Algorítmica

Tema 1. La eficiencia de los algoritmos

Francisco Javier Cabrerizo Lorite

Curso académico 2023-2024

ETS de Ingenierías Informática y de Telecomunicación. Universidad de Granada

Índice

Objetivos

- Conocer el concepto de algoritmo.
- Plantearse la búsqueda de varias soluciones distintas para un mismo problema y evaluar la bondad de cada una de ellas.
- Ser consciente de la importancia del análisis de la eficiencia de un algoritmo como paso previo a su implementación.
- Conocer la notación asintótica para describir la eficiencia de un algoritmo (mejor, peor y caso promedio).
- Saber realizar el análisis de eficiencia de un algoritmo, tanto a nivel teórico como empírico.
- Conocer las técnicas básicas de resolución de ecuaciones de recurrencia.

Concepto de algoritmo

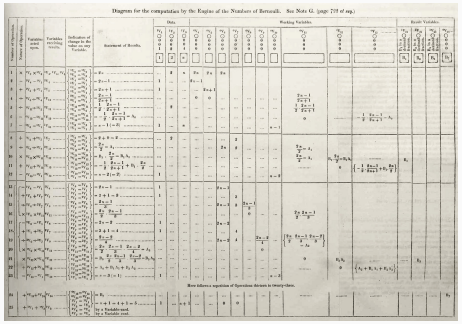
Concepto de algoritmo

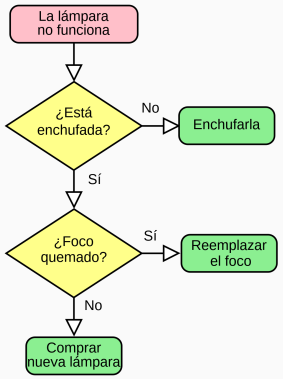
Definición

Conjunto ordenado y finito de pasos exentos de ambigüedad que, al llevarse a cabo con fidelidad, dará como resultado la tarea para la que se ha diseñado.

Su nombre proviene del matemático persa del siglo IX Muḥammad ibn Mūsā al-Khwārizmī.

Ejemplos





Concepto de algoritmo

Propiedades

- **Es una noción abstracta**. No depende del lenguaje de programación.
- **Está bien definido**. Pasos claros y sin ambigüedades.
- **Es coherente**. Los mismos datos iniciales siempre producen el mismo resultado.
- **Finito**. Debe terminar.
- **Efectivo**. Debe resolver el problema para el que se ha diseñado.

Concepto de algoritmo

Propiedades

- **Es una noción abstracta**. No depende del lenguaje de programación.
- **Está bien definido**. Pasos claros y sin ambigüedades.
- **Es coherente**. Los mismos datos iniciales siempre producen el mismo resultado.
- **Finito**. Debe terminar.
- **Efectivo**. Debe resolver el problema para el que se ha diseñado.

Todas las condiciones deben de cumplirse

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén.
2. Esperar a que esté caliente.
3. Echar carne.
4. Echar sal.
5. Mientras no esté lista, mover la carne.
6. Servir la carne.

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén.
2. Esperar a que esté caliente.
3. Echar carne.
4. Echar sal.
5. Mientras no esté lista, mover la carne.
6. Servir la carne.

¿Es un «buen» o «mal» algoritmo?

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén → ¿Cuánto aceite?
2. Esperar a que esté caliente → ¿Cuánto?
3. Echar carne → ¿Qué tipo de carne? ¿Cuánta?
4. Echar sal → ¿Cuánta sal?
5. Mientras no esté lista, mover la carne → ¿Cuándo está lista? ¿Cómo se mueve?
6. Servir la carne.

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén → ¿Cuánto aceite?
2. Esperar a que esté caliente → ¿Cuánto?
3. Echar carne → ¿Qué tipo de carne? ¿Cuánta?
4. Echar sal → ¿Cuánta sal?
5. Mientras no esté lista, mover la carne → ¿Cuándo está lista? ¿Cómo se mueve?
6. Servir la carne.

No es correcto

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén → ¿Cuánto aceite?
2. Esperar a que esté caliente → ¿Cuánto?
3. Echar carne → ¿Qué tipo de carne? ¿Cuánta?
4. Echar sal → ¿Cuánta sal?
5. Mientras no esté lista, mover la carne → ¿Cuándo está lista? ¿Cómo se mueve?
6. Servir la carne.

No es correcto

- No está bien definido.

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén → ¿Cuánto aceite?
2. Esperar a que esté caliente → ¿Cuánto?
3. Echar carne → ¿Qué tipo de carne? ¿Cuánta?
4. Echar sal → ¿Cuánta sal?
5. Mientras no esté lista, mover la carne → ¿Cuándo está lista? ¿Cómo se mueve?
6. Servir la carne.

No es correcto

- No está bien definido.
- No es coherente (no siempre igual).

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén → ¿Cuánto aceite?
2. Esperar a que esté caliente → ¿Cuánto?
3. Echar carne → ¿Qué tipo de carne? ¿Cuánta?
4. Echar sal → ¿Cuánta sal?
5. Mientras no esté lista, mover la carne → ¿Cuándo está lista? ¿Cómo se mueve?
6. Servir la carne.

No es correcto

- No está bien definido.
- No es coherente (no siempre igual).
- Tampoco es finito:

Concepto de algoritmo

Algoritmo para freír carne

1. Poner aceite en la sartén → ¿Cuánto aceite?
2. Esperar a que esté caliente → ¿Cuánto?
3. Echar carne → ¿Qué tipo de carne? ¿Cuánta?
4. Echar sal → ¿Cuánta sal?
5. Mientras no esté lista, mover la carne → ¿Cuándo está lista? ¿Cómo se mueve?
6. Servir la carne.

No es correcto

- No está bien definido.
- No es coherente (no siempre igual).
- Tampoco es finito:
  - Se nos ha olvidado encender el fuego, la carne nunca estará hecha.

Concepto de algoritmo

Ejemplo de «buen» algoritmo

1. Encender el horno a 210ºC y esperar a que esté a dicha temperatura.
2. Esparcir una cucharada de aceite de oliva homogéneamente en la bandeja del horno.
3. Colocar dos truchas en la bandeja del horno.
4. Esperar 35' o hasta que las truchas comiencen a tostarse por encima.
5. Apagar el horno.

Concepto de algoritmo

Ejemplo de «buen» algoritmo

1. Encender el horno a 210ºC y esperar a que esté a dicha temperatura.
2. Esparcir una cucharada de aceite de oliva homogéneamente en la bandeja del horno.
3. Colocar dos truchas en la bandeja del horno.
4. Esperar 35' o hasta que las truchas comiencen a tostarse por encima.
5. Apagar el horno.

Cumple la definición y sus propiedades

Concepto de algoritmo

Ejemplo de «buen» algoritmo

1. Encender el horno a 210ºC y esperar a que esté a dicha temperatura.
2. Esparcir una cucharada de aceite de oliva homogéneamente en la bandeja del horno.
3. Colocar dos truchas en la bandeja del horno.
4. Esperar 35' o hasta que las truchas comiencen a tostarse por encima.
5. Apagar el horno.

Cumple la definición y sus propiedades

- Conjunto ordenado y finito de pasos.

Concepto de algoritmo

Ejemplo de «buen» algoritmo

1. Encender el horno a 210ºC y esperar a que esté a dicha temperatura.
2. Esparcir una cucharada de aceite de oliva homogéneamente en la bandeja del horno.
3. Colocar dos truchas en la bandeja del horno.
4. Esperar 35' o hasta que las truchas comiencen a tostarse por encima.
5. Apagar el horno.

Cumple la definición y sus propiedades

- Conjunto ordenado y finito de pasos.
- Sin ambigüedad.

Concepto de algoritmo

Ejemplo de «buen» algoritmo

1. Encender el horno a 210ºC y esperar a que esté a dicha temperatura.
2. Esparcir una cucharada de aceite de oliva homogéneamente en la bandeja del horno.
3. Colocar dos truchas en la bandeja del horno.
4. Esperar 35' o hasta que las truchas comiencen a tostarse por encima.
5. Apagar el horno.

Cumple la definición y sus propiedades

- Conjunto ordenado y finito de pasos.
- Sin ambigüedad.
- Es efectivo.

Concepto de algoritmo

Ejemplo de «buen» algoritmo

1. Encender el horno a 210ºC y esperar a que esté a dicha temperatura.
2. Esparcir una cucharada de aceite de oliva homogéneamente en la bandeja del horno.
3. Colocar dos truchas en la bandeja del horno.
4. Esperar 35' o hasta que las truchas comiencen a tostarse por encima.
5. Apagar el horno.

Cumple la definición y sus propiedades

- Conjunto ordenado y finito de pasos.
- Sin ambigüedad.
- Es efectivo.
- Es abstracto (no depende del horno).

Multiplicación de enteros

Concepto de algoritmo

Multiplicación de enteros

- Algoritmo clásico.

Concepto de algoritmo

Multiplicación de enteros

- Algoritmo clásico.
- Algoritmo de multiplicación a la rusa.

Concepto de algoritmo

Multiplicación de enteros

- Algoritmo clásico.
- Algoritmo de multiplicación a la rusa.

Concepto de algoritmo

Multiplicación de enteros

- Algoritmo clásico.
- Algoritmo de multiplicación a la rusa.

Posibles criterios de elección de un algoritmo

1. Su adaptabilidad a los ordenadores.
2. Su simplicidad y elegancia.
3. El coste económico de su confección y puesta a punto.
4. El tiempo consumido para llevarlo a cabo.

Eficiencia de los algoritmos

Eficiencia de los algoritmos

¿Por qué estudiar eficiencia?

- Existe un método para implementar, ¿es viable?
- Existen métodos que resuelven el mismo problema:
  - ¿Cuál es el mejor?
  - ¿En qué situaciones?

Cómo medir la eficiencia

En tiempo Tiempo que tarda un algoritmo en resolver un problema.

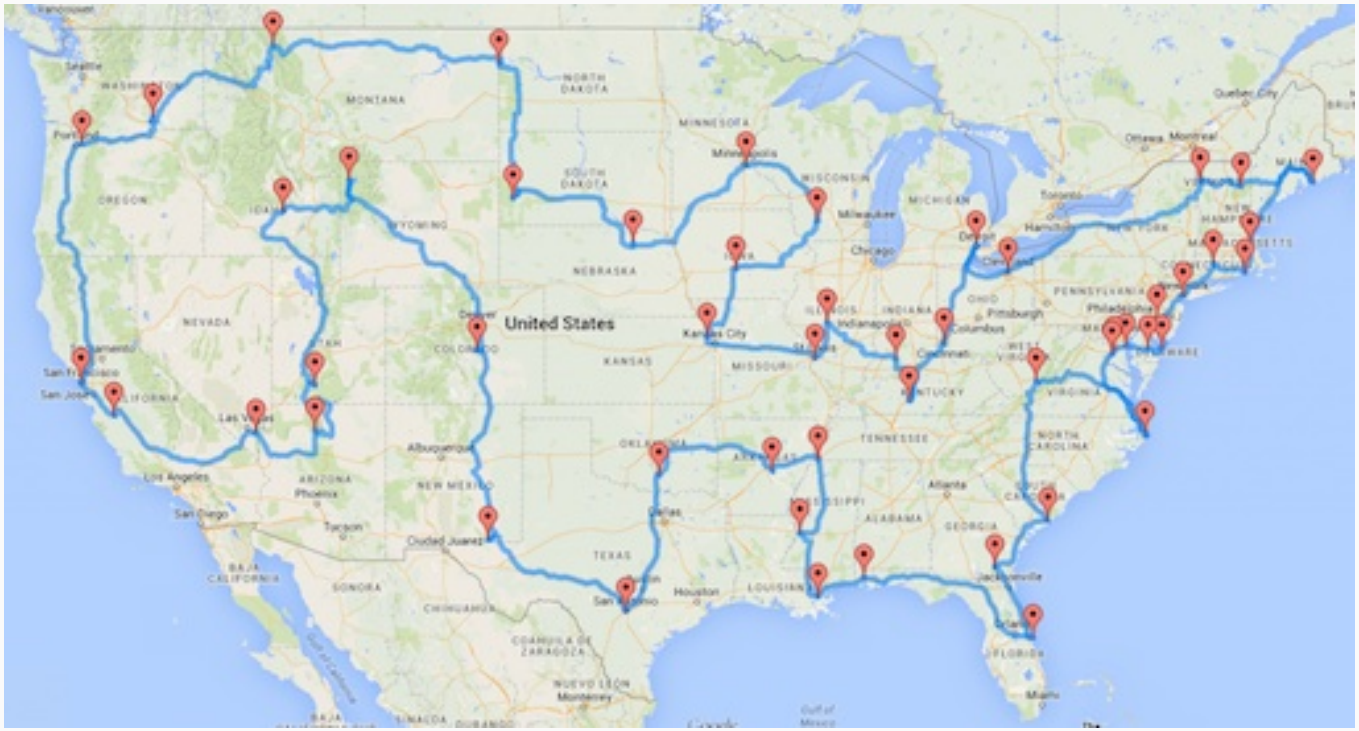
En espacio Recursos (memoria, espacio en disco, ...) necesarios para resolver el problema.

Mediremos la eficiencia basándonos en el tiempo

Eficiencia de los algoritmos

Problema del viajante de comercio

- Recorrer n ciudades una única vez y volver al punto de origen.



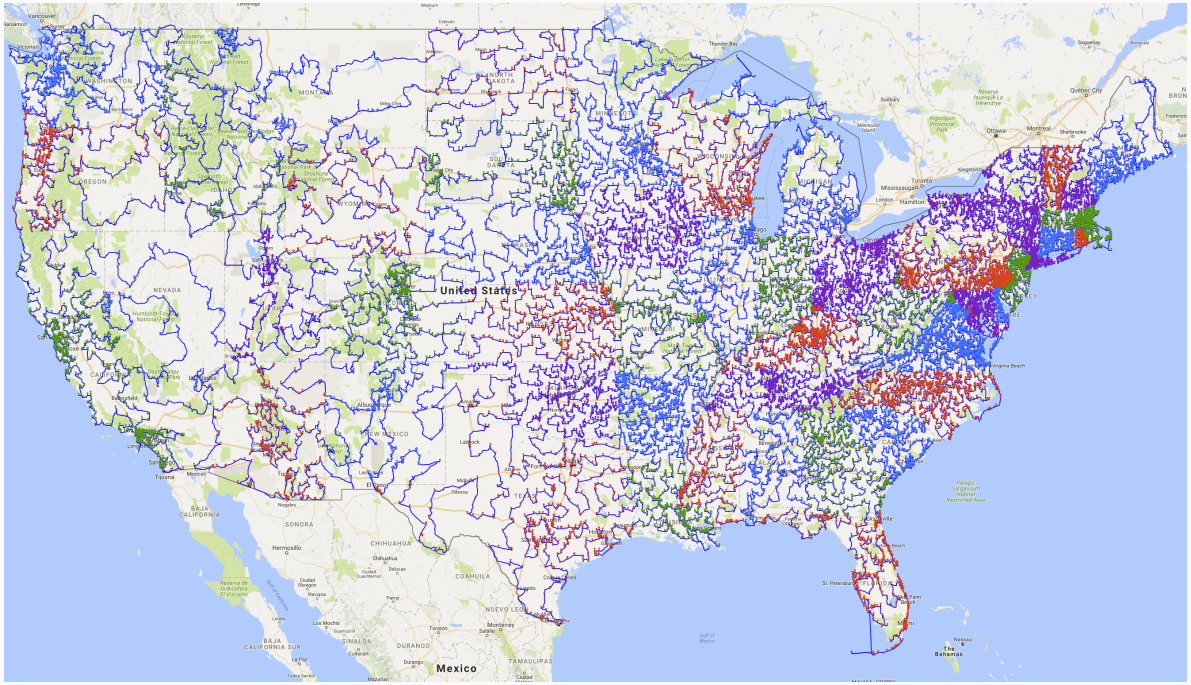
Eficiencia de los algoritmos

El viajante de comercio: aplicaciones

Rutas aéreas Entre aeropuertos.

Reparto almacén Optimizar entregas.

Cableado eléctrico Recorrido mínimo.



Eficiencia de los algoritmos

Ciudades (n)	Fuerza bruta	Algoritmo Held-Karp
10	2 segundos	0.1 segundos
11	22 segundos	0.2 segundos
12	4 minutos	0.4 segundos
14	13 horas	3 segundos
16	200 días	11 segundos
18	112 años	1 minuto
25	270 000 años	4 horas
50	$5 \cdot 10^{50}$ años	58 000 años

Fuente: `https://nbviewer.jupyter.org/url/norvig.com/ipython/TSP.ipynb`

Conclusiones

- El algoritmo importa y mucho.
- Es necesario estudiar la eficiencia de los algoritmos.

Eficiencia de los algoritmos

- **Problema**: problema general que queremos resolver.
  - Ordenación, búsqueda, etc.
- **Instancia del problema**: problema concreto.
  - Ordenar el vector [9, 6, 10, 24, 11, 14].
- **Caso**: instancia o conjunto de instancias con dificultad similar.
- **Tamaño del caso**: tamaño de la instancia a resolver.
  - En el caso anterior es igual a 6.

Eficiencia de los algoritmos

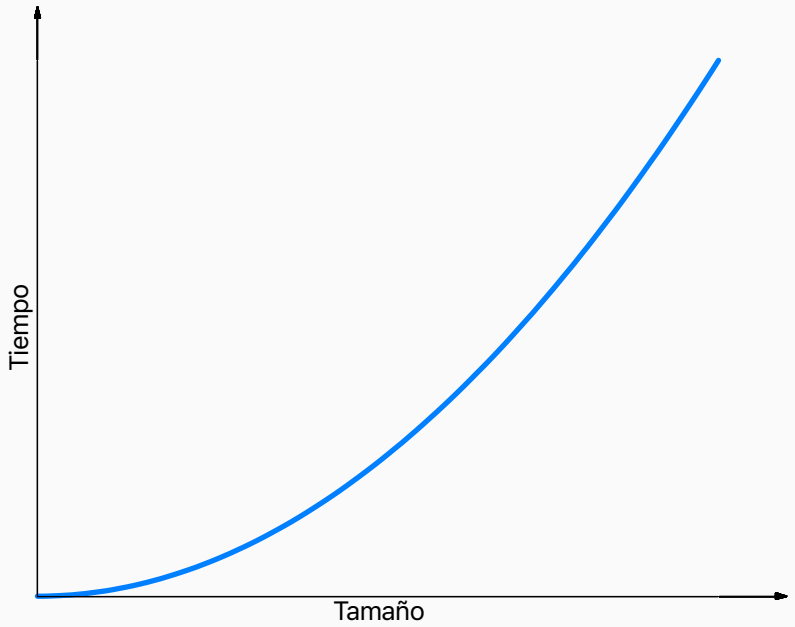
Problemas y casos

- Los problemas más interesantes incluyen una colección «infinita» de casos.
- Un algoritmo debe trabajar correctamente con cualquier caso del problema.
- Un algoritmo es incorrecto si encontramos un caso del problema que no produzca el resultado correcto.
- ¿Diferencia entre algoritmo y programa?

Eficiencia de los algoritmos

Se pretende relacionar

- El tiempo consumido por un algoritmo.
- Con el tamaño del caso que se quiere resolver.



Eficiencia de los algoritmos

Algoritmos de ordenación: inserción y selección

- Sean u y v dos vectores de n elementos ordenados ascendente y descendentemente, respectivamente.
- Comportamiento del algoritmo de selección:
  - Indiferente de u y v. El tiempo requerido para ordenar no varía en más de un 15%.
- Comportamiento del algoritmo de inserción:
  - Consume menos de 1/5 de segundo si u tiene 5000 elementos.
  - Consume más de tres minutos si v tiene 5000 elementos.

¿Tiempo consumido solo en función del tamaño del caso?

Eficiencia de los algoritmos

Diferentes tipos de casos

- El tiempo consumido por un algoritmo puede variar mucho entre dos casos diferentes del mismo tamaño.
- v describe el peor caso de este problema (caso sobre el que se tarda más tiempo). Es útil cuando necesitamos una garantía sobre el tiempo de ejecución del algoritmo.
- Si el problema se ha de resolver para muchos casos distintos, se usará el caso promedio: la media de los tiempos de todos los posibles casos del mismo tamaño.
- u describe el mejor caso de este problema (caso sobre el que se tarda menos tiempo).

Eficiencia de los algoritmos

- **Mejor caso**: aquel que hace que el algoritmo se ejecute el mínimo número de operaciones posible y, por tanto, que tarde el mínimo tiempo en ejecutarse comparado con otros casos de igual tamaño.
  - Depende de la entrada.
  - Es demasiado optimista.
- **Caso promedio**: situación media (esperanza matemática).
  - Difícil de calcular.
  - Hay que conocer la distribución de probabilidad.
- **Peor caso**: aquel que hace que el algoritmo se ejecute el máximo número de operaciones posible y, por tanto, que tarde el máximo tiempo en ejecutarse comparado con otros casos de igual tamaño.
  - Fácil de calcular.
  - Permite acotar el tiempo de ejecución.
  - Suele aproximarse al tiempo de ejecución real.

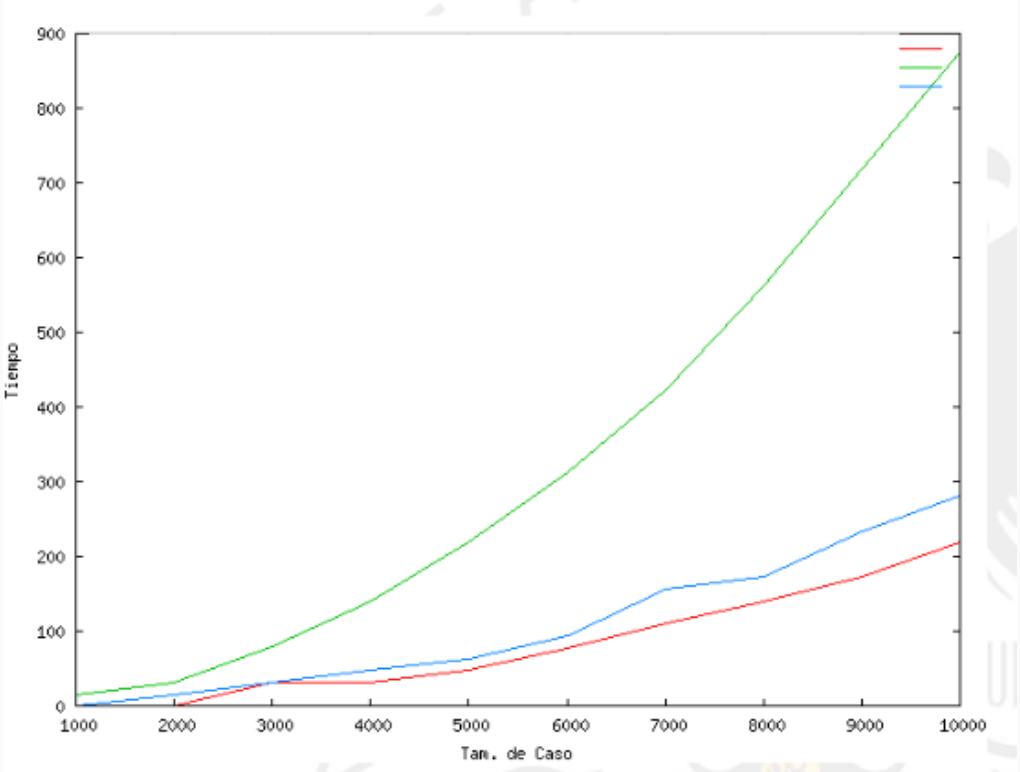
Eficiencia de los algoritmos

Algoritmos de ordenación

Rojo Inserción.

Verde Burbuja.

Azul Selección.



¿Cuál es más eficiente?

Eficiencia de los algoritmos

Factores que influyen en el tiempo de ejecución

- Datos de entrada.
- Calidad de código generado por el compilador para crear el programa objeto.
- Naturaleza y rapidez de las instrucciones máquina del procesador que ejecuta el programa.
- Complejidad en tiempo del algoritmo que subyace en el programa.

Eficiencia de los algoritmos

Métodos para calcular la eficiencia

- **Empírico**: ofrece una medida «real» (*a posteriori*), consistente en medir el tiempo de ejecución del algoritmo para unos valores de entrada dados y en un ordenador concreto.

- **Teórico**: ofrece una medida «teórica» (*a priori*), consistente en obtener una función que acote (por arriba o por abajo) el tiempo de ejecución del algoritmo para unos valores de entrada dados.

- **Híbrido**: combinación de los anteriores.

Eficiencia de los algoritmos

La eficiencia de un algoritmo es independiente del lenguaje de programación y del ordenador usado.

Principio de invarianza

Dado un algoritmo y dos implementaciones suyas I_1 e I_2, que tardan $T_1(n)$ y $T_2(n)$ unidades de tiempo, respectivamente, el principio de invarianza afirma que existe una constante real $c > 0$ y un número natural n_0 tales que para todo $n \geq n_0$ se verifica que $T_1(n) \leq c \cdot T_2(n)$.

El tiempo de ejecución de dos implementaciones distintas de un algoritmo no diferirá más que en una constante multiplicativa.

Eficiencia de los algoritmos

Otras consideraciones

- El tiempo de ejecución no depende directamente de la entrada, sino del tamaño de esta.
- $T(n)$ denota el tiempo de ejecución para unos datos de entrada de tamaño n.
- No hay unidad para expresar el tiempo de ejecución (se usa una constante que acumula los factores relativos a aspectos tecnológicos).
- Un algoritmo consume un tiempo de orden $T(n)$ si existe una constante $c > 0$ y una implementación del algoritmo capaz de resolver cualquier caso del problema en un tiempo acotado superiormente por $c \cdot T(n)$.
- El uso de segundos es arbitrario (basta con cambiar la constante «oculta» para expresar el tiempo en días o años).

Eficiencia de los algoritmos

Ejemplo

- El algoritmo A tiene un tiempo de ejecución de n^3 segundos.
- El algoritmo B tiene un tiempo de ejecución de $50 \cdot n^2$ segundos.

Eficiencia de los algoritmos

Ejemplo

- El algoritmo A tiene un tiempo de ejecución de n^3 segundos.
- El algoritmo B tiene un tiempo de ejecución de $50 \cdot n^2$ segundos.

n	n^3	$50 \cdot n^2$
2	8	200
4	64	800
49	117649	120050
100	1000000	500000
1000	1000000000	50000000

- Aunque el algoritmo A es mejor que el B para entradas de tamaño pequeño, el B es preferible para entradas de tamaño grande.

Eficiencia de los algoritmos

Otro ejemplo

- El algoritmo A tiene un tiempo de ejecución de n^2 días.
- El algoritmo B tiene un tiempo de ejecución de n^3 segundos.
- ¿Cuál es mejor asintóticamente?

Eficiencia de los algoritmos

Otro ejemplo

- El algoritmo A tiene un tiempo de ejecución de n^2 días.
- El algoritmo B tiene un tiempo de ejecución de n^3 segundos.
- ¿Cuál es mejor asintóticamente? El algoritmo A.

Eficiencia de los algoritmos

Otro ejemplo

- El algoritmo A tiene un tiempo de ejecución de n^2 días.
- El algoritmo B tiene un tiempo de ejecución de n^3 segundos.
- ¿Cuál es mejor asintóticamente? El algoritmo A.
- ¿En qué casos el algoritmo A es más rápido que el B?

Otro ejemplo

- El algoritmo A tiene un tiempo de ejecución de n^2 días.
- El algoritmo B tiene un tiempo de ejecución de n^3 segundos.
- ¿Cuál es mejor asintóticamente? El algoritmo A.
- ¿En qué casos el algoritmo A es más rápido que el B? Aquellos que requieren más de 20 millones de años para resolverlos ($n > 86400$).
- Desde un punto de vista práctico, el alto valor de la constante «oculta» recomienda usar el algoritmo B.

Notación asintótica

Notación asintótica

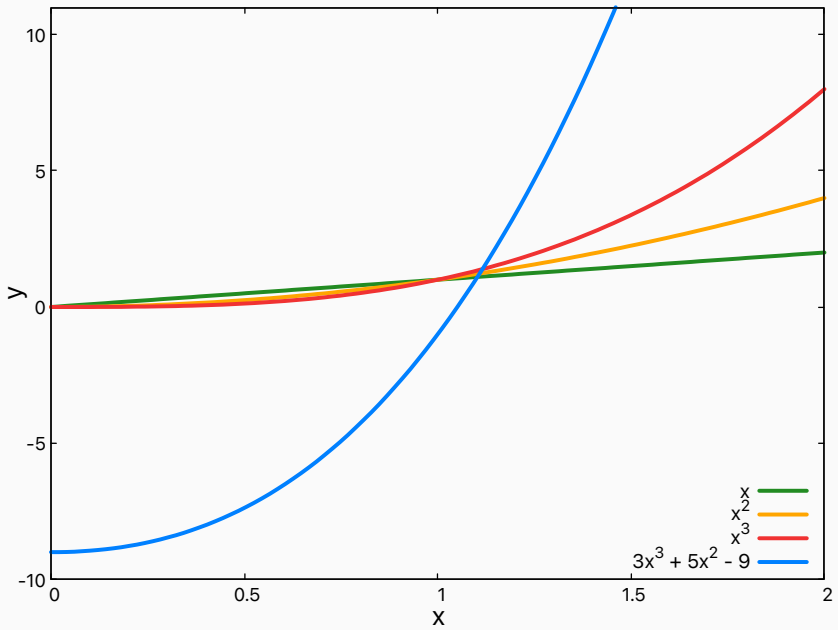
- Estudia el comportamiento del algoritmo para n grandes.
- No tiene en cuenta tamaños de n pequeños ni los factores constantes.
- Sirve para comparar funciones.
- Útil para calcular la eficiencia teórica de los algoritmos.

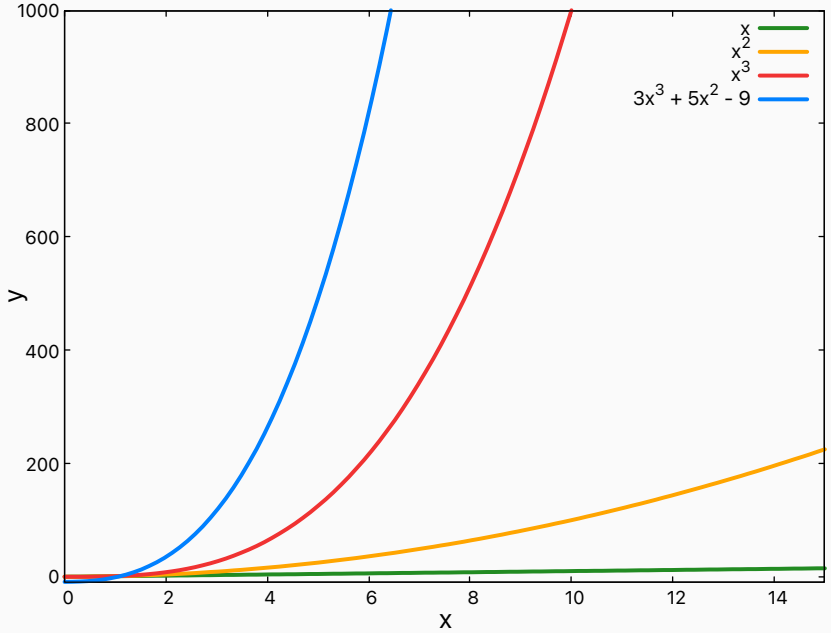
Notación asintótica

- Captura la conducta de las funciones para valores grandes de x.
- ¿Término dominante de $3x^3 + 5x^2 - 9$?

Notación asintótica

- Captura la conducta de las funciones para valores grandes de x.
- ¿Término dominante de $3x^3 + 5x^2 - 9$? El x^3.





Notación asintótica

La notación O

Sea $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$. Se define el conjunto de funciones de orden O de f como:

$$O(f(n)) = \{g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\} \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \text{tal que } g(n) \leq c \cdot f(n),\ \forall n > n_0\}$$

Una función $t : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ es de orden O de $f(n)$ si $t \in \mathrm{O}(f(n))$.

Intuitivamente, $t(n) \in \mathrm{O}(f(n))$ indica que $t(n)$ está acotada superiormente por algún múltiplo de $f(n)$.

Nos interesa en la menor función $f(n)$ tal que $t(n)$ pertenezca a $\mathrm{O}(f(n))$.

Conociendo la cota superior de un algoritmo podemos asegurar que, en ningún caso, el tiempo empleado será de un orden superior al de la cota.

Notación asintótica

- Si un algoritmo tiene un tiempo de ejecución $O(f(n))$, a $f(n)$ se le llama tasa de crecimiento.
- Suponemos que podemos evaluar algoritmos comparando sus tiempos de ejecución (despreciando constantes de proporcionalidad).

Ejemplo

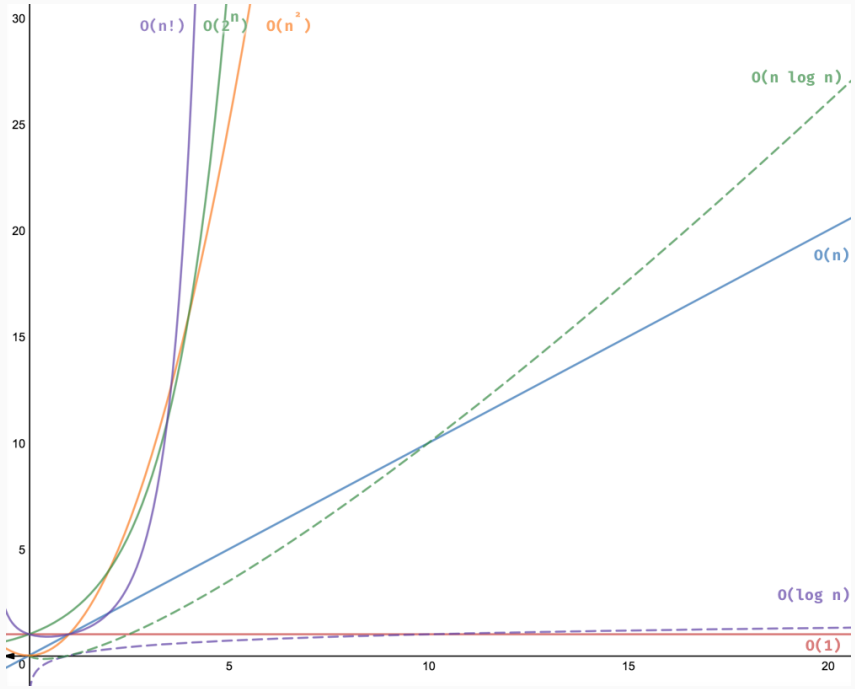
Un algoritmo con un tiempo de ejecución $O(n^2)$ es mejor que uno con un tiempo de ejecución $O(n^3)$.

- ¿$100n^2$ milisegundos o $5n^3$ milisegundos?
- ¿n^6 o $1000n^{5.9}$?

Notación asintótica

Órdenes

- Constante: $O(1)$.
- Logarítmico: $O(log(n))$.
- Lineal: $O(n)$.
- $O(n \cdot log(n))$.
- Polinomial: $O(n^k)$, $k > 1$.
- Exponencial: $O(a^n)$, $n > 1$.
- Factorial: $O(n!)$.
- ...



Notación asintótica

Equivalencia entre órdenes (regla del límite)

- $O(f(n)) \equiv O(g(n)) \Leftrightarrow lim_{n\to\infty} \frac{f(n)}{g(n)} \to c \in \mathbb{R}^+$.
- $O(f(n)) > O(g(n)) \Leftrightarrow lim_{n\to\infty} \frac{f(n)}{g(n)} \to \infty$.
- $O(f(n)) < O(g(n)) \Leftrightarrow lim_{n\to\infty} \frac{f(n)}{g(n)} \to 0$.

Por comodidad, ante equivalencias de órdenes de eficiencia, siempre nos referiremos al más simple.

$$O(1) \subset O(log(n)) \subset O(\sqrt{(n)}) \subset O(n) \subset O(n \cdot log(n)) \subset O(n^2) \subset$$
$$\subset O(n^2 \cdot log(n)) \subset O(n^3) \subset \ldots \subset O(n^k) \subset O(2^n) \subset O(n!)$$

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2 + n)$

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

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2 + n)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n^2}{(4n+1)^2+n} =$

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2 + n)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n^2}{(4n+1)^2+n} = lim_{n\to\infty} \frac{n^2}{(16n^2+1^2+8n)+n}$

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2 + n)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n^2}{(4n+1)^2+n} = lim_{n\to\infty} \frac{n^2}{(16n^2+1^2+8n)+n} \to \frac{1}{16}$

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2 + n)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n^2}{(4n+1)^2+n} = lim_{n\to\infty} \frac{n^2}{(16n^2+1^2+8n)+n} \to \frac{1}{16}$

Ambos algoritmos son equivalentes

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2+n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{n^2}{(4n+1)^2+n} = lim_{n\to\infty}\frac{n^2}{(16n^2+1^2+8n)+n} \to \frac{1}{16}$

Ambos algoritmos son equivalentes

Ejemplo 2

Algoritmo A: $O(2^n)$

Algoritmo B: $O(3^n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{2^n}{3^n} =$

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2+n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{n^2}{(4n+1)^2+n} = lim_{n\to\infty}\frac{n^2}{(16n^2+1^2+8n)+n} \to \frac{1}{16}$

Ambos algoritmos son equivalentes

Ejemplo 2

Algoritmo A: $O(2^n)$

Algoritmo B: $O(3^n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{2^n}{3^n} = lim_{n\to\infty}(\frac{2}{3})^n$

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2+n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{n^2}{(4n+1)^2+n} = lim_{n\to\infty}\frac{n^2}{(16n^2+1^2+8n)+n} \to \frac{1}{16}$

Ambos algoritmos son equivalentes

Ejemplo 2

Algoritmo A: $O(2^n)$

Algoritmo B: $O(3^n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{2^n}{3^n} = lim_{n\to\infty}(\frac{2}{3})^n \to 0$

Notación asintótica

Ejemplo 1

Algoritmo A: $O(n^2)$

Algoritmo B: $O((4n+1)^2+n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{n^2}{(4n+1)^2+n} = lim_{n\to\infty}\frac{n^2}{(16n^2+1^2+8n)+n} \to \frac{1}{16}$

Ambos algoritmos son equivalentes

Ejemplo 2

Algoritmo A: $O(2^n)$

Algoritmo B: $O(3^n)$

$lim_{n\to\infty}\frac{f(n)}{g(n)} = lim_{n\to\infty}\frac{2^n}{3^n} = lim_{n\to\infty}(\frac{2}{3})^n \to 0$

El algoritmo A es más eficiente que el B

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

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

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} =$

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)})$

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)}) \to 0$

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)}) \to 0$

El algoritmo A es más eficiente que el B

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)}) \to 0$

El algoritmo A es más eficiente que el B

Ejemplo 4

Algoritmo A: $O((n^2 + 29)^2)$

Algoritmo B: $O(n^3)$

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

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)}) \to 0$

El algoritmo A es más eficiente que el B

Ejemplo 4

Algoritmo A: $O((n^2 + 29)^2)$

Algoritmo B: $O(n^3)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{(n^2+29)^2}{n^3}$

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)}) \to 0$

El algoritmo A es más eficiente que el B

Ejemplo 4

Algoritmo A: $O((n^2 + 29)^2)$

Algoritmo B: $O(n^3)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{(n^2+29)^2}{n^3} = lim_{n\to\infty} \frac{n^4+58n^2+841}{n^3}$

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)}) \to 0$

El algoritmo A es más eficiente que el B

Ejemplo 4

Algoritmo A: $O((n^2 + 29)^2)$

Algoritmo B: $O(n^3)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{(n^2+29)^2}{n^3} = lim_{n\to\infty} \frac{n^4+58n^2+841}{n^3} \to \infty$

Notación asintótica

Ejemplo 3

Algoritmo A: $O(n)$

Algoritmo B: $O(n \cdot log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{n \cdot log(n)} = lim_{n\to\infty} (\frac{1}{log(n)}) \to 0$

El algoritmo A es más eficiente que el B

Ejemplo 4

Algoritmo A: $O((n^2 + 29)^2)$

Algoritmo B: $O(n^3)$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{(n^2+29)^2}{n^3} = lim_{n\to\infty} \frac{n^4+58n^2+841}{n^3} \to \infty$

El algoritmo B es más eficiente que el A

Notación asintótica

Ejemplo 5

Algoritmo A: $O(n)$

Algoritmo B: $O(log(n))$

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

Notación asintótica

Ejemplo 5

Algoritmo A: $O(n)$

Algoritmo B: $O(log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{log(n)}$

Notación asintótica

Ejemplo 5

Algoritmo A: $O(n)$

Algoritmo B: $O(log(n))$

$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{log(n)} \xrightarrow{\text{l'Hôpital}} lim_{n\to\infty} \left(\frac{1}{\frac{1}{n}} \right) =$

Notación asintótica

Ejemplo 5

Algoritmo A: $O(n)$

Algoritmo B: $O(log(n))$

$$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{log(n)} \xrightarrow{\text{l'Hôpital}} lim_{n\to\infty} \left(\frac{1}{\frac{1}{n}} \right) = lim_{n\to\infty} \frac{n}{1}$$

Notación asintótica

Ejemplo 5

Algoritmo A: $O(n)$

Algoritmo B: $O(log(n))$

$$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{log(n)} \xrightarrow{\text{l'Hôpital}} lim_{n\to\infty} \left(\frac{1}{\frac{1}{n}}\right) = lim_{n\to\infty} \frac{n}{1} \to \infty$$

Notación asintótica

Ejemplo 5

Algoritmo A: $O(n)$

Algoritmo B: $O(log(n))$

$$lim_{n\to\infty} \frac{f(n)}{g(n)} = lim_{n\to\infty} \frac{n}{log(n)} \xrightarrow{\text{l'Hôpital}} lim_{n\to\infty} \left(\frac{1}{\frac{1}{n}} \right) = lim_{n\to\infty} \frac{n}{1} \to \infty$$

El algoritmo B es más eficiente que A

Notación asintótica

La notación Ω

Sea $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$. Se define el conjunto de funciones de orden Ω de f como:

$$\begin{aligned}\Omega(f(n)) = &\{g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\} \mid \exists c \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \text{tal que} \\ &g(n) \geq c \cdot f(n),\ \forall n > n_0\}\end{aligned}$$

Una función $t : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ es de orden Ω de $f(n)$ si $t(n) \in \Omega(f(n))$.

Intuitivamente, $t(n) \in \Omega(f(n))$ indica que $t(n)$ está acotada inferiormente por algún múltiplo de $f(n)$.

Nos interesa la mayor función $f(n)$ tal que $t(n)$ pertenezca a $\Omega(f(n))$.

Conociendo la cota inferior de un algoritmo podemos asegurar que, en ningún caso, el tiempo empleado será de un orden inferior al de la cota.

Notación asintótica

Código de ejemplo

```
1  void insercion(int *v, int tam) {
2    int actual, clave;
3
4    for (int i = 1; i < tam; i++) {
5      clave = v[i];
6      actual = i-1;
7
8      while (actual >= 0 && v[actual] > clave) {
9        v[actual+1] = v[actual];
10       actual--;
11     }
12
13     v[actual+1] = clave;
14   }
15 }
```

¿Caso mejor? ¿Caso peor?

Notación asintótica

Código de ejemplo

```
1  void burbuja(int *v, int tam) {
2    bool cambiado = true;
3
4    while (cambiado) {
5        cambiado = false;
6
7        for (int i = 1; i < tam; i++) {
8          if (v[i-1] > v[i]) {
9            intercambiar(v[i-1], v[i]);
10           cambiado = true;
11         }
12       }
13   }
14 }
```

¿Caso mejor? ¿Caso peor?

Notación asintótica

Orden exacto Θ

Sea $f : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$. Se define el conjunto de funciones de orden Θ de f como:

$$\Theta(f(n)) = O(f(n)) \cap \Omega(f(n))$$

o, lo que es igual:

$$\begin{aligned}\Theta(f(n)) = &\{g : \mathbb{N} \to \mathbb{R}^+ \cup \{0\} \mid \exists c, d \in \mathbb{R}^+, \exists n_0 \in \mathbb{N}, \text{tal que} \\ &c \cdot f(n) \leq g(n) \leq d \cdot f(n),\ \forall n > n_0\}\end{aligned}$$

Una función $t : \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$ es de orden Θ de $f(n)$ si $t(n) \in \Theta(f(n))$.

Intuitivamente, $t(n) \in \Theta(f(n))$ indica que $t(n)$ está acotada tanto superior como inferiormente por múltiplos de $f(n)$, es decir, que $t(n)$ y $f(n)$ crecen de la misma forma.

Notación asintótica

Propiedades

- **Transitividad**: $f(n) \in O(g(n))$ y $g(n) \in O(h(n)) \Rightarrow f(n) \in O(h(n))$
  Igual para Ω y Θ.
- **Reflexiva**: $f(n) \in O(f(n))$
  Igual para Ω y Θ.
- **Simétrica**: $f(n) \in \Theta(g(n)) \Leftrightarrow g(n) \in \Theta(f(n))$
- **Suma**: Si $T1 \in O(f(n))$ y $T2 \in O(g(n))$,
  entonces $T1(n) + T2(n) \in O(\max\{f(n), g(n)\})$
- **Producto**: Si $T1 \in O(f(n))$ y $T2 \in O(g(n))$,
  entonces $T1(n) \cdot T2(n) \in O(f(n) \cdot g(n))$

Notación asintótica

Otras propiedades

- **Regla del máximo**: $\mathrm{O}(f(n) + g(n)) = \max\{\mathrm{O}(f(n)), \mathrm{O}(g(n))\}$
- **Regla de la suma**: $\mathrm{O}(f(n) + g(n)) = \mathrm{O}(f(n)) + \mathrm{O}(g(n))$
- **Regla del producto**: $\mathrm{O}(f(n) \cdot g(n)) = \mathrm{O}(f(n)) \cdot \mathrm{O}(g(n))$

Ejemplo 1: aplicación de la regla de la suma

En un programa, se ejecuta un código de eficiencia $O(n^2)$ y, a continuación, un código de eficiencia $O(n)$.

¿Cuál es la eficiencia del programa completo?

Ejemplo 1: aplicación de la regla de la suma

En un programa, se ejecuta un código de eficiencia $O(n^2)$ y, a continuación, un código de eficiencia $\mathrm{O}(n)$.

¿Cuál es la eficiencia del programa completo?

Regla de la suma $\mathrm{O}(f(n) + g(n)) = \mathrm{O}(f(n)) + \mathrm{O}(g(n))$

Ejemplo 1: aplicación de la regla de la suma

En un programa, se ejecuta un código de eficiencia $O(n^2)$ y, a continuación, un código de eficiencia $\mathrm{O}(n)$.

¿Cuál es la eficiencia del programa completo?

Regla de la suma $\mathrm{O}(f(n) + g(n)) = \mathrm{O}(f(n)) + \mathrm{O}(g(n))$

Por tanto:

$\mathrm{O}(n^2) + \mathrm{O}(n) =$

Ejemplo 1: aplicación de la regla de la suma

En un programa, se ejecuta un código de eficiencia $O(n^2)$ y, a continuación, un código de eficiencia $\mathrm{O}(n)$.

¿Cuál es la eficiencia del programa completo?

Regla de la suma $\mathrm{O}(f(n) + g(n)) = \mathrm{O}(f(n)) + \mathrm{O}(g(n))$

Por tanto:

$\mathrm{O}(n^2) + \mathrm{O}(n) = \mathrm{O}(n^2 + n) \Rightarrow$

Ejemplo 1: aplicación de la regla de la suma

En un programa, se ejecuta un código de eficiencia $O(n^2)$ y, a continuación, un código de eficiencia $\mathrm{O}(n)$.

¿Cuál es la eficiencia del programa completo?

Regla de la suma $\mathrm{O}(f(n) + g(n)) = \mathrm{O}(f(n)) + \mathrm{O}(g(n))$

Por tanto:

$\mathrm{O}(n^2) + \mathrm{O}(n) = \mathrm{O}(n^2 + n) \Rightarrow$ El programa tiene eficiencia $\mathrm{O}(n^2 + n)$.

Ejemplo 2: aplicación de la regla del máximo

Un programa se ejecuta en un tiempo igual a $T(n) = n^2 + n$.

¿Cuál es su eficiencia?

Ejemplo 2: aplicación de la regla del máximo

Un programa se ejecuta en un tiempo igual a $T(n) = n^2 + n$.

¿Cuál es su eficiencia?

Principio de invarianza $T(n) \leq c \cdot (n^2 + n) \rightarrow$ nos vale cualquier $c \geq 1$

Entonces, el orden es $O(n^2 + n)$.

Ejemplo 2: aplicación de la regla del máximo

Un programa se ejecuta en un tiempo igual a $T(n) = n^2 + n$.

¿Cuál es su eficiencia?

Principio de invarianza $T(n) \leq c \cdot (n^2 + n) \rightarrow$ nos vale cualquier $c \geq 1$

Entonces, el orden es $O(n^2 + n)$.

Regla del máximo $O(f(n) + g(n)) = \max\{O(f(n)), O(g(n))\}$

Ejemplo 2: aplicación de la regla del máximo

Un programa se ejecuta en un tiempo igual a $T(n) = n^2 + n$.

¿Cuál es su eficiencia?

Principio de invarianza $T(n) \leq c \cdot (n^2 + n) \rightarrow$ nos vale cualquier $c \geq 1$

Entonces, el orden es $O(n^2 + n)$.

Regla del máximo $O(f(n) + g(n)) = \max\{O(f(n)), O(g(n))\}$

Por tanto:

$O(n^2 + n) =$

Ejemplo 2: aplicación de la regla del máximo

Un programa se ejecuta en un tiempo igual a $T(n) = n^2 + n$.

¿Cuál es su eficiencia?

Principio de invarianza $T(n) \leq c \cdot (n^2 + n) \rightarrow$ nos vale cualquier $c \geq 1$

Entonces, el orden es $O(n^2 + n)$.

Regla del máximo $O(f(n) + g(n)) = \max\{O(f(n)), O(g(n))\}$

Por tanto:

$O(n^2 + n) = \max\{O(n^2), O(n)\} \Rightarrow$

Ejemplo 2: aplicación de la regla del máximo

Un programa se ejecuta en un tiempo igual a $T(n) = n^2 + n$.

¿Cuál es su eficiencia?

Principio de invarianza $T(n) \leq c \cdot (n^2 + n) \rightarrow$ nos vale cualquier $c \geq 1$

Entonces, el orden es $O(n^2 + n)$.

Regla del máximo $O(f(n) + g(n)) = \max\{O(f(n)), O(g(n))\}$

Por tanto:

$O(n^2 + n) = \max\{O(n^2), O(n)\} \Rightarrow$ El programa tiene eficiencia $O(n^2)$.

Ejemplo 3: aplicación de la regla del producto

Un programa consiste en un bucle cuya eficiencia es $O(n)$. En cada iteración dentro del bucle, se ejecuta un código cuya eficiencia es $O(n^2)$.

¿Cuál es la eficiencia del programa?

Ejemplo 3: aplicación de la regla del producto

Un programa consiste en un bucle cuya eficiencia es $O(n)$. En cada iteración dentro del bucle, se ejecuta un código cuya eficiencia es $O(n^2)$.

¿Cuál es la eficiencia del programa?

Si la ejecución del bucle es $O(n)$ y en cada iteración se ejecuta el cuerpo, que es $O(n^2)$, en total se realizan $O(n) \cdot O(n^2)$ operaciones.

Ejemplo 3: aplicación de la regla del producto

Un programa consiste en un bucle cuya eficiencia es $O(n)$. En cada iteración dentro del bucle, se ejecuta un código cuya eficiencia es $O(n^2)$.

¿Cuál es la eficiencia del programa?

Si la ejecución del bucle es $O(n)$ y en cada iteración se ejecuta el cuerpo, que es $O(n^2)$, en total se realizan $O(n) \cdot O(n^2)$ operaciones.

Regla del producto $O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$

Ejemplo 3: aplicación de la regla del producto

Un programa consiste en un bucle cuya eficiencia es $O(n)$. En cada iteración dentro del bucle, se ejecuta un código cuya eficiencia es $O(n^2)$.

¿Cuál es la eficiencia del programa?

Si la ejecución del bucle es $O(n)$ y en cada iteración se ejecuta el cuerpo, que es $O(n^2)$, en total se realizan $O(n) \cdot O(n^2)$ operaciones.

Regla del producto $O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$

Por tanto:

$O(n) \cdot O(n^2) =$

Ejemplo 3: aplicación de la regla del producto

Un programa consiste en un bucle cuya eficiencia es $O(n)$. En cada iteración dentro del bucle, se ejecuta un código cuya eficiencia es $O(n^2)$.

¿Cuál es la eficiencia del programa?

Si la ejecución del bucle es $O(n)$ y en cada iteración se ejecuta el cuerpo, que es $O(n^2)$, en total se realizan $O(n) \cdot O(n^2)$ operaciones.

Regla del producto $O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$

Por tanto:

$O(n) \cdot O(n^2) = O(n \cdot n^2) =$

Ejemplo 3: aplicación de la regla del producto

Un programa consiste en un bucle cuya eficiencia es $O(n)$. En cada iteración dentro del bucle, se ejecuta un código cuya eficiencia es $O(n^2)$.

¿Cuál es la eficiencia del programa?

Si la ejecución del bucle es $O(n)$ y en cada iteración se ejecuta el cuerpo, que es $O(n^2)$, en total se realizan $O(n) \cdot O(n^2)$ operaciones.

Regla del producto $O(f(n) \cdot g(n)) = O(f(n)) \cdot O(g(n))$

Por tanto:

$O(n) \cdot O(n^2) = O(n \cdot n^2) = O(n^3) \Rightarrow$ El programa es $O(n^3)$.

Notación asintótica

Órdenes con varios parámetros

En ocasiones, nos encontramos que el tamaño del problema no depende de una única variable n, si no de varias. En estos casos, se analiza igual, considerando que f tiene varias variables:

Sea $f : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+ \cup \{0\}$. Se define el conjunto de funciones de orden O de f como:

$$O(f(n,m)) = \{g : \mathbb{N} \times \mathbb{N} \to \mathbb{R}^+ \cup \{0\} \mid \exists c \in \mathbb{R}^+, \exists n_0, m_0 \in \mathbb{N}, \text{tal que } g(n,m) \leq c \cdot f(n,m),\ \forall n > n_0, m > m_0\}$$

Se dice que el algoritmo es de orden $O(f(n,m))$ si para cualquier valor «muy grande» de n y m:

$$T(n,m) \leq c \cdot f(n,m)$$

Notación asintótica

Ejemplo: suma de matrices de n filas y m columnas

$$A_{n,m} = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,m} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n,1} & a_{n,2} & \cdots & a_{n,m} \end{pmatrix} \qquad B_{n,m} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,m} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n,1} & b_{n,2} & \cdots & b_{n,m} \end{pmatrix}$$

Suma $A_{n,m} + B_{n,m} = C_{n,m}; c_{ij} = a_{ij} + b_{ij}$

Orden de eficiencia: $O(n \cdot m)$.

Análisis de algoritmos

Calcular de forma teórica la función $f(n)$ del orden O de un algoritmo

1. Analizar el problema y detectar de qué variables/parámetros depende su tamaño.
   - Ordenación de un vector: el tiempo depende del tamaño del vector.
   - Suma de matrices: el tiempo depende del tamaño de las matrices (número de filas y columnas).
2. Aplicar las reglas de análisis de eficiencia de las operaciones en el algoritmo.

Análisis de algoritmos

El análisis de la eficiencia se basa en el tipo de sentencias

- Operaciones elementales.
- Sentencias condicionales.
- Sentencias repetitivas.
- Secuencias de sentencias.
- Llamadas a funciones no recursivas.
- Llamadas a funciones recursivas.

Sentencias simples (operaciones elementales)

- Aquellas cuya ejecución no depende del tamaño de caso.
- Operaciones sobre tipos básicos:
  - Operaciones aritmétricas (suma, multiplicación, etc.).
  - Asignaciones y acceso a estructuras básicas indexadas.
  - Entrada/salida de datos desde teclado/fichero o hacia fichero/pantalla.
  - Comparaciones, operaciones booleanas, etc.
- Tiempo de ejecución acotado por una constante (suponemos 1).

Ejemplos

```
1 x = x+8;
2 cin >> x;
3 fich.open("mifichero.txt");
```

Sentencias simples (operaciones elementales)

- Aquellas cuya ejecución no depende del tamaño de caso.
- Operaciones sobre tipos básicos:
  - Operaciones aritmétricas (suma, multiplicación, etc.).
  - Asignaciones y acceso a estructuras básicas indexadas.
  - Entrada/salida de datos desde teclado/fichero o hacia fichero/pantalla.
  - Comparaciones, operaciones booleanas, etc.
- Tiempo de ejecución acotado por una constante (suponemos 1).

Ejemplos

```
1 x = x+8;
2 cin >> x;
3 fich.open("mifichero.txt");
```

Complejidades ocultas

```
1 void calcular(Vector<int>& a,
2                Vector<int>& b) {
3   Vector<int> c = a*b;
```

Sentencias simples (operaciones elementales)

- Aquellas cuya ejecución no depende del tamaño de caso.
- Operaciones sobre tipos básicos:
  - Operaciones aritmétricas (suma, multiplicación, etc.).
  - Asignaciones y acceso a estructuras básicas indexadas.
  - Entrada/salida de datos desde teclado/fichero o hacia fichero/pantalla.
  - Comparaciones, operaciones booleanas, etc.
- Tiempo de ejecución acotado por una constante (suponemos 1).

Ejemplos

```
1 x = x+8;
2 cin >> x;
3 fich.open("mifichero.txt");
```

Complejidades ocultas

```
1 void calcular(Vector<int>& a,
2                Vector<int>& b) {
3   Vector<int> c = a*b;
```

Coste de las operaciones elementales $O(1)$.

Sentencias condicionales

1. Evaluar `condición`.
2. Ejecutar `bloque1` solo si es cierta.
3. Ejecutar `bloque2` solo si no es cierta.

Esquema

```
si (condición) hacer:
    bloque1
si no hacer:
    bloque2
```

Peor caso

$O(\text{condición}) + \max\{O(\text{bloque1}), O(\text{bloque2})\}$

Mejor caso

$\Omega(\text{condicion}) + \min\{\Omega(\text{bloque1}), \Omega(\text{bloque2})\}$

Ejemplo

```
1  cout << "Dime un numero: ";
2  cin >> n;
3
4  if (n % 2 == 1) {
5    cout << "Es impar";
6  }
7  else {
8    for (int i = 1; i <= n; ++i) {
9      cout << i;
10   }
11 }
```

Eficiencia del condicional

- (n % 2 == 1)
- Bloque 1 (cout)
- Bloque 2 (for)

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   if (n % 2 == 1) {
5     cout << "Es impar";
6   }
7   else {
8     for (int i = 1; i <= n; ++i) {
9       cout << i;
10    }
11  }
```

Eficiencia del condicional

- (n % 2 == 1)
  $\Rightarrow O(1), \Omega(1)$
- Bloque 1 (cout)
- Bloque 2 (for)

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   if (n % 2 == 1) {
5     cout << "Es impar";
6   }
7   else {
8     for (int i = 1; i <= n; ++i) {
9       cout << i;
10    }
11  }
```

Eficiencia del condicional

- (n % 2 == 1)
  $\Rightarrow O(1), \Omega(1)$
- Bloque 1 (cout)
  $\Rightarrow O(1), \Omega(1)$
- Bloque 2 (for)

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   if (n % 2 == 1) {
5     cout << "Es impar";
6   }
7   else {
8     for (int i = 1; i <= n; ++i) {
9       cout << i;
10    }
11  }
```

Eficiencia del condicional

- (n % 2 == 1)
  $\Rightarrow O(1), \Omega(1)$
- Bloque 1 (cout)
  $\Rightarrow O(1), \Omega(1)$
- Bloque 2 (for)
  $\Rightarrow O(n), \Omega(n)$

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   if (n % 2 == 1) {
5     cout << "Es impar";
6   }
7   else {
8     for (int i = 1; i <= n; ++i) {
9       cout << i;
10    }
11  }
```

Eficiencia del condicional

- (n % 2 == 1)
  $\Rightarrow O(1), \Omega(1)$
- Bloque 1 (cout)
  $\Rightarrow O(1), \Omega(1)$
- Bloque 2 (for)
  $\Rightarrow O(n), \Omega(n)$

Tiempo de ejecución

Peor caso: $O(1) + \max\{O(1), O(n)\} = O(1) + O(n) = O(n)$

Mejor caso: $\Omega(1) + \min\{\Omega(1), \Omega(n)\} = \Omega(1) + \Omega(1) = \Omega(1)$

Sentencias repetitivas «mientras»

- Constan de la evaluación de una condición, y la ejecución de un bloque de sentencias, mientras la condición se cumpla.

```
mientras (condición) hacer:
    bloque de sentencias
```

Sentencias repetitivas «mientras»

- Constan de la evaluación de una condición, y la ejecución de un bloque de sentencias, mientras la condición se cumpla.

```
mientras (condición) hacer:
    bloque de sentencias
```

Cálculo de eficiencia de sentencias repetitivas «mientras»

f(n) Eficiencia del `bloque de sentencias`.

g(n) Eficiencia de comprobar la `condición`.

h(n) Número de veces que se ejecuta el bucle.

Coste $O(g(n) + h(n) \cdot (g(n) + f(n))$

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      cout << n;
9      n--;
10   }
11
12   cout << endl;
13   return 0;
14 }
```

Tiempo de ejecución while

- (n $>$ 0)
- Bloque de sentencias
- Repeticiones

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      cout << n;
9      n--;
10   }
11
12   cout << endl;
13   return 0;
14 }
```

Tiempo de ejecución while

- $(\text{n} > 0) \Rightarrow g(n) = 1$
- Bloque de sentencias
- Repeticiones

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      cout << n;
9      n--;
10   }
11
12   cout << endl;
13   return 0;
14 }
```

Tiempo de ejecución while

- $(\text{n} > 0) \Rightarrow g(n) = 1$
- Bloque de sentencias
  $\Rightarrow f(n) = 2$
- Repeticiones

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      cout << n;
9      n--;
10   }
11
12   cout << endl;
13   return 0;
14 }
```

Tiempo de ejecución while

- $(\text{n} > 0) \Rightarrow g(n) = 1$
- Bloque de sentencias $\Rightarrow f(n) = 2$
- Repeticiones $\Rightarrow h(n) = n$

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      cout << n;
9      n--;
10   }
11
12   cout << endl;
13   return 0;
14 }
```

Tiempo de ejecución while

- $(\text{n} > 0) \Rightarrow g(n) = 1$
- Bloque de sentencias $\Rightarrow f(n) = 2$
- Repeticiones $\Rightarrow h(n) = n$

Tiempo de ejecución

$g(n) + h(n) \cdot (g(n) + f(n)) = 1 + n \cdot (1 + 2) \Rightarrow \text{O}(3 \cdot n + 1)$

$\text{O}(3 \cdot n + 1) = \max\{\text{O}(3 \cdot n), \text{O}(1)\} = \text{O}(3 \cdot n) \Rightarrow \text{O}(n)$

Cálculo de eficiencia de sentencias repetitivas «para»

- Constan de la evaluación de una condición, la ejecución de un bloque de sentencias, y una actualización, mientras la condición se cumpla.

```
para (inicialización; condición; actualización) hacer:
    bloque de sentencias
```

Cálculo de eficiencia de sentencias repetitivas «para»

- Constan de la evaluación de una condición, la ejecución de un bloque de sentencias, y una actualización, mientras la condición se cumpla.

```
para (inicialización; condición; actualización) hacer:
    bloque de sentencias
```

Cálculo de eficiencia de sentencias repetitivas «para»

f(n) Eficiencia del `bloque de sentencias`.

g(n) Eficiencia de comprobar la `condición`.

h(n) Número de veces que se ejecuta el bucle.

a(n) Eficiencia de la `actualización`.

i(n) Eficiencia de la `inicialización`.

Coste $O(i(n) + g(n) + h(n) \cdot (g(n) + f(n) + a(n))$

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      for (int i = 1; i <= n; i *= 2)
       {
9        cout << i;
10     }
11
12     n--;
13   }
14
15   cout << endl;
16   return 0;
17 }
```

Eficiencia for

- Inicialización
- Condición
- Actualización
- Bloque sentencias

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      for (int i = 1; i <= n; i *= 2)
       {
9        cout << i;
10     }
11
12     n--;
13   }
14
15   cout << endl;
16   return 0;
17 }
```

Eficiencia for

- Inicialización $\Rightarrow O(1)$
- Condición
- Actualización
- Bloque sentencias

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      for (int i = 1; i <= n; i *= 2)
       {
9        cout << i;
10     }
11
12     n--;
13   }
14
15   cout << endl;
16   return 0;
17 }
```

Eficiencia for

- Inicialización $\Rightarrow O(1)$
- Condición $\Rightarrow O(1)$
- Actualización
- Bloque sentencias

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      for (int i = 1; i <= n; i *= 2)
       {
9        cout << i;
10     }
11
12     n--;
13   }
14
15   cout << endl;
16   return 0;
17 }
```

Eficiencia for

- Inicialización $\Rightarrow O(1)$
- Condición $\Rightarrow O(1)$
- Actualización $\Rightarrow O(1)$
- Bloque sentencias

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      for (int i = 1; i <= n; i *= 2)
       {
9        cout << i;
10     }
11
12     n--;
13   }
14
15   cout << endl;
16   return 0;
17 }
```

Eficiencia for

- Inicialización $\Rightarrow O(1)$
- Condición $\Rightarrow O(1)$
- Actualización $\Rightarrow O(1)$
- Bloque sentencias $\Rightarrow O(1)$

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      for (int i = 1; i <= n; i *= 2)
       {
9        cout << i;
10     }
11
12     n--;
13   }
14
15   cout << endl;
16   return 0;
17 }
```

Iteraciones

- Iteración 1, i = 1.
- Iteración 2, i = 2.
- Iteración 3, i = 4.
- ...
- Iteraciones

Ejemplo

```
1  int main() {
2    int n;
3
4    cout << "Dime un numero: ";
5    cin >> n;
6
7    while (n > 0) {
8      for (int i = 1; i <= n; i *= 2)
       {
9        cout << i;
10     }
11
12     n--;
13   }
14
15   cout << endl;
16   return 0;
17 }
```

Iteraciones

- Iteración 1, i = 1.
- Iteración 2, i = 2.
- Iteración 3, i = 4.
- ...
- Iteraciones $\Rightarrow log_2(n)$.

Ejemplo

```
1
2   while (n > 0) {
3     for (int i = 1; i <= n; i *= 2) {
4       cout << i;
5     }
6
7     n--;
8   }
```

Tiempo de ejecución

- for: $O(1) + O(1) + O(log_2(n)) \cdot (O(1) + O(1) + O(1))) = O(log_2(n))$
- while: $O(1 + n \cdot (1 + log_2(n) + 1) = O(n \cdot log_2(n))$

Secuencia de sentencias

- Constan de la ejecución de un bloque de sentencias.

```
sentencia 1
sentencia 2
...
sentencia S
```

Cálculo de eficiencia de secuencia de sentencias

- Asumiendo que cada sentencia i tiene eficiencia $O(f_i(n))$, la eficiencia se obtiene aplicando las reglas de la suma y el máximo:

$$O(f_1(n) + f_2(n) + \ldots + f_S(n)) = \max\{O(f_1(n)), O(f_2(n)), \ldots, O(f_S(n))\}$$

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   while (n > 0) {
5     for (int i = 1; i <= n; i *= 2)
      {
6       cout << i;
7     }
8
9     n--;
10  }
```

Eficiencia

- Sentencia 1 (`cout`)
- Sentencia 2 (`cin`)
- Sentencia 3 (`while`)

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   while (n > 0) {
5     for (int i = 1; i <= n; i *= 2)
      {
6       cout << i;
7     }
8
9     n--;
10  }
```

Eficiencia

- Sentencia 1 (`cout`)
  $\Rightarrow O(1)$
- Sentencia 2 (`cin`)
- Sentencia 3 (`while`)

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   while (n > 0) {
5     for (int i = 1; i <= n; i *= 2)
      {
6       cout << i;
7     }
8
9     n--;
10  }
```

Eficiencia

- Sentencia 1 (`cout`) $\Rightarrow O(1)$
- Sentencia 2 (`cin`) $\Rightarrow O(1)$
- Sentencia 3 (`while`)

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   while (n > 0) {
5     for (int i = 1; i <= n; i *= 2)
      {
6       cout << i;
7     }
8
9     n--;
10  }
```

Eficiencia

- Sentencia 1 (`cout`) $\Rightarrow O(1)$
- Sentencia 2 (`cin`) $\Rightarrow O(1)$
- Sentencia 3 (`while`) $\Rightarrow O(n \cdot log_2(n))$

Ejemplo

```
1   cout << "Dime un numero: ";
2   cin >> n;
3
4   while (n > 0) {
5     for (int i = 1; i <= n; i *= 2)
      {
6       cout << i;
7     }
8
9     n--;
10  }
```

Eficiencia

- Sentencia 1 (`cout`) $\Rightarrow O(1)$
- Sentencia 2 (`cin`) $\Rightarrow O(1)$
- Sentencia 3 (`while`) $\Rightarrow O(n \cdot log_2(n))$

Tiempo de ejecución

$\max\{O(1), O(1), O(n \cdot log_2(n))\} = O(n \cdot log_2(n))$

Funciones

- La eficiencia de una función es el máximo de las eficiencias de las sentencias que la componen.
- La eficiencia de una llamada a función dependerá de si sus parámetros de entrada dependen o no del tamaño del problema.

Ejemplo

```
1  bool es_primo(int valor) {
2    double tope = sqrt(valor);
3
4    for (int i = 2; i <= tope; i++) {
5      if (valor % i == 0) {
6        return false;
7      }
8    }
9
10   return true;
11 }
```

¿Cuál es la eficiencia de la función?

Ejemplo

```
1  for (int i = 1; i < n; i++) {
2    if (es_primo(i)) {
3      cout << i;
4    }
5  }
```

¿Cuál es la eficiencia?

Ejemplo

```
1  for (int i = 1; i < n; i++) {
2    if (es_primo(i)) {
3      cout << i;
4    }
5  }
```

¿Cuál es la eficiencia?

Otro ejemplo

```
1  for (int i = 1; i < n; i++) {
2    if (es_primo(7654321)) {
3      cout << i;
4    }
5  }
```

¿Cuál es la eficiencia?

Otro ejemplo

```
1  void funcion(int *v, int n) {
2    for (int i = 0; i < n; i++) {
3      v[i] = (i*3+15-5*i) / n;
4      v[i] = otra_fun(v, n-1)*otra_fun(v, n-2);
5    }
6  }
7
8  int otra_func(int *v, int n){
9    for(int i = n-1; i > 0; i = i/2)
10     v[i] = v[i]-1;
11   return v[0];
12 }
```

¿Cuál es la eficiencia de `funcion`?

O(n*log(n)).

Otro ejemplo

```
1  bool es_palindromo(char *v) {
2    bool palindromo = true;
3    int inicio = 0;
4    int fin = strlen(v) - 1;
5
6    while (palindromo && (inicio < fin)) {
7      if (v[inicio] != v[fin]) {
8        palindromo = false;
9      }
10     inicio++;
11     fin--;
12   }
13
14   return palindromo;
```

Calcular la Longitud de la Cadena: La llamada a strlen(v) tiene una complejidad de O(n).
Bucle While: El bucle while tiene una complejidad de O(n).

¿Cuál es la eficiencia de la función?

Por lo tanto, la complejidad total de la función es_palindromo es:
O(n)+O(n)=O(n)

Funciones recursivas

- Calcular el tiempo de ejecución $T(n)$ de la función con tamaño n.
- Expresarlo como una ecuación de recurrencias.
- Resolver la ecuación de recurrencias para calcular el orden de eficiencia.

Eficiencia de una función recursiva

Caso base No depende de la recursividad.

Caso general Sí depende de la recursividad.

Mejor caso Caso general más favorable.

Peor caso Caso general más desfavorable.

Ejemplo

```
1 unsigned long factorial(int n) {
2   if (n <= 1) {
3     return 1;
4   }
5   else {
6     return n*factorial(n-1);
7   }
8 }
```

¿Eficiencia?

Ejemplo

```
1 unsigned long factorial(int n) {
2   if (n <= 1) {
3     return 1;
4   }
5   else {
6     return n*factorial(n-1);
7   }
8 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Ejemplo

```
1 unsigned long factorial(int n) {
2   if (n <= 1) {
3     return 1;
4   }
5   else {
6     return n*factorial(n-1);
7   }
8 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Caso general $\Rightarrow O(1) + T(n-1)$

Ejemplo

```
1 unsigned long factorial(int n) {
2   if (n <= 1) {
3     return 1;
4   }
5   else {
6     return n*factorial(n-1);
7   }
8 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Caso general $\Rightarrow O(1) + T(n-1)$

$$T(n) = \begin{cases} d & n \leq 1 \\ c + T(n-1) & n > 1 \end{cases}$$

Ejemplo

```
1 int fibonacci(int n) {
2   if (n <= 1) {
3     return n;
4   }
5   else {
6     return fibonacci(n-1) + fibonacci(n-2);
7   }
8 }
```

¿Eficiencia?

Ejemplo

```
1 int fibonacci(int n) {
2   if (n <= 1) {
3     return n;
4   }
5   else {
6     return fibonacci(n-1) + fibonacci(n-2);
7   }
8 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Ejemplo

```
1 int fibonacci(int n) {
2   if (n <= 1) {
3     return n;
4   }
5   else {
6     return fibonacci(n-1) + fibonacci(n-2);
7   }
8 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Caso general $\Rightarrow O(1) + T(n-1) + T(n-2)$

Ejemplo

```
1 int fibonacci(int n) {
2   if (n <= 1) {
3     return n;
4   }
5   else {
6     return fibonacci(n-1) + fibonacci(n-2);
7   }
8 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Caso general $\Rightarrow O(1) + T(n-1) + T(n-2)$

$$T(n) = \begin{cases} d & n \leq 1 \\ c + T(n-1) + T(n-2) & n > 1 \end{cases}$$

Ejemplo

```
1  int bus_bin(double *v, int ini, int fin, double buscado) {
2    int centro = (ini + fin) / 2;
3    if (ini > fin)
4      return -1;
5    else if (v[centro] == buscado) {
6      return centro;
7    } else if (buscado < v[centro]){
8      return bus_bin(v, ini, centro-1, buscado);
9    } else{
10     return bus_bin(v, centro+1, fin, buscado);
11   }
12 }
```

¿Eficiencia?

Ejemplo

```
1  int bus_bin(double *v, int ini, int fin, double buscado) {
2    int centro = (ini + fin) / 2;
3    if (ini > fin)
4      return -1;
5    else if (v[centro] == buscado) {
6      return centro;
7    } else if (buscado < v[centro]){
8      return bus_bin(v, ini, centro-1, buscado);
9    } else{
10     return bus_bin(v, centro+1, fin, buscado);
11   }
12 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Ejemplo

```
1  int bus_bin(double *v, int ini, int fin, double buscado) {
2    int centro = (ini + fin) / 2;
3    if (ini > fin)
4      return -1;
5    else if (v[centro] == buscado) {
6      return centro;
7    } else if (buscado < v[centro]){
8      return bus_bin(v, ini, centro-1, buscado);
9    } else{
10     return bus_bin(v, centro+1, fin, buscado);
11   }
12 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Caso general $\Rightarrow O(1) + T(n/2)$

Ejemplo

```
1  int bus_bin(double *v, int ini, int fin, double buscado) {
2    int centro = (ini + fin) / 2;
3    if (ini > fin)
4      return -1;
5    else if (v[centro] == buscado) {
6      return centro;
7    } else if (buscado < v[centro]){
8      return bus_bin(v, ini, centro-1, buscado);
9    } else{
10     return bus_bin(v, centro+1, fin, buscado);
11   }
12 }
```

¿Eficiencia?

Caso base $\Rightarrow O(1)$

Caso general $\Rightarrow O(1) + T(n/2)$

$$T(n) = \begin{cases} d & \text{caso base} \\ c + T(n/2) & \text{caso general} \end{cases}$$

Resolución de recurrencias

Resolución de recurrencias

- Una vez planteada la ecuación de recurrencias, hay que resolverla para obtener su orden de eficiencia.

- Métodos:
  - Desarrollo en series de la fórmula (expansión).
  - Método de la ecuación característica.

Resolución de recurrencias

Expansión

- Desarrollar progresivamente la ecuación para diversos niveles.
- Sustituir cada aparición del valor de la función por la expresión expedificada en la recurrencia.
- Identificar un patrón general,
- Aplicar ese patrón para resolver, llevando el argumento de la función a algún caso básico.

Resolución de recurrencias

Ejemplo

$$T(n) = c + T(n-1)$$

Ejemplo

$$T(n) = c + T(n-1)$$

Desarrollo

$$\begin{aligned} T(n) &= c + T(n-1) \\ &= c + (c + T(n-2)) = 2 \cdot c + T(n-2) \\ &= 2 \cdot c + (c + T(n-3)) = 3 \cdot c - T(n-3) \\ &\dots \\ &= i \cdot c + T(n-i) \\ &\dots \\ &= (n-1) \cdot c + T(n-(n-1)) = (n-1) \cdot c + d \end{aligned}$$

Por tanto, $T(n)$ es $O(n)$

Resolución de recurrencias

Ejemplo

$$T(n) = c + T(n/2)$$

Ejemplo

$$T(n) = c + T(n/2)$$

Desarrollo

$$\begin{aligned} T(n) &= c + T(n/2) \\ &= c + (c + T(n/4)) = 2 \cdot c + T(n/4) \\ &= 2 \cdot c + (c + T(n/8)) = 3 \cdot c + T(n/8) \\ &\dots \\ &= i \cdot c + T(n/2^i) = c \cdot log_2(i) + T(n/i) \\ &\dots \\ &= c \cdot log_2(n) + T(n/n) = c \cdot log_2(n) + d \end{aligned}$$

Por tanto, $T(n)$ es $O(log_2(n))$

Método de la ecuación característica

- Proporciona una metodología muy organizada para obtener la eficiencia de manera simple.
- Se estudiarán los siguiente casos:
  - Ecuaciones lineales homogéneas.
  - Ecuaciones lineales no homogéneas.
- También se tendrán en cuenta otros aspectos:
  - Cambios de variable.
  - Transformaciones del rango.

Resolución de recurrencias

Ecuaciones lineales homogéneas

- Son de la forma:

$$a_0 \cdot T(n) + a_1 \cdot T(n-1) + a_2 \cdot T(n-2) + \ldots + a_k \cdot T(n-k) = 0$$

- Los coeficientes a_i son números reales y k es un número natural entre 1 y n.

Resolución

- Se hace el cambio $x^n = T(n)$:

$$a_0 \cdot x^n + a_1 \cdot x^{n-1} + a_2 \cdot x^{n-2} + \ldots + a_k \cdot x^{n-k} = 0$$

- Se saca factor común x^{n-k}:

$$(a_0 \cdot x^k + a_1 \cdot x^{k-1} + a_2 \cdot x^{k-2} + \ldots + a_k) \cdot x^{n-k} = 0$$

- Si satisface si $x = 0$ (solución trivial que no interesa).

Resolución (continuación)

- Ecuación característica (resultante de los pasos anteriores):

$$a_0 \cdot x^k + a_1 \cdot x^{k-1} + a_2 \cdot x^{k-2} + \ldots + a_k$$

- La expresión anterior se denomina polinomio característico:

$$p(x) = a_0 \cdot x^k + a_1 \cdot x^{k-1} + a_2 \cdot x^{k-2} + \ldots + a_k$$

- Por el Teorema Fundamental del Álgebra, se sabe que:

$$p(x) = \prod_{i=1}^{k} (x - r_i)$$

  Debemos calcular las raíces r_i del polinomio.

Resolución (continuación)

- El tiempo de ejecución es:

$$T(n) = \sum_{i=1}^{k} \sum_{j=0}^{m_i - 1} c_{ij} \cdot r_i^n \cdot n^j$$

- c_{ij} son coeficientes reales.
- r_i son las raíces del polinomio.
- k es el número de raíces distintas.
- m_i es el número de veces que r_i es raíz de $p(x)$.

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado: $T(n) - T(n-1) - T(n-2) = 0$
2. Cambio notación:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado: $T(n) - T(n-1) - T(n-2) = 0$
2. Cambio notación: $x^n - x^{n-1} - x^{n-2} = 0$
3. Polinomio característico:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado: $T(n) - T(n-1) - T(n-2) = 0$
2. Cambio notación: $x^n - x^{n-1} - x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - x - 1$
4. Quedaría:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado: $T(n) - T(n-1) - T(n-2) = 0$
2. Cambio notación: $x^n - x^{n-1} - x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - x - 1$
4. Quedaría: $p(x) = (x - r_1) \cdot (x - r_2)$
5. Raíces:

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado: $T(n) - T(n-1) - T(n-2) = 0$
2. Cambio notación: $x^n - x^{n-1} - x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - x - 1$
4. Quedaría: $p(x) = (x - r_1) \cdot (x - r_2)$
5. Raíces: $r_1 = \frac{1+\sqrt{5}}{2}, r_2 = \frac{1-\sqrt{5}}{2}$
6. Ambas raíces aparecen una única vez:

Factorizamos x^(n-2) en todos los términos de la ecuación

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado: $T(n) - T(n-1) - T(n-2) = 0$
2. Cambio notación: $x^n - x^{n-1} - x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - x - 1$
4. Quedaría: $p(x) = (x - r_1) \cdot (x - r_2)$
5. Raíces: $r_1 = \frac{1+\sqrt{5}}{2}, r_2 = \frac{1-\sqrt{5}}{2}$
6. Ambas raíces aparecen una única vez: $T(n) = c_{10} \cdot r_1^n + c_{20} \cdot r_2^n$

Ejemplo

$$T(n) = T(n-1) + T(n-2)$$

1. En el mismo lado: $T(n) - T(n-1) - T(n-2) = 0$
2. Cambio notación: $x^n - x^{n-1} - x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - x - 1$
4. Quedaría: $p(x) = (x - r_1) \cdot (x - r_2)$
5. Raíces: $r_1 = \frac{1+\sqrt{5}}{2}, r_2 = \frac{1-\sqrt{5}}{2}$
6. Ambas raíces aparecen una única vez: $T(n) = c_{10} \cdot r_1^n + c_{20} \cdot r_2^n$
7. $T(n) = c_{10} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n + c_{20} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n$

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado: $T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$
2. Cambio notación:

Resolución de recurrencias

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado: $T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$
2. Cambio notación: $x^n - 5 \cdot x^{n-1} + 8 \cdot x^{n-2} - 4 \cdot x^{n-3} = 0$
3. Polinomio característico:

Resolución de recurrencias

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado: $T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$
2. Cambio notación: $x^n - 5 \cdot x^{n-1} + 8 \cdot x^{n-2} - 4 \cdot x^{n-3} = 0$
3. Polinomio característico: $p(x) = x^3 - 5 \cdot x^2 + 8 \cdot x - 4$
4. Resolviendo:

Resolución de recurrencias

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado: $T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$
2. Cambio notación: $x^n - 5 \cdot x^{n-1} + 8 \cdot x^{n-2} - 4 \cdot x^{n-3} = 0$
3. Polinomio característico: $p(x) = x^3 - 5 \cdot x^2 + 8 \cdot x - 4$
4. Resolviendo: $p(x) = (x-2)^2 \cdot (x-1)$
5. Raíces:

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado: $T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$
2. Cambio notación: $x^n - 5 \cdot x^{n-1} + 8 \cdot x^{n-2} - 4 \cdot x^{n-3} = 0$
3. Polinomio característico: $p(x) = x^3 - 5 \cdot x^2 + 8 \cdot x - 4$
4. Resolviendo: $p(x) = (x-2)^2 \cdot (x-1)$
5. Raíces: $r_1 = 2, m_1 = 2, r_2 = 1, m_2 = 1$
6. El primero aparece dos veces y el otro una vez:

Resolución de recurrencias

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado: $T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$
2. Cambio notación: $x^n - 5 \cdot x^{n-1} + 8 \cdot x^{n-2} - 4 \cdot x^{n-3} = 0$
3. Polinomio característico: $p(x) = x^3 - 5 \cdot x^2 + 8 \cdot x - 4$
4. Resolviendo: $p(x) = (x-2)^2 \cdot (x-1)$
5. Raíces: $r_1 = 2, m_1 = 2, r_2 = 1, m_2 = 1$
6. El primero aparece dos veces y el otro una vez:
   $T(n) = c_{10} \cdot r_1^n \cdot n^0 + c_{11} \cdot r_1^n \cdot n + c_{20} r_2^n \cdot n^0$

Ejemplo

$$T(n) = 5T(n-1) - 8T(n-2) + 4T(n-3)$$

1. En el mismo lado: $T(n) - 5T(n-1) + 8T(n-2) - 4T(n-3) = 0$
2. Cambio notación: $x^n - 5 \cdot x^{n-1} + 8 \cdot x^{n-2} - 4 \cdot x^{n-3} = 0$
3. Polinomio característico: $p(x) = x^3 - 5 \cdot x^2 + 8 \cdot x - 4$
4. Resolviendo: $p(x) = (x-2)^2 \cdot (x-1)$
5. Raíces: $r_1 = 2, m_1 = 2, r_2 = 1, m_2 = 1$
6. El primero aparece dos veces y el otro una vez:
   $T(n) = c_{10} \cdot r_1^n \cdot n^0 + c_{11} \cdot r_1^n \cdot n + c_{20} r_2^n \cdot n^0$
7. $T(n) = c_{10} \cdot 2^n + c_{11} \cdot n \cdot 2^n + c_{20} \cdot 1^n$

Resolución de recurrencias

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

Resolución de recurrencias

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado: $T(n) - 2T(n-1) + T(n-2) = 0$
2. Cambio notación:

Resolución de recurrencias

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado: $T(n) - 2T(n-1) + T(n-2) = 0$
2. Cambio notación: $x^n - 2 \cdot x^{n-1} + x^{n-2} = 0$
3. Polinomio característico:

Resolución de recurrencias

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado: $T(n) - 2T(n-1) + T(n-2) = 0$
2. Cambio notación: $x^n - 2 \cdot x^{n-1} + x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - 2 \cdot x + 1$
4. Resolviendo:

Resolución de recurrencias

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado: $T(n) - 2T(n-1) + T(n-2) = 0$
2. Cambio notación: $x^n - 2 \cdot x^{n-1} + x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - 2 \cdot x + 1$
4. Resolviendo: $p(x) = (x-1)^2$
5. Raíces:

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado: $T(n) - 2T(n-1) + T(n-2) = 0$
2. Cambio notación: $x^n - 2 \cdot x^{n-1} + x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - 2 \cdot x + 1$
4. Resolviendo: $p(x) = (x-1)^2$
5. Raíces: $r_1 = 1, m_1 = 2$
6. Una única raíz que aparece dos veces:

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado: $T(n) - 2T(n-1) + T(n-2) = 0$
2. Cambio notación: $x^n - 2 \cdot x^{n-1} + x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - 2 \cdot x + 1$
4. Resolviendo: $p(x) = (x-1)^2$
5. Raíces: $r_1 = 1, m_1 = 2$
6. Una única raíz que aparece dos veces: $T(n) = c_{10} \cdot r_1^n \cdot n^0 + c_{11} \cdot r_1^n \cdot n$

Ejemplo

$$T(n) = 2T(n-1) - T(n-2)$$

1. En el mismo lado: $T(n) - 2T(n-1) + T(n-2) = 0$
2. Cambio notación: $x^n - 2 \cdot x^{n-1} + x^{n-2} = 0$
3. Polinomio característico: $p(x) = x^2 - 2 \cdot x + 1$
4. Resolviendo: $p(x) = (x-1)^2$
5. Raíces: $r_1 = 1, m_1 = 2$
6. Una única raíz que aparece dos veces: $T(n) = c_{10} \cdot r_1^n \cdot n^0 + c_{11} \cdot r_1^n \cdot n$
7. $T(n) = c_{10} \cdot 1^n + c_{11} \cdot n \cdot 1^n$

Ecuaciones Lineales no homogéneas

- En la ecuación aparecen términos no recursivos:

$$\textbf{Ejemplo}:\ T(n) = T(n-1) + 1$$

- Los términos no recursivos se pueden expresar como $\sum b_i^n \cdot q_i(n)$ donde b_i son números reales y $q_i(n)$ son polinomios en n de grado d_i.
- Forma general:

$$a_0 \cdot T(n) + a_1 \cdot T(n-1) + \ldots + a_k \cdot T(n-k) = b_1^n \cdot q_1(n) + \ldots + b_s^n \cdot q_s(n)$$

Ejemplo

- $T(n) = T(n-1) + 1$

Ecuaciones Lineales no homogéneas

- En la ecuación aparecen términos no recursivos:

  Ejemplo: $T(n) = T(n-1) + 1$

- Los términos no recursivos se pueden expresar como $\sum b_i^n \cdot q_i(n)$ donde b_i son números reales y $q_i(n)$ son polinomios en n de grado d_i.
- Forma general:

$$a_0 \cdot T(n) + a_1 \cdot T(n-1) + \ldots + a_k \cdot T(n-k) = b_1^n \cdot q_1(n) + \ldots + b_s^n \cdot q_s(n)$$

Ejemplo

- $T(n) = T(n-1) + 1$
- Hacemos $b_1 = 1$ y $q_1(n) = 1$.

Resolución de recurrencias

Ecuaciones Lineales no homogéneas

- En la ecuación aparecen términos no recursivos:

$$\textbf{Ejemplo}: \; T(n) = T(n-1) + 1$$

- Los términos no recursivos se pueden expresar como $\sum b_i^n \cdot q_i(n)$ donde b_i son números reales y $q_i(n)$ son polinomios en n de grado d_i.
- Forma general:

$$a_0 \cdot T(n) + a_1 \cdot T(n-1) + \ldots + a_k \cdot T(n-k) = b_1^n \cdot q_1(n) + \ldots + b_s^n \cdot q_s(n)$$

Ejemplo

- $T(n) = T(n-1) + 1$
- Hacemos $b_1 = 1$ y $q_1(n) = 1$.
- $T(n) = T(n-1) + b_1^n \cdot q_1(n) \rightarrow T(n) = T(n-1) + 1^n \cdot 1$.

Resolución

- Se pasan todos los términos recurrentes a un lado:

$$a_0 \cdot T(n) + a_1 \cdot T(n-1) + \ldots + a_k \cdot T(n-k) = b_1^n \cdot q_1(n) + \ldots + b_s^n \cdot q_s(n)$$

- Se resuelve la parte recurrente como si fuese homogénea:

$$a_0 \cdot T(n) + a_1 \cdot T(n-1) + a_2 \cdot T(n-2) + \ldots + a_k \cdot T(n-k) = 0$$

- Se obtiene el polinomio característico de la parte homogénea:

$$p_H(x) = a_0 \cdot x^k + a_1 \cdot x^{k-1} + a_2 \cdot x^{k-2} + \ldots + a_k$$

- El polinomio característico de la ecuación lineal no homogénea es:

$$p(x) = p_H(x) \cdot (x - b_1)^{d_1+1} \cdot (x - b_2)^{d_2+1} \ldots \cdot (x - b_s)^{d_s+1}$$

- Se aplica la fórmula para el cálculo de eficiencia como antes.

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + 1$$

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado: $T(n) - T(n-1) = 1$
2. Polinomio de la parte homogénea:

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado: $T(n) - T(n-1) = 1$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado: $T(n) - T(n-1) = 1$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $1 = b_1^n \cdot q_1(n)$

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado: $T(n) - T(n-1) = 1$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $1 = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = 1, d_1 = 0$
4. Completo:

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado: $T(n) - T(n-1) = 1$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $1 = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = 1, d_1 = 0$
4. Completo: $p(x) = (x-1) \cdot (x-b_1)^{d+1} = (x-1) \cdot (x-1) = (x-1)^2$
5. Una única raíz que aparece dos veces:

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado: $T(n) - T(n-1) = 1$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $1 = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = 1, d_1 = 0$
4. Completo: $p(x) = (x-1) \cdot (x-b_1)^{d+1} = (x-1) \cdot (x-1) = (x-1)^2$
5. Una única raíz que aparece dos veces:

$$T(n) = c_{10} \cdot R_1^n \cdot n^0 + c_{11} \cdot R_1^n \cdot n$$

Ejemplo

$$T(n) = T(n-1) + 1$$

1. En el mismo lado: $T(n) - T(n-1) = 1$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $1 = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = 1, d_1 = 0$
4. Completo: $p(x) = (x-1) \cdot (x-b_1)^{d+1} = (x-1) \cdot (x-1) = (x-1)^2$
5. Una única raíz que aparece dos veces:

$$T(n) = c_{10} \cdot R_1^n \cdot n^0 + c_{11} \cdot R_1^n \cdot n$$

6. $T(n) = c_{10} \cdot 1^n + c_{11} \cdot n \cdot 1^n$

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + n$$

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado: $T(n) - T(n-1) = n$
2. Polinomio de la parte homogénea:

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado: $T(n) - T(n-1) = n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado: $T(n) - T(n-1) = n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $n = b_1^n \cdot q_1(n)$

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado: $T(n) - T(n-1) = n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
4. Completo:

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado: $T(n) - T(n-1) = n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
4. Completo:
   $p(x) = (x-1) \cdot (x-b_1)^{d+1} = (x-1) \cdot (x-1)^2 = (x-1)^3$
5. Una única raíz que aparece tres veces:

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado: $T(n) - T(n-1) = n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
4. Completo:
   $p(x) = (x-1) \cdot (x-b_1)^{d+1} = (x-1) \cdot (x-1)^2 = (x-1)^3$
5. Una única raíz que aparece tres veces:

$$T(n) = c_{10} \cdot R_1^n \cdot n^0 + c_{11} \cdot R_1^n \cdot n + c_{12} \cdot R_1^n \cdot n^2$$

Ejemplo

$$T(n) = T(n-1) + n$$

1. En el mismo lado: $T(n) - T(n-1) = n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
4. Completo:
   $p(x) = (x-1) \cdot (x-b_1)^{d+1} = (x-1) \cdot (x-1)^2 = (x-1)^3$
5. Una única raíz que aparece tres veces:

$$T(n) = c_{10} \cdot R_1^n \cdot n^0 + c_{11} \cdot R_1^n \cdot n + c_{12} \cdot R_1^n \cdot n^2$$

6. $T(n) = c_{10} \cdot 1^n + c_{11} \cdot n \cdot 1^n + c_{12} \cdot n^2 \cdot 1^n$

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea:

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n)$

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
   - $3^n = b_2^n \cdot q_2(n)$

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
   - $3^n = b_2^n \cdot q_2(n) \Rightarrow b_2 = 3, q_2(n) = 1, d_2 = 0$
4. Completo:

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
   - $3^n = b_2^n \cdot q_2(n) \Rightarrow b_2 = 3, q_2(n) = 1, d_2 = 0$
4. Completo: $p(x) = (x-1) \cdot (x-b_1)^{d_1+1} \cdot (x-b_2)^{d_2+1} = (x-1) \cdot (x-1)^2 \cdot (x-3) = (x-1)^3 \cdot (x-3)$
5. Raíces:

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
   - $3^n = b_2^n \cdot q_2(n) \Rightarrow b_2 = 3, q_2(n) = 1, d_2 = 0$
4. Completo: $p(x) = (x-1) \cdot (x-b_1)^{d_1+1} \cdot (x-b_2)^{d_2+1} = (x-1) \cdot (x-1)^2 \cdot (x-3) = (x-1)^3 \cdot (x-3)$
5. Raíces: $r_1 = 1, m_1 = 3, r_2 = 3, m_1 = 1$

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
   - $3^n = b_2^n \cdot q_2(n) \Rightarrow b_2 = 3, q_2(n) = 1, d_2 = 0$
4. Completo: $p(x) = (x-1) \cdot (x-b_1)^{d_1+1} \cdot (x-b_2)^{d_2+1} = (x-1) \cdot (x-1)^2 \cdot (x-3) = (x-1)^3 \cdot (x-3)$
5. Raíces: $r_1 = 1, m_1 = 3, r_2 = 3, m_1 = 1$
6. $T(n) = c_{10} \cdot R_1^n \cdot n^0 + c_{11} \cdot R_1^n \cdot n + c_{12} \cdot R_1^n \cdot n^2 + c_{20} \cdot R_2^n \cdot n^0$

Ejemplo

$$T(n) = T(n-1) + n + 3^n$$

1. En el mismo lado: $T(n) - T(n-1) = n + 3^n$
2. Polinomio de la parte homogénea: $p(x) = x - 1$
3. Parte no homogénea:
   - $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
   - $3^n = b_2^n \cdot q_2(n) \Rightarrow b_2 = 3, q_2(n) = 1, d_2 = 0$
4. Completo: $p(x) = (x-1) \cdot (x-b_1)^{d_1+1} \cdot (x-b_2)^{d_2+1} = (x-1) \cdot (x-1)^2 \cdot (x-3) = (x-1)^3 \cdot (x-3)$
5. Raíces: $r_1 = 1, m_1 = 3, r_2 = 3, m_1 = 1$
6. $T(n) = c_{10} \cdot R_1^n \cdot n^0 + c_{11} \cdot R_1^n \cdot n + c_{12} \cdot R_1^n \cdot n^2 + c_{20} \cdot R_2^n \cdot n^0$
7. $T(n) = c_{10} \cdot 1^n + c_{11} \cdot n \cdot 1^n + c_{12} \cdot n^2 \cdot 1^n + c_{20} \cdot 3^n$

Cambio de variable

- Cuando $T(n)$ no está expresado en función de $T(n-k)$ es necesario hacer un cambio de variable.
- Se aplica cuando n es potencia de un número real a: $n = a^h$.
- $T(n)$ describe la recurrencia general y t_h el término de la nueva recurrencia obtenida mediante el cambio de variable.
- El polinomio característico se expresa en función de la nueva variable.

¡No hay que olvidar deshacer el cambio!

Resolución de recurrencias

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado: $t_h - t_{h-1} = 1$
4. Polinomio de la parte homogénea:

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado: $t_h - t_{h-1} = 1$
4. Polinomio de la parte homogénea: $p(x) = x - 1$
5. Parte no homogénea:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado: $t_h - t_{h-1} = 1$
4. Polinomio de la parte homogénea: $p(x) = x - 1$
5. Parte no homogénea: $1 = b_1^h \cdot q_1(h)$

Resolución de recurrencias

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado: $t_h - t_{h-1} = 1$
4. Polinomio de la parte homogénea: $p(x) = x - 1$
5. Parte no homogénea: $1 = b_1^h \cdot q_1(h) \Rightarrow b_1 = 1, q_1(h) = 1, d_1 = 0$
6. Completo:

Resolución de recurrencias

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado: $t_h - t_{h-1} = 1$
4. Polinomio de la parte homogénea: $p(x) = x - 1$
5. Parte no homogénea: $1 = b_1^h \cdot q_1(h) \Rightarrow b_1 = 1, q_1(h) = 1, d_1 = 0$
6. Completo:
   $p(x) = (x-1) \cdot (x-b_1)^{d_1+1} = (x-1) \cdot (x-1) = (x-1)^2$

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado: $t_h - t_{h-1} = 1$
4. Polinomio de la parte homogénea: $p(x) = x - 1$
5. Parte no homogénea: $1 = b_1^h \cdot q_1(h) \Rightarrow b_1 = 1, q_1(h) = 1, d_1 = 0$
6. Completo:
   $p(x) = (x-1) \cdot (x-b_1)^{d_1+1} = (x-1) \cdot (x-1) = (x-1)^2$
7. $t_h = c_{10} \cdot 1^h + c_{11} \cdot h \cdot 1^h$
8. Cambio:

Ejemplo

$$T(n) = T(n/2) + 1 \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación: $T(2^h) = T(2^{h-1}) + 1 \rightarrow t_h = t_{h-1} + 1$
3. En el mismo lado: $t_h - t_{h-1} = 1$
4. Polinomio de la parte homogénea: $p(x) = x - 1$
5. Parte no homogénea: $1 = b_1^h \cdot q_1(h) \Rightarrow b_1 = 1, q_1(h) = 1, d_1 = 0$
6. Completo:
   $p(x) = (x-1) \cdot (x-b_1)^{d_1+1} = (x-1) \cdot (x-1) = (x-1)^2$
7. $t_h = c_{10} \cdot 1^h + c_{11} \cdot h \cdot 1^h$
8. Cambio: $T(n) = c_{10} \cdot 1^{log_2(n)} + c_{11} \cdot log_2(n) \cdot 1^{log_2(n)}$

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable:

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado: $t_h - 2 \cdot t_{h-1} = 2^h$
4. Polinomio de la parte homogénea:

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado: $t_h - 2 \cdot t_{h-1} = 2^h$
4. Polinomio de la parte homogénea: $p(x) = x - 2$
5. Parte no homogénea:

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado: $t_h - 2 \cdot t_{h-1} = 2^h$
4. Polinomio de la parte homogénea: $p(x) = x - 2$
5. Parte no homogénea: $2^h = b_1^h \cdot q_1(h)$

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado: $t_h - 2 \cdot t_{h-1} = 2^h$
4. Polinomio de la parte homogénea: $p(x) = x - 2$
5. Parte no homogénea: $2^h = b_1^h \cdot q_1(h) \Rightarrow b_1 = 2, q_1(h) = 1, d_1 = 0$
6. Completo:

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado: $t_h - 2 \cdot t_{h-1} = 2^h$
4. Polinomio de la parte homogénea: $p(x) = x - 2$
5. Parte no homogénea: $2^h = b_1^h \cdot q_1(h) \Rightarrow b_1 = 2, q_1(h) = 1, d_1 = 0$
6. Completo: $p(x) = (x-2) \cdot (x-b_1)^{d_1+1} = (x-2)^2$

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de 2}$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado: $t_h - 2 \cdot t_{h-1} = 2^h$
4. Polinomio de la parte homogénea: $p(x) = x - 2$
5. Parte no homogénea: $2^h = b_1^h \cdot q_1(h) \Rightarrow b_1 = 2, q_1(h) = 1, d_1 = 0$
6. Completo: $p(x) = (x-2) \cdot (x-b_1)^{d_1+1} = (x-2)^2$
7. $t_h = c_{10} \cdot 2^h + c_{11} \cdot 2^h \cdot h$
8. Cambio:

Resolución de recurrencias

Ejemplo

$$T(n) = 2 \cdot T(n/2) + n \quad \text{si } n \text{ es potencia de } 2$$

1. Cambio de variable: $n = 2^h$ $(h = log_2(n))$
2. Se reescribe la ecuación:
   $T(2^h) = 2 \cdot T(2^{h-1}) + 2^h \rightarrow t_h = 2 \cdot t_{h-1} + 2^h$
3. En el mismo lado: $t_h - 2 \cdot t_{h-1} = 2^h$
4. Polinomio de la parte homogénea: $p(x) = x - 2$
5. Parte no homogénea: $2^h = b_1^h \cdot q_1(h) \Rightarrow b_1 = 2, q_1(h) = 1, d_1 = 0$
6. Completo: $p(x) = (x-2) \cdot (x-b_1)^{d_1+1} = (x-2)^2$
7. $t_h = c_{10} \cdot 2^h + c_{11} \cdot 2^h \cdot h$
8. Cambio:
   $T(n) = c_{10} \cdot 2^{log_2(n)} + c_{11} \cdot 2^{log_2(n)} \cdot log_2(n) = c_{10} \cdot n + c_{11} \cdot n \cdot log_2(n)$

Transformaciones del rango

- La ecuación que relaciona $T(n)$ con el resto de los términos no es lineal.
- Se resuelve intentando convertirla en una ecuación lineal de las estudiadas anteriormente.
- Esta transformación se denomina cambio de recorrido, transformación del rango o transformación del intervalo.
- Generalmente, el cambio es del tipo $U(n) = log_2(T(n))$.

¡No hay que olvidar deshacer el cambio!

Resolución de recurrencias

Ejemplo

$$T(n) = T^2(n-1)$$

Resolución de recurrencias

Ejemplo

$$T(n) = T^2(n-1)$$

1. $log_2(T(n)) = log_2(T^2(n-1)) = 2 \cdot log_2(T(n-1))$
2. Cambio:

Resolución de recurrencias

Ejemplo

$$T(n) = T^2(n-1)$$

1. $log_2(T(n)) = log_2(T^2(n-1)) = 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = T^2(n-1)$$

1. $log_2(T(n)) = log_2(T^2(n-1)) = 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = 0$
4. Ecuación:

Resolución de recurrencias

Ejemplo

$$T(n) = T^2(n-1)$$

1. $log_2(T(n)) = log_2(T^2(n-1)) = 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = 0$
4. Ecuación: $p(x) = (x-2)$
5. Raíces:

Resolución de recurrencias

Ejemplo

$$T(n) = T^2(n-1)$$

1. $log_2(T(n)) = log_2(T^2(n-1)) = 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = 0$
4. Ecuación: $p(x) = (x-2)$
5. Raíces: $r_1 = 2, m_1 = 1$

Ejemplo

$$T(n) = T^2(n-1)$$

1. $log_2(T(n)) = log_2(T^2(n-1)) = 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = 0$
4. Ecuación: $p(x) = (x-2)$
5. Raíces: $r_1 = 2, m_1 = 1$
6. $U(n) = c_{10} \cdot 2^n$
7. Deshacer el cambio ($T(n) = 2^{U(n)}$):

Ejemplo

$$T(n) = T^2(n-1)$$

1. $log_2(T(n)) = log_2(T^2(n-1)) = 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = 0$
4. Ecuación: $p(x) = (x-2)$
5. Raíces: $r_1 = 2, m_1 = 1$
6. $U(n) = c_{10} \cdot 2^n$
7. Deshacer el cambio ($T(n) = 2^{U(n)}$): $T(n) = 2^{c_{10} \cdot 2^n}$

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio:

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado:

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación:

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea:

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea: $p_H(x) = x - 2$
6. Parte no homogénea:

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea: $p_H(x) = x - 2$
6. Parte no homogénea: $n = b_1^n \cdot q_1(n)$

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea: $p_H(x) = x - 2$
6. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
7. Completo:

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea: $p_H(x) = x - 2$
6. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
7. Completo: $p(x) = (x-2) \cdot (x-b_1)^{d_1+1} = (x-2) \cdot (x-1)^2$
8. Raíces:

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea: $p_H(x) = x - 2$
6. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
7. Completo: $p(x) = (x-2) \cdot (x-b_1)^{d_1+1} = (x-2) \cdot (x-1)^2$
8. Raíces: $r_1 = 2, m_1 = 1, r_2 = 1, m_2 = 2$

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea: $p_H(x) = x - 2$
6. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
7. Completo: $p(x) = (x-2) \cdot (x-b_1)^{d_1+1} = (x-2) \cdot (x-1)^2$
8. Raíces: $r_1 = 2, m_1 = 1, r_2 = 1, m_2 = 2$
9. $U(n) = c_{10} \cdot 2^n + c_{20} + c_{21} \cdot n$

Resolución de recurrencias

Ejemplo

$$T(n) = 2^n \cdot T^2(n-1)$$

1. $log_2(T(n)) = log_2(2^n) + 2 \cdot log_2(T(n-1))$
2. Cambio: $U(n) = log_2(T(n))$
3. En el mismo lado: $U(n) - 2 \cdot U(n-1) = n$
4. Ecuación: $p(x) = (x-2)$
5. Polinomio de la parte homogénea: $p_H(x) = x - 2$
6. Parte no homogénea: $n = b_1^n \cdot q_1(n) \Rightarrow b_1 = 1, q_1(n) = n, d_1 = 1$
7. Completo: $p(x) = (x-2) \cdot (x-b_1)^{d_1+1} = (x-2) \cdot (x-1)^2$
8. Raíces: $r_1 = 2, m_1 = 1, r_2 = 1, m_2 = 2$
9. $U(n) = c_{10} \cdot 2^n + c_{20} + c_{21} \cdot n$
10. $T(n) = 2^{c_{10} \cdot 2^n + c_{20} + c_{21} \cdot n}$

Hay que deshacer el cambio. ¡No olvidarse!

Bibliografía

Bibliografía

Aclaración

- El contenido de las diapositivas es esquemático y representa un apoyo para las clases teóricas.
- Se recomienda completar los contenidos del tema 1 con apuntes propios tomados en clase y con la bibliografía principal de la asignatura.

Por ejemplo


G. Brassard and P. Bratley.
***Fundamentals of Algorithmics*.**
Prentice Hall, Englewwod Cliffs, New Jersey, 1996.


J. L. Verdegay.
***Lecciones de Algorítmica*.**
Editorial Técnica AVICAM, 2017.