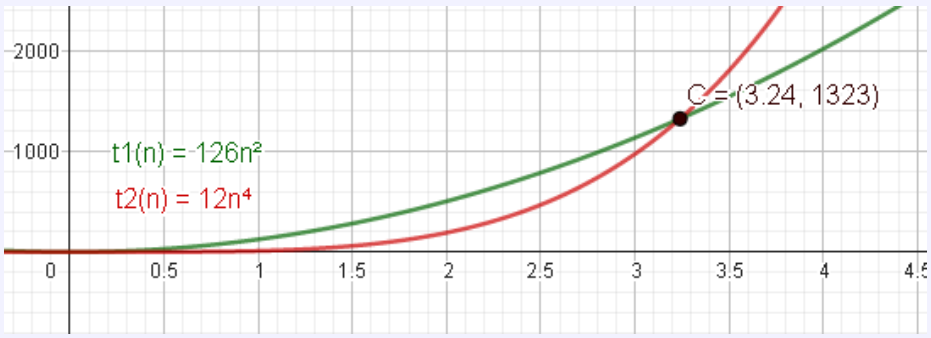
c) $T_1(n) = 126n^2$ $T_2(n) = 12n^4$

Ejercicio 2.5.c

**Resolución:**
Para determinar en qué condiciones un programa $P_2$ se ejecuta más rápido que otro programa $P_1$, necesitamos comparar sus tiempos de ejecución. En este caso, dado que los tiempos de ejecución de los programas están representados por las funciones $T_1(n)$ y $T_2(n)$ respectivamente, la tarea consiste en encontrar los valores de $n$ para los cuales $T_2(n)$ es menor que $T_1(n)$.
Para hacer esta comparación de manera efectiva, debemos encontrar los puntos en los cuales los tiempos de ejecución son iguales, es decir, debemos igualar:

$$
\begin{aligned}
T_1(n) &= T_2(n) \\
126n^2 &= 12n^4 \\
0 &= 12n^4 - 126n^2 \\
0 &= 6n^2(2n^2 - 21) \\
\therefore \quad 6n^2 = 0 \quad &\vee \quad 2n^2 - 21 = 0 \\
n^2 = 0 \quad &\vee \quad n^2 = \frac{21}{2} \\
n = 0 \quad &\vee \quad n = \pm\sqrt{\frac{21}{2}}
\end{aligned}
$$

Gráfico de $T_1$ y $T_2$:




$\therefore$ Cuando $n > \sqrt{\frac{21}{2}}$, $P_2$ se ejecuta más rápido que $P_1$, es decir, $T_2$ es una cota superior de $T_1$.

6. Considere las siguientes funciones $f : \mathbb{N}^0 \to \mathbb{R}^+$ :

$f_1(n) = 2 * n^5 - 16 * n^3$

$f_2(n) = 2 * n^5$

$$f_3(n) = \begin{cases} 2 * n^3 & si\ n\ es\ par \\ 5 * n^4 & si\ n\ es\ impar \end{cases}$$

Demuestre si se cumple que $f_1(n) \in O(f_2(n))$, y que $f_3(n)$ es $\Omega(n^4)$.

**Ejercicio 2.6 - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani**

Sean:

- $f_1(n) = 2 * n^5 - 16 * n^3$
- $f_2(n) = 2 * n^5$
- $f_3(n) = \begin{cases} 2 * n^3 & si\ n\ es\ par \\ 5 * n^4 & si\ n\ es\ impar \end{cases}$

Para probar que $f_1 \in O(f_2(n))$, utilizamos la regla del limite:

$$\lim_{n\to\infty} \frac{f_1(n)}{f_2(n)} \in \mathbb{R} \text{ entonces } f_1(n) \in O(g(n)) \wedge f_2(n) \notin O(f_1(n))$$

Desarrollo:
Evaluando el limite:

$$\lim_{n\to\infty} \frac{2 * n^5 - 16 * n^3}{2 * n^3} = \frac{\infty}{\infty}\ indeterminado$$

Transformamos la expresión realizando factor común $2 * n^3$

$$\lim_{n\to\infty} \frac{2 * n^5 - 16 * n^3}{2 * n^3} = \lim_{n\to\infty} \frac{(2 * n^3) * (n^2 - 8)}{2 * n^3} =$$

$$\lim_{n\to\infty} n^2 - 8 = \infty$$

$$\therefore \lim_{n\to\infty} \frac{f_1(n)}{f_2(n)} = \infty \text{ entonces } f_1(n) \notin O(f_2(n)) \wedge f_2(n) \in O(f_1(n))$$

**Ejercicio 2.6 - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani**

Para probar que $f_3 \in \Omega(n^4)$, utilizamos la regla del limite:
Evaluamos el limite por partes donde:

- $g_1(n) = 2 * n^3$
- $g_2(n) = 5 * n^4$

Desarrollo:

Evaluando el limite:

$$\lim_{n\to\infty} \frac{2*n^3}{n^4} = \frac{\infty}{\infty}\ indeterminado$$

Transformamos la expresión realizando factor común $n^3$

$$\lim_{n\to\infty} \frac{2*n^3}{n^4} = \lim_{n\to\infty} \frac{(n^3)*(2)}{n^4} =$$

$$\lim_{n\to\infty} \frac{2}{n} = 0$$

$$\therefore \lim_{n\to\infty} \frac{g_1(n)}{n^4} = 0 \text{ entonces } g_1(n) \in O(n^4) \wedge g_1(n) \notin \Omega(n^4)$$

Evaluando el limite:

$$\lim_{n\to\infty} \frac{5*n^4}{n^4} = \frac{\infty}{\infty}\ indeterminado$$

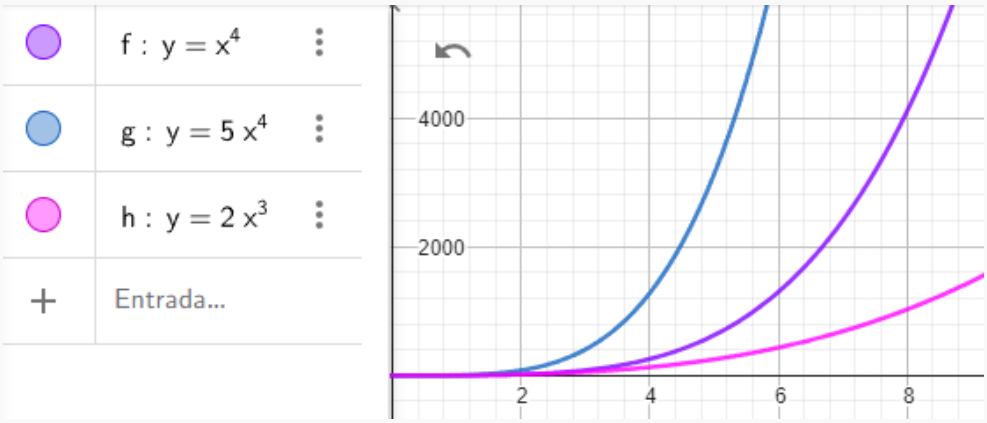
Simplificamos $n^4$

$$\lim_{n\to\infty} \frac{5*n^4}{n^4} = \lim_{n\to\infty} 5 = 5$$

$$\therefore \lim_{n\to\infty} \frac{g_2(n)}{n^4} = 5 \text{ entonces } g_2(n) \in O(n^4) \wedge g_2(n) \in \Omega(n^4)$$

$\therefore$ *Como* $g_1(n) \notin \Omega(n^4)$ y $g_2(n) \in \Omega(n^4) \Rightarrow f_3(n) \notin \Omega(n^4)$

Graficamente




## 3. Algoritmia en Algoritmos iterativos

1. Considere el siguiente algoritmo para encontrar la distancia más corta entre dos elementos del arreglo de números:

```
1  MODULO distMin (ARREGLO ENTERO a, ENTERO n) RETORNA ENTERO
2  //Entrada: arreglo de n numeros
3  //Salida: Minima distancia entre dos elementos cualesquiera del arreglo.
4      dmin ← ∞
5      PARA i ← 0 HASTA n-1 HACER
6          PARA j ← 0 HASTA n-1 HACER
7              if (i! = j y a[i] − a[j]) < dmin ENTONCES
8                  dmin ← |a[i] − a[j]|
9      RETORNA dmin
10 FIN ALGORITMO
```

a) Realiza tantas mejoras como consideres necesario al pseudocódigo anterior, tanto de eficiencia como de sintaxis y semántica del algoritmo. Por ejemplo, intenta minimizar el número de comparaciones entre elementos.

b) Analizar el tiempo de ejecución teórico de la versión original y de la versión mejorada y verificarlo empíricamente.

c) Responda a las siguientes preguntas

(i) Exprese el tiempo teórico en notación $\Theta(f(n))$ ¿Por qué en este caso es mejor utilizar $\Theta$ que $O$?

(ii) A partir de sus pruebas empíricas, podría decir cuál es el valor $n_0$ para el cual la definición de $\Theta$ ocurre.

**Resolución ejercicio 3.1 (a)** - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

**Código corregido:** El código se corrigió en cuanto a sintaxis y semántica, también se cambio la forma del recorrido para que sea eficiente y conserve el mismo orden.

```
1  ALGORITMO distMin(ARREGLO ENTERO a) {
2      ENTERO n ← longitud(a)
3      ENTERO distancia ← 0
4      ENTERO dmin ← ∞
5      PARA ENTERO i ← 0 HASTA n - 2 HACER
6          PARA ENTERO j ← i + 1 HASTA n-1 HACER
7              distancia ← |a[i] - a[j]|
8              SI (distancia < dmin) ENTONCES
9                  dmin ← distancia;
10             FIN SI
11         FIN PARA
12     FIN PARA
13 FIN ALGORITMO
```

**Código Mejorado:** Como el algoritmo original consiste en retornar la distancia en un arreglo, se planteó que lo más eficiente sería ordenar el arreglo con mergeSort, y una vez ordenado, realizar un solo recorrido al arreglo y encontrar la distancia más corta.

```
1  ALGORITMO distanciaMinima (ARREGLO ENTERO a) RETORNA ENTERO
2      mergeSort(a)
3      ENTERO largo    ← longitud(a)
4      ENTERO dmin     ← ∞
5      ENTERO dist     ← 0
6      PARA i ← 1 HASTA largo-1 HACER
7          dist ← |a[i-1]-a[i]|
8          SI (dist<dmin) ENTONCES
9              dmin ← dist
10         FIN SI
11     FIN PARA
12     RETORNA dmin
13 FIN ALGORITMO
14
15 MODULO mergeSort (ARREGLO ENTERO a) RETORNA ∅
16     mergeSort (a,0,longitud(a)-1)
17 FIN MODULO
18
19 MODULO mergeSort (ARREGLO ENTERO a, ENTERO izq, ENTERO der) RETORNA ∅
20     ENTERO largo ← arreglo.length;
21         SI (largo > 1) ENTONCES
22             ENTERO          medio       ← largo / 2;
23             ARREGLO ENTERO  izquierda   ← copiarEnRango(arreglo, 0,
                   medio)
24             ARREGLO ENTERO  derecha     ← copiarEnRango(arreglo, medio,
                   largo)
25             mergeSort(izquierda)
26             mergeSort(derecha)
27             merge(arreglo, izquierda, derecha)
28         FIN SI
29 FIN MODULO
30
31 MODULO merge(ARREGLO ENTERO arreglo, ARREGLO ENTERO izquierda, ARREGLO
       ENTERO derecha) RETORNA ∅
32     ENTERO indiceIzq    ← 0
33     ENTERO indiceDer    ← 0
34     ENTERO indiceArr    ← 0
35     ENTERO largoIzq     ← longitud(izquierda)
36     ENTERO largoDer     ← longitud(length)
37     MIENTRAS (indiceIzq < largoIzq AND indiceDer < largoDer) HACER
38             SI (izquierda[indiceIzq] <= derecha[indiceDer]) ENTONCES
39                 arreglo[indiceArr] ← izquierda[indiceIzq]
40                 indiceArr ← indiceArr+1
41                 indiceIzq ← indiceIzq+1
42             SINO
43                 arreglo[indiceArr] ← derecha[indiceDer]
44                 indiceArr ← indiceArr+1
45                 indiceDer ← indiceDer+1
```

```
46              FIN SI
47
48          MIENTRAS (indiceIzq < largoIzq) HACER
49              arreglo[indiceArr] ← izquierda[indiceIzq]
50              indiceArr ← indiceArr+1
51                  indiceIzq ← indiceIzq+1
52          FIN MIENTRAS
53
54          MIENTRAS (indiceDer < largoDer) HACER
55              arreglo[indiceArr] ← derecha[indiceDer]
56               indiceArr ← indiceArr+1
57                  indiceDer ← indiceDer+1
58          FIN MIENTRAS
59  FIN MODULO
60
61  MODULO copiarEnRango(ARREGLO ENTERO a, ENTERO desde, ENTERO hasta)
        RETORNA ARREGLO ENTERO
62      ENTERO[] retorno ←new ENTERO[hasta - desde];
63      PARA ENTERO i ← 0 HASTA longitud(retorno) HACER
64          retorno[i] ← a[desde]
65          desde ← desde+1
66      FIN PARA
67      RETORNA retorno;
68  FIN MODULO
```

Resolución ejercicio 3.1 (b) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

**Algoritmo original:**

```
1  MODULO distMin (ARREGLO ENTERO a, ENTERO n) RETORNA ENTERO
2  //Entrada: arreglo de n numeros
3  //Salida: Minima distancia entre dos elementos cualesquiera del
       arreglo.
4       dmin ← ∞
5       PARA i ← 0 HASTA n-1 HACER
6           PARA j ← 0 HASTA n-1 HACER
7               if (i! = j y |a[i] − a[j]| < dmin) ENTONCES
8                   dmin ← |a[i] − a[j]|
9       RETORNA dmin
10 FIN ALGORITMO
```

**Tiempo teórico:**

$T_4 = 1$

$T_5 = Tini_5 + cantIt \cdot (Tcond_5 + Tint_5 + Tinc_5) + Tcond_5$

$T_5 = 1 + (n-1) \cdot (1 + 18 \cdot n - 15 + 2) + 1$

$T_5 = 2 + (n-1) \cdot (18 \cdot n - 12)$

$T_5 = 2 + 18 \cdot n \cdot n - 12 \cdot n - 18 \cdot n + 12 = \mathbf{14 - 30 \cdot n + 18 \cdot n^2}$

- $Tini_5 = 1$
- $cantIt = \frac{n-1-0}{1} = n - 1$
- $Tcond_5 = 1$
- $Tint_5 = T_6 = 18 \cdot n - 15$
- $Tinc_5 = 2$
- $T_6 = Tini_6 + cantIt \cdot (Tcond_6 + Tint_6 + Tinc_6) + Tcond_6$
- $T_6 = 1 + (n-1) \cdot (1 + 16 + 2) + 2$
- $T_6 = 3 + (n-1) \cdot 18$
- $T_6 = 18 \cdot n - 15$
  - $Tini_6 = 1$
  - $cantIt = \frac{n-1-0}{1} = n - 1$
  - $Tcond_6 = 1$
  - $Tint_6 = T_7 = 16$
  - $Tinc_6 = 2$
  - $T_7 = Tcond_7 + Tint_7$
  - $T_7 = 9 + 7 = 16$
    - $Tcond_7 = ((1+1) + 1 + ((2) + 1 + (2))) + 1$
    - $Tcond_7 = (2) + 1 + (5) + 1 = 9$
    - $Tint_7 = ((2) + 1 + (2) + 1) + 1 = 6 + 1 = 7$

$T_9 = 1$

$T_{original} = T_4 + T_5 + T_9$

$T_{original} = 1 + (14 - 30 \cdot n + 18 \cdot n^2) + 1$

$T_{original} = 16 - 30 \cdot n + 18 \cdot n^2$

Luego, el tiempo teórico del algoritmo original es: $16 - 30 \cdot n + 18 \cdot n^2$

Resolución ejercicio 3.1 (b) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

**Algoritmo corregido:**

```
ALGORITMO distMin(ARREGLO ENTERO a) {
    ENTERO n ← longitud(a)
    ENTERO distancia ← 0
    ENTERO dmin ← ∞
    PARA ENTERO i ← 0 HASTA n - 2 HACER
        PARA ENTERO j ← i + 1 HASTA n-1 HACER
            distancia ← |a[i] - a[j]|
            SI (distancia < dmin) ENTONCES
                dmin ← distancia;
            FIN SI
        FIN PARA
    FIN PARA
FIN ALGORITMO
```

**Tiempo teórico:**

$T_2 = 2$
$T_3 = 1$
$T_4 = 1$
$T_5 = Tini_5 + cantIt_5 \cdot (Tcond_5 + Tinc_5) + Tint_5$
$T_5 = 1 + (n-1) \cdot (2+2) + 2 + \frac{(n-1) \cdot n}{2} \cdot 12$
$T_5 = 3 + (n-1) \cdot 4 + 6 \cdot (n^2 - n)$
$T_5 = 3 + 4 \cdot n - 4 + 6 \cdot n^2 - 6 \cdot n$
$T_5 = -1 - 2 \cdot n + 6 \cdot n^2$

$Tini_5 = 1$
$Tcond_5 = 1 + 1 = 2$
$Tint_5 = T6 = 2 + \frac{(n-1) \cdot n}{2} \cdot 12$
$Tinc_5 = 2$
$cantIt_5 = \frac{n-1-0}{1} = n - 1$
$T_6 = Tini_6 + cantIt_6 \cdot (Tcond_6 + Tint_6 + Tinc_6) + (n - 1 - (n-1)) \cdot Tcond_6$
$T_6 = 2 + \frac{(n-1) \cdot n}{2} \cdot (2 + 8 + 2) + 0 \cdot Tcond_6$
$T_6 = 2 + \frac{(n-1) \cdot n}{2} \cdot 12$

$Tini_6 = 1 + 1 = 2$
$Tcond_6 = 1 + 1 = 2$
$Tint_6 = T_7 = 8$
$Tinc_6 = 2$
$cantIt_6 = \sum_{i=0}^{n-1} [n - (i+1)] = (n-1) + (n-2) + ... + 1 = \frac{(n-1) \cdot n}{2}$
$T_7 = (2 + 1 + 2) + 1 + T8 = 6 + 2 = 8$

$T_8 = 1 + T9 = 1 + 1 = 2$

$T_9 = 1$

$T_{corregido} = T_2 + T_3 + T_4 + T_5$
$T_{corregido} = 2 + 1 + 1 - 1 - 2 \cdot n + 6 \cdot n^2$
$T_{corregido} = 3 - 2 \cdot n + 6 \cdot n^2$

Luego, el tiempo teórico del algoritmo corregido es: $3 - 2 \cdot n + 6 \cdot n^2$

Resolución ejercicio 3.1 (b) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido

**Tiempos empíricos:**

Se realizó una prueba tipo Batch, de 100 intentos con un arreglo de 10000 enteros random entre 0 y 5000. Se presentan a continuación los tiempos máximo, mínimo y promedio:

**Algoritmo original:**

Cantidad de pruebas = 100
Cantidad de elementos = 10000
_____
Tiempo Máximo = 305776900 ns = 0,30577690 s.
Tiempo Mínimo = 96340300 ns = 0,0963403 s.
Tiempo promedio = 1.40689443E8 ns = 0.140689443 s.

**Algoritmo corregido:**

Cantidad de pruebas = 100
Cantidad de elementos = 10000
_____-
Tiempo Máximo = 160094300 ns = 0,1600943 s.
Tiempo Mínimo = 45295600 ns = 0,0452956 s.
Tiempo promedio = 7.477431E7 ns = 0,07477431 s.

**Conclusión:**
La comparación entre los tiempos teóricos calculados muestra que, si bien ambos algoritmos son de orden $O(n^2)$, el corregido tiene un tiempo mejor. Esto es debido a que la constante que multiplica al término cuadrático en el original es 18, mientras que en el corregido es 6, lo que hace que para volúmenes grandes de datos de entrada la diferencia se note.

**Algoritmo original:**
$T_{original} = 16 - 30 \cdot n + 18 \cdot n^2$

**Algoritmo corregido:**
$T_{corregido} = 3 - 2 \cdot n + 6 \cdot n^2$

Al realizar las pruebas empíricas, se verifica la mejora en los tiempos de ejecución del algoritmo mejorado respecto al original. Se evidencia esto dado que todos los valores del algoritmo mejorado son menores al del original.

**Resolución ejercicio 3.1 (c) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido**

- $f(n) = 3 - 2 \cdot n + 6 \cdot n^2$
- $g(n) = n^2$

Por definición:

$$f(n) \in \Theta(g(n)) \Leftrightarrow \exists c_0, d_0 \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \forall n \geq n_0 : c_0 \cdot g(n) \leq f(n) \leq d_0 \cdot g(n)$$

$\therefore f(n) = 3 - 2 \cdot n + 6 \cdot n^2 \in \Theta(n^2)$
Luego, ¿Por qué en este caso es mejor utilizar $\Theta$ que $O$?

Usar $\Theta$ (Theta) es mejor que $O$ (Big-O) cuando quieres describir el comportamiento exacto de un algoritmo, tanto en su crecimiento superior como inferior. Mientras que $O$ solo indica un límite superior (lo máximo que puede crecer), $\Theta$ asegura que el crecimiento es exactamente f(n), acotado por arriba y por abajo.

(ii) A partir de sus pruebas empíricas, podría decir cuál es el valor $n_0$ para el cual la definición de $\Theta$ ocurre.

Para encontrar las constantes $c_1$ y $c_2$ en la definición de $\Theta(f(n))$ empíricamente, seguimos los siguientes pasos:

a) Ejecutamos el algoritmo para varios valores de $n_0$ y registramos los tiempos de ejecución reales $T(n)$.

b) Identificamos la función de crecimiento $f(n)$, en este caso $f(n) = 2 - 2 \cdot n + 6 \cdot n^2$.

c) Calculamos $c_1$ y $c_2$ usando las fórmulas:

$$c_1 = \text{mín}\left(\frac{T(n)}{f(n)}\right), \quad c_2 = \text{máx}\left(\frac{T(n)}{f(n)}\right)$$

donde $T(n)$ es el tiempo de ejecución y $f(n)$ es la función de crecimiento esperada.

Datos y Cálculo de $\frac{T(n)}{f(n)}$

| $n$ | $T(n)$ (ns) | $f(n)$ | $\frac{T(n)}{f(n)}$ |
|---|---|---|---|
| 10 | 5838 | 582 | $\frac{5838}{582} \approx 10{,}03$ |
| 100 | 36814 | 59802 | $\frac{36814}{59802} \approx 0{,}615$ |
| 500 | 123807 | 1499002 | $\frac{123807}{1499002} \approx 0{,}0826$ |
| 1000 | 1715593 | 5998002 | $\frac{1715593}{5998002} \approx 0{,}286$ |
| 5000 | $1.11606 \times 10^7$ | 149990002 | $\frac{1{,}11606 \times 10^7}{149990002} \approx 0{,}0744$ |
| 10000 | $5.8468202 \times 10^7$ | 599980002 | $\frac{5{,}8468202 \times 10^7}{599980002} \approx 0{,}0974$ |

**Resolución ejercicio 3.1 (c) - Grupo 1 - Bugli, Cedeño, Coronel, Ferraris, Guido**

Cálculo de $c_1 \cdot f(n)$ y $c_2 \cdot f(n)$

| $n$ | $T(n)$ (ns) | $c_1 \cdot f(n)$ |
|---|---|---|
| 10 | 5838 | $0{,}0744 \times 582 \approx 43{,}31$ |
| 100 | 36814 | $0{,}0744 \times 59802 \approx 4450{,}65$ |
| 500 | 123807 | $0{,}0744 \times 1499002 \approx 111475{,}73$ |
| 1000 | 1715593 | $0{,}0744 \times 5998002 \approx 445887{,}75$ |
| 5000 | $1.11606 \times 10^7$ | $0{,}0744 \times 149990002 \approx 11166136{,}95$ |
| 10000 | $5.8468202 \times 10^7$ | $0{,}0744 \times 599980002 \approx 44594889{,}48$ |

| $n$ | $T(n)$ (ns) | $c_2 \cdot f(n)$ |
|---|---|---|
| 10 | 5838 | $10{,}03 \times 582 \approx 5837{,}46$ |
| 100 | 36814 | $10{,}03 \times 59802 \approx 599539{,}06$ |
| 500 | 123807 | $10{,}03 \times 1499002 \approx 15034860{,}06$ |
| 1000 | 1715593 | $10{,}03 \times 5998002 \approx 60159720{,}06$ |
| 5000 | $1.11606 \times 10^7$ | $10{,}03 \times 149990002 \approx 1504497006{,}06$ |
| 10000 | $5.8468202 \times 10^7$ | $10{,}03 \times 599980002 \approx 6023798200{,}06$ |

| $n$ | ¿Se cumple que $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$? |
|---|---|
| 10 | No |
| 100 | Sí |
| 500 | Sí |
| 1000 | Sí |
| 5000 | Sí |
| 10000 | Sí |

Conclusión

- Para $c_1$: El valor mínimo de $\frac{T(n)}{f(n)}$ es 0,0744.
- Para $c_2$: El valor máximo de $\frac{T(n)}{f(n)}$ es 10,03.
- El valor de $n_0$ es 100, ya que a partir de $n = 100$ se cumple que $c_1 \cdot f(n) \leq T(n) \leq c_2 \cdot f(n)$.

**Resolución ejercicio 3.1 - Grupo 4: Triñanes, Jan, Wernly, Corrales, Sepulveda**

**Inciso a:**
Para este algoritmo se realizaron las siguientes mejoras:

- Se cambiaron los nombres de las variables para que sea mas fácil identificarlas.
- La variable distancia mínima no inicia desde infinito sino que inicia desde el valor mas

grande que soporta la variable de tipo entero.

- Hay una condición de corte para que cuando la distancia sea 0, el programa no continúe ya que no hay distancia menor.
- Ahora en la segunda iteración siempre comienza a probar los valores del arreglo que están a partir del índice i y no recorre todos los valores 2 veces.

```
1  public static int distanciaMinima(int[] arr, int n){
2      int distanciaMin = Integer.MAX_VALUE;
3      int i=0, j;
4      while(distanciaMin != 0 && i<n){
5          j = i+1;
6          while(distanciaMin != 0 && j<n){
7              if(Math.abs(arr[i] - arr[j]) < distanciaMin){
8                  distanciaMin = Math.abs(arr[i] - arr[j]);
9              }
10             j++;
11         }
12         i++;
13     }
14     return distanciaMin;
15 }
```

**Inciso b:**
Los tiempos de ejecuciones teóricos de los algoritmos son:

- Versión Original:

$$
\begin{aligned}
t_{\text{original}}(n) &= t_4 + t_5 \\
t_4 &= 1 \\
t_5 &= t_{\text{ini1}} + \sum_{i=0}^{n-1} (t_{\text{cond1}} + t_6 + t_{\text{incr1}}) + t_{\text{cond1}} \\
t_{\text{ini1}} &= 1 \\
t_{\text{cond1}} &= 2 \\
t_{\text{incr1}} &= 2 \\
t_6 &= t_{\text{ini2}} + \sum_{j=0}^{n-1} (t_{\text{cond2}} + t_7 + t_{\text{incr2}}) + t_{\text{cond2}} \\
t_{\text{ini2}} &= 1 \\
t_{\text{cond2}} &= 2 \\
t_{\text{incr2}} &= 2 \\
t_7 &= t_{\text{cond3}} + t_8 \\
t_{\text{cond3}} &= 6 \\
t_8 &= 5 \\
t_7 &= 6 + 5 = 11 \\
t_6 &= 1 + (n-1) \cdot (2 + 11 + 2) + 2 = 15n - 12 \\
t_5 &= 1 + (n-1) \cdot (2 + 15n - 12 + 2) + 2 = 15n^2 - 23n + 11 \\
t_{\text{original}}(n) &= 1 + 15n^2 - 23n + 11 = 15n^2 - 23n + 12
\end{aligned}
$$

- Versión Mejorada:

$$
\begin{aligned}
T_{\text{final}} &= t_1 + t_2 + t_3 \\
t_1 &= 1 \\
t_2 &= 1 \\
t_3 &= \sum_{i=0}^{n} (t_{\text{cond1}}) + t_{\text{int1}} + t_{\text{cond1}} \\
t_{\text{cond1}} &= 3 \\
t_{\text{int1}} &= t_4 + t_5 + t_{11} \\
t_4 &= 2 \\
t_5 &= \sum_{j=i+1}^{n} (t_{\text{cond2}} + t_{\text{int2}}) + t_{\text{cond2}} \\
t_{\text{cond2}} &= 3 \\
t_{\text{int2}} &= t_6 + t_9 \\
t_9 &= 2 \\
t_6 &= t_{\text{cond3}} + t_7 \\
t_{\text{cond3}} &= 5 \\
t_7 &= 5 \\
t_6 &= 5 + 5 = 10 \\
t_{\text{int2}} &= 10 + 2 = 12 \\
t_5 &= \sum_{i=0}^{n-1} (n - (i+1)) \cdot (3 + 12) + 3 = \frac{n \cdot (n-1)}{2} \cdot 15 + 3 = \frac{15n^2 - 15n}{2} + 3 \\
t_{11} &= 2 \\
t_{\text{int1}} &= 2 + \frac{15n^2 - 15n}{2} + 3 + 2 = \frac{15n^2 - 15n}{2} + 7 \\
t_3 &= 3n + \frac{15n^2 - 15n}{2} + 7 + 3 = \frac{15n^2 - 21n}{2} + 10 \\
T_{\text{final}} &= 1 + 1 + \frac{15n^2 - 21n}{2} + 10 = \frac{15n^2 - 15n}{2} + 12
\end{aligned}
$$

Los resultados empíricos obtenidos por el algoritmo mejorado con un n=1000000 y con 10000 ejecuciones fueron:
El tiempo medio de la ejecución de cada programa es: 2959.97ns
El tiempo medio de la ejecución de cada programa es: 831.79ns
El tiempo medio de la ejecución de cada programa es: 3260.14ns
El tiempo medio de la ejecución de cada programa es: 6564.1ns
El tiempo medio de la ejecución de cada programa es: 7883.02ns

armar tabla con los datos de las pruebas, diciendo qué magnitud tiene la muestra y cua´les fueron los resultados de la cantidad de ejecuciones para cada n probado

**Inciso c:**

Se define el conjunto de funciones de orden Θ (Theta) como:

$$\Theta(g(n)) = \{t : \mathbb{N} \to [0, \infty) \mid \exists c, d \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n \geq n_0, c \cdot g(n) \leq t(n) \leq d \cdot g(n)\}$$

revisar el N en mayuscula que debería estar en minúscula

Reescribimos con nuestra funcion:

Sea $T(n) = 15N^2 - 23n + 12$

$T(n) \in \Theta(N^2) = \{t : \mathbb{N} \to [0, \infty) \mid \exists c, d \in \mathbb{R}^+, \exists n_0 \in \mathbb{N} : \forall n \geq n_0, c \cdot N^2 \leq 15N^2 - 23n + 12 \leq d \cdot N^2\}$

En este caso es mejor hablar de Θ ya que estamos buscando la distancia mas corta entre elementos de un arreglo en un arreglo que es aleatoreo, por lo cual necesitamos saber el orden exacto. Si tuvieramos la informacion de que la distancia mas corta entre los elementos suele estar en los ultimos elemtentos entonces estariamos hablando de $\mathcal{O}$(N) que seria la cota superior osea el peor caso

revisar la justificación de por qué se utiliza theta en vez de O, o bien por qué conviene más. No está relacionado a la distancia más corta, sino a la información que nos brinda theta respecto de sostener el mismo comportamiento en tiempo del algoritmo.

2. Implementar un algoritmo tal que dado un número entero $n$ (con $n < 10^{18}$) compute la parte entera de la raíz cuadrada de $n$, usando solo operaciones primitivas ( $+, -, *, /, =, ==, <, >$) y cualquier estructura de control.

   - Cualquier solución lograda con funciones predefinidas tales como **import Math.sqrt()** o similares, **no** es válida.
   - La solución en $O(\sqrt{n})$ es lenta dado el tamaño de entrada. Tratar de resolver el problema en tiempo $log(n)$
   - Muestre, a partir de calcular el tiempo teórico $T(n)$ que su solución es de orden logarítmico.

Grupo 2: Sarmiento Fernandez Gramajo - Reibold - Petit - Keller - 3 - Implementar un algoritmo para encontrar elementos comunes entre dos listas ordenadas

**Problema:** Implementar un algoritmo que encuentre todos los elementos comunes entre dos listas ordenadas de números, como se muestra en el siguiente ejemplo:
**Ejemplo:**

- Para las listas:
$$l1 = [2, 5, 5, 5], \quad l2 = [2, 2, 3, 5, 5, 7]$$
- La salida esperada es:
$$elementosComunes = [2, 5, 5]$$

El código es el siguiente:

```
import java.util.ArrayList;
import java.util.List;

public class ElementosComunes {

    public static List<Integer> encontrarComunes(List<Integer> l1,
        List<Integer> l2) {
        List<Integer> comunes = new ArrayList<>();
        int i = 0, j = 0;
        int comparaciones = 0;

        while (i < l1.size() && j < l2.size()) {
            comparaciones++;
            if (l1.get(i).equals(l2.get(j))) {
                comunes.add(l1.get(i));
                i++;
                j++;
            } else if (l1.get(i) < l2.get(j)) {
                i++;
            } else {
                j++;
            }
        }

        System.out.println("Comparaciones realizadas: " +
            comparaciones);
        return comunes;
    }

    public static void main(String[] args) {
        List<Integer> l1 = List.of(2, 5, 5, 5);
        List<Integer> l2 = List.of(2, 2, 3, 5, 5, 7);

        List<Integer> comunes = encontrarComunes(l1, l2);
        System.out.println("Elementos comunes: " + comunes);
    }
}
```

**Pregunta 3. a - Máximo número de comparaciones**

El número máximo de comparaciones que realiza el algoritmo es $m + n$, donde $m$ y $n$ son las longitudes de las listas $l_1$ y $l_2$, respectivamente. Esto ocurre cuando el puntero de una lista alcanza el final antes que el de la otra.
En el peor de los casos, debemos comparar todos los elementos de ambas listas, lo que resulta en $m + n$ comparaciones.

**Pregunta 3. b - Análisis asintótico**

El análisis asintótico del algoritmo muestra que su complejidad es $O(m + n)$, donde $m$ y $n$ son las longitudes de las listas $l_1$ y $l_2$. Debido a que ambas listas están ordenadas, podemos recorrerlas una sola vez utilizando dos punteros. El algoritmo es óptimo, ya que cada elemento se examina solo una vez.

**Pregunta 3. C - Prueba empirica para contar el numero de comparaciones:**

Tenido en cuenta en el código del inciso anterior.

**Ejercicio 3.2 - Grupo 6: Diaz, Oliva, Gattas, Zuñiga, Mamani**

```
1  public static long parte_entera(long n) {
2      if (n < 2){// Casos triviales (n=0 o 1)
3          return n;
4      }
5
6      long x = n;//
7      long y = (x + 1) / 2;
8
9      while (y < x) {
10         x = y;
11         y = (x + n / x) / 2;
12     }
13
14     return x;
15 }
```

Muestre, a partir de calcular el tiempo teórico T(n) que su solución es de orden logarıtmico:

1) $T_n = T_{s1} + T_{s2} + T_{s3} + T_{s4} = 2 + 1 + 3 + (6 * log(n) + 1) =$ **6*log (n) + 7**

2) $T_{s1} = T_{cond} + 1 = 1 + 1 = 2$

3) $T_{s2} = 1_{asig} = 1$

4) $T_{s3} = 1_{asig} + 2_{op} = 1 + 2 = 3$

5) $T_{s4} = log(n) * (1 + T_b) + 1 = log(n) * (1 + 5) + 1 = 6 * log(n) + 1$
$T_b = 1_{asig} + (1_{asig} + 3_{op}) = 1 + 4 = 5$

El número de iteraciones k que se requieren para obtener una aproximación precisa depende logarítmicamente de la magnitud del número n. En otras palabras, el número de iteraciones es **O(log n)**.

reordenar las sentencias, revisar el número de referencia. Justificar de dònde sale el tiempo logarítmico. Mostrar que el resultado del algoritmo es el esperado. Indicar la fuente desde donde se obtuvo la solución.

Ejercicio 3.2 - Grupo 8: Belén, Bruno, Facundo, Jeremias, Kevin

```java
public static void main(String[] args) {
    long numero = 100000000;
    long bajo = 0;
    long alto = numero;
    long resultado = 0;
    boolean encontrado = false;

    // Realiza la busqueda binaria.
    while (bajo <= alto && !encontrado) {
        long medio = (bajo + alto) / 2;
        long cuadradoMedio = medio * medio;

        if (cuadradoMedio == numero) {
            resultado = medio; // Si medio^2 es igual al numero ingresado, entonces encontramos la raiz cuadrada exacta.
            encontrado = true;
        } else if (cuadradoMedio < numero) {
            bajo = medio + 1;
            resultado = medio; // Actualiza el resultado al mayor valor conocido cuya raiz cuadrada es < numero
        } else {
            alto = medio - 1;
        }
    }

    System.out.println("La parte entera de la raiz cuadrada de " + numero + " es: " + resultado);

```

```
26  }
```

Este código sirve para obtener la parte entera de una raíz cuadrada, utilizando una búsqueda binaria, en cada iteración se reduce a la mitad la búsqueda. Se sabe que el Orden de una búsqueda binaria es $O(\log n)$, por lo que este algoritmo, al utilizar el mismo mecanismo, también es de $O(\log n)$

3. Implementar un algoritmo para encontrar todos los elementos comunes entre dos listas ordenadas de números. Por ejemplo:

```
1  //
2  para las listas
3  l1 = [ 2, 5, 5, 5 ]
4  l2 = [ 2, 2, 3, 5, 5, 7 ]
5  //
6  la salida deberia ser
7  elementosComunes = [ 2, 5, 5 ]
```

a) ¿Cuál es el máximo número de comparaciones que tu algoritmo realiza si las longitudes de las listas son $m$ y $n$ respectivamente?

b) Analizar asintóticamente.

c) Hacer un programa que pruebe empíricamente que dadas dos listas diga cuántas comparaciones se realizan.

**Resolución - Grupo 3: Margni, Villarroel, Fernandez, Mendiberri, Fabris**

```java
public static List<Integer> encontrarComunes(int[] arr1, int[]
    arr2, int[] numComp) {
    List<Integer> listaComunes = new ArrayList<>();
    int puntero1 = 0, puntero2 = 0, largo1, largo2;
    largo1 = arr1.length;
    largo2 = arr2.length;

    // Mientras ninguno de los punteros haya alcanzado el
       final de los arreglos
    while (puntero1 < largo1 && puntero2 < largo2) {
        numComp[0]++;
        if (arr1[puntero1] == arr2[puntero2]) {
            // Si los elementos son iguales, aniadir a la
               lista de comunes
            listaComunes.add(arr1[puntero1]);
            puntero1++;
            puntero2++;
        } else if (arr1[puntero1] < arr2[puntero2]) {
```

```
                // Si el elemento de arr1 es menor, avanzar el
                   puntero de arr1
                puntero1++;
            } else {
                // Si el elemento de arr2 es menor, avanzar el
                   puntero de arr2
                puntero2++;
            }
        }

        return listaComunes;
    }
```

El máximo número de comparaciones es igual a $n + m - 1$

Cuando los elementos de los arreglos tienden a ser muy grandes, podemos decir que la función que representa a la cantidad de comparaciones es $O(n)$

Ejercicio 3.3 - Grupo 7

Listing 1: Algoritmo elementos repetidos

```
1  public static List elementosRepetidos(int[] arr1, int[] arr2, int
       [] cantComprob) {
2          int n = arr1.length, m = arr2.length, i = 0, j = 0;
3          List<Integer> lista = new ArrayList<Integer>();
4          do {
5              if (arr1[i] == arr2[j]) {
6                  lista.add(arr1[i]);
7                  cantComprob[0]++;
8                  i++;
9                  j++;
10             } else {
11                 if (arr1[i] < arr2[j]) {
12                     i++; // Si no son iguales muevo el puntero del
                           menor
13                     cantComprob[0]++;
14                 } else {
15                     j++;
16                     cantComprob[0]++;
17                 }
18             }
19
20         } while (!((i >= n) || (j >= m)));
21         return lista;
22     }
```

Codigo completo repositorio github

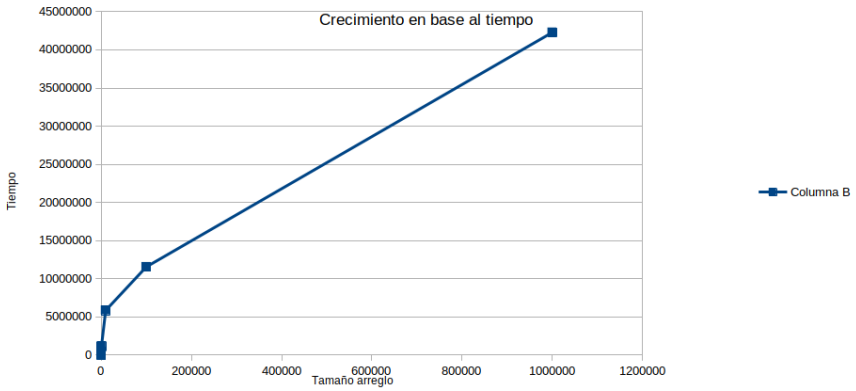
Ejercicio 3.3 - Grupo 7

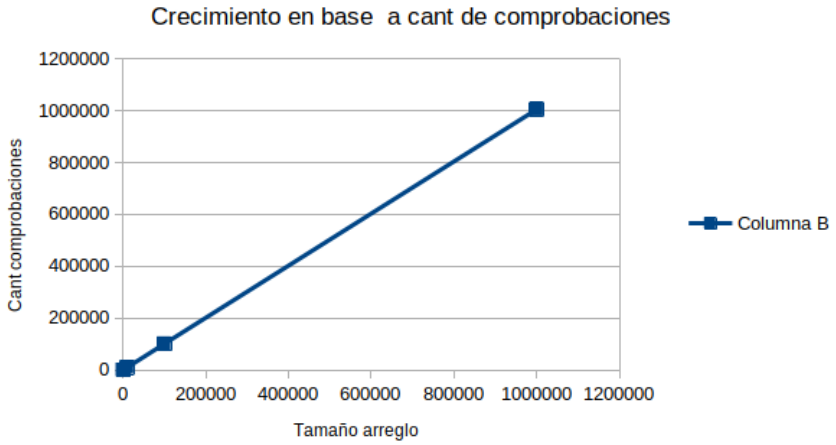
a) En el peor de los casos se recorren las dos listas en su totalidad por lo tanto el algoritmo pertence a O(n+m)

b) Analizando los graficos podemos ver que su crecimiento es lineal , coincidiendo con el analisis realizado en el a) por lo tanto no tiene asintota , su crecimiento es lineal

c) Esta funcion de contar las comprobaciones esta incluido en el algoritmo

PARA TODAS LAS RESOLUCIONES DEL EJERCICIO 3: - mostrar tablas para las pruebas empíricas que hicieron indicando la magnitud de la muestra y la cantidad de ejecuciones hecas con cada magnitud de muestra. También decir si hay aleatoreidad en la muestra o es siempre la misma (mismos datos)




ojo en la justificación de por qué es mejor Theta que Orden o en qué casos es mejor. Revisar si están justificando con el problema específico del dominio. tiene que ver con la información que tenemos de la función que nos permite determinar la asíntota

está bueno usar gráficas de comportamiento de función para diferentes magnitudes de muestra, esto es, si probaron con n = 100, n=1000, n=10000, etc, entonces prueben unos 10 o 15 valores de n (o más, lo que les de la máquina) y hagan una gráfica para dibujar el comportamiento de la variable