

# Introducción

IIC2283

# El objetivo de este curso

Introducir técnicas tanto para el **diseño** como para el **análisis de la complejidad computacional** de un **algoritmo**

- ▶ Técnicas básicas y avanzadas

Se dará énfasis a:

- ▶ La comprensión del modelo computacional sobre el cual se diseña y analiza un algoritmo
- ▶ El uso de estructuras de datos adecuadas para la implementación de un algoritmo
- ▶ El uso de ejemplos de distintas áreas para mostrar las potencialidades de las técnicas estudiadas

# ¿Qué es un algoritmo?

¿Podemos formalizar este concepto?

- ▶ **Máquinas de Turing:** Intento por formalizar este concepto

¿Podemos demostrar que las máquinas de Turing capturan la noción de algoritmo?

- ▶ No, el concepto de algoritmo es intuitivo

# Máquinas de Turing

¿Por qué creemos que las máquinas de Turing son una buena formalización del concepto de algoritmo?

- ▶ Porque cada **programa** de una máquina de Turing puede ser implementado
- ▶ Porque todos los algoritmos conocidos han podido ser implementados en máquinas de Turing
- ▶ Porque todos los otros intentos por formalizar este concepto fueron reducidos a las máquinas de Turing
  - ▶ Los mejores intentos resultaron ser equivalentes a las máquinas de Turing
  - ▶ Todos los intentos “razonables” fueron reducidos **eficientemente**
- ▶ Tesis de Church: **Algoritmo = Máquina de Turing**

## Definición

*Máquina de Turing:  $(Q, \Sigma, \Gamma, q_0, \delta, F)$*

- ▶  $Q$  es un conjunto finito de estados
- ▶  $\Sigma$  es un alfabeto tal que  $B \notin \Sigma$
- ▶  $\Gamma$  es un alfabeto tal que  $\Sigma \cup \{B\} \subseteq \Gamma$
- ▶  $q_0 \in Q$  es el estado inicial
- ▶  $F \subseteq Q$  es un conjunto de estados finales
- ▶  $\delta$  es una función parcial:

$$\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{\leftarrow, \square, \rightarrow\}$$

$\delta$  es llamada función de transición

# Máquinas de Turing: Funcionamiento

La cinta de la máquina de Turing es infinita en ambas direcciones.

$\Sigma$  es el alfabeto de entrada y  $\Gamma$  es el alfabeto de la cinta.

- ▶ Una palabra de entrada  $w \in \Sigma^*$  es colocada en posiciones consecutivas de la cinta
- ▶ Todas las otras posiciones contienen el símbolo B

# Máquinas de Turing: Funcionamiento

Al comenzar a funcionar, la máquina se encuentra en el estado  $q_0$  y su cabeza lectora está en la posición inicial de la palabra de entrada

En cada instante la máquina se encuentra en un estado  $q$  y su cabeza lectora está en una posición  $p$

- ▶ Si el símbolo en la posición  $p$  es  $a$  y  $\delta(q, a) = (q', b, X)$ , entonces:
  - ▶ La máquina escribe el símbolo  $b$  en la posición  $p$  de la cinta
  - ▶ Cambia de estado desde  $q$  a  $q'$
  - ▶ Mueve la cabeza lectora a la posición  $p - 1$  si  $X$  es  $\leftarrow$ , y a la posición  $p + 1$  si  $X$  es  $\rightarrow$ . Si  $X$  es  $\square$ , entonces la cabeza lectora permanece en la posición  $p$

# Máquinas de Turing: Aceptación

Los estados de  $F$  son utilizados como estados de aceptación.

- Una palabra  $w$  es aceptada por una máquina  $M$  si y sólo si la ejecución de  $M$  con entrada  $w$  se detiene en un estado de  $F$

## Definición

*Lenguaje aceptado por una máquina de Turing  $M$ :*

$$L(M) = \{w \in \Sigma^* \mid M \text{ acepta } w\}$$

# Máquinas de Turing: Ejemplo

Queremos construir una máquina de Turing  $M$  que acepte el lenguaje  $L = \{w \in \{0,1\}^* \mid w \text{ es un palíndromo}\}$

Definimos  $M = (Q, \Sigma, \Gamma, q_0, \delta, F)$  de la siguiente forma:

- ▶  $Q = \{q_0, q_f, q_{\rightarrow}^0, q_{\rightarrow}^1, q_v^0, q_v^1, q_{\leftarrow}\}$
- ▶  $\Sigma = \{0, 1\}$
- ▶  $\Gamma = \{0, 1, B\}$
- ▶  $F = \{q_f\}$

# Máquinas de Turing: Ejemplo

►  $\delta$  es definida como:

$$\delta(q_0, B) = (q_f, B, \square)$$

$$\delta(q_0, 0) = (q_{\rightarrow}^0, B, \rightarrow)$$

$$\delta(q_0, 1) = (q_{\rightarrow}^1, B, \rightarrow)$$

$$\delta(q_{\leftarrow}, B) = (q_0, B, \rightarrow)$$

$$\delta(q_{\leftarrow}, 0) = (q_{\leftarrow}, 0, \leftarrow)$$

$$\delta(q_{\leftarrow}, 1) = (q_{\leftarrow}, 1, \leftarrow)$$

$$\delta(q_{\rightarrow}^0, B) = (q_v^0, B, \leftarrow)$$

$$\delta(q_{\rightarrow}^0, 0) = (q_{\rightarrow}^0, 0, \rightarrow)$$

$$\delta(q_{\rightarrow}^0, 1) = (q_{\rightarrow}^0, 1, \rightarrow)$$

$$\delta(q_v^0, B) = (q_f, B, \square)$$

$$\delta(q_v^0, 0) = (q_{\leftarrow}, B, \leftarrow)$$

$$\delta(q_{\rightarrow}^1, B) = (q_v^1, B, \leftarrow)$$

$$\delta(q_{\rightarrow}^1, 0) = (q_{\rightarrow}^1, 0, \rightarrow)$$

$$\delta(q_{\rightarrow}^1, 1) = (q_{\rightarrow}^1, 1, \rightarrow)$$

$$\delta(q_v^1, B) = (q_f, B, \square)$$

$$\delta(q_v^1, 1) = (q_{\leftarrow}, B, \leftarrow)$$

¿Cuál es la complejidad del algoritmo anterior?

- ▶ ¿Qué operaciones contamos para realizar este análisis?

Para una MT  $M$  con alfabeto de entrada  $\Sigma$ :

- ▶ **Paso de  $M$ :** Ejecutar una instrucción de la función de transición
- ▶  **$tiempo_M(w)$ :** Número de pasos ejecutados por  $M$  con entrada  $w \in \Sigma^*$

## Supuesto

Consideramos máquinas de Turing que se detienen en todas sus entradas.

- ▶ En general consideramos algoritmos que se detienen en todas sus entradas

## Definición

El tiempo de funcionamiento de una MT  $M$  en el *peor caso* es definido por la función  $t_M$ :

$$t_M(n) = \text{máx}\{ \text{tiempo}_M(w) \mid w \in \Sigma^* \text{ y } |w| = n \}$$

## Ejercicio

Para la MT  $M$  del ejemplo sobre palíndromos demuestre que  $t_M(n)$  es  $O(n^2)$

## Definición

*Máquina de Turing con  $k$  cintas:*  $(Q, \Sigma, \Gamma, q_0, \delta, F)$

- ▶  $Q$  es un conjunto finito de estados
- ▶  $\Sigma$  es un alfabeto tal que  $B \notin \Sigma$
- ▶  $\Gamma$  es un alfabeto tal que  $\Sigma \cup \{B\} \subseteq \Gamma$
- ▶  $q_0 \in Q$  es el estado inicial
- ▶  $F \subseteq Q$  es un conjunto de estados finales
- ▶  $\delta$  es una función parcial:

$$\delta : Q \times \Gamma^k \rightarrow Q \times \Gamma^k \times \{\leftarrow, \square, \rightarrow\}^k$$

$\delta$  es llamada función de transición.

# MT con $k$ cintas: Funcionamiento

La máquina tiene  $k$  cintas infinitas en ambas direcciones.

$\Sigma$  es el alfabeto de entrada y  $\Gamma$  es el alfabeto de las cintas.

- ▶ Una palabra de entrada  $w \in \Sigma^*$  es colocada en posiciones consecutivas de la primera cinta
- ▶ Las otras posiciones de la primera cinta contienen el símbolo B
- ▶ Las restantes cintas contienen el símbolo B en todas las posiciones

# MT con $k$ cintas: Funcionamiento

La máquina tiene una cabeza lectora por cinta.

Al comenzar la máquina se encuentra en el estado  $q_0$ , la cabeza lectora de la primera cinta está en la primera posición de la palabra de entrada, el resto de las cabezas están en posiciones arbitrarias.

# MT con $k$ cintas: Funcionamiento

En cada instante la máquina se encuentra en un estado  $q$  y su cabeza lectora  $i$  se encuentra en la posición  $p_i$ .

- ▶ Si el símbolo en la posición  $p_i$  es  $a_i$  y  $\delta(q, a_1, \dots, a_k) = (q', b_1, \dots, b_k, X_1, \dots, X_k)$ , entonces:
  - ▶ La máquina escribe el símbolo  $b_i$  en la posición  $p_i$  de la  $i$ -ésima cinta
  - ▶ Cambia de estado desde  $q$  a  $q'$
  - ▶ Mueve la cabeza lectora de la  $i$ -ésima cinta a la posición  $p_i - 1$  si  $X_i$  es  $\leftarrow$ , y a la posición  $p_i + 1$  si  $X_i$  es  $\rightarrow$ . Si  $X_i$  es  $\square$ , entonces la máquina no mueve la cabeza lectora de la  $i$ -ésima cinta

# MT con $k$ cintas: Aceptación y complejidad

Una palabra  $w$  es aceptada por una MT  $M$  con  $k$  cintas si y sólo si la ejecución de  $M$  con entrada  $w$  se detiene en un estado final. Tenemos entonces que:

$$L(M) = \{w \in \Sigma^* \mid M \text{ acepta } w\}$$

Para una MT con  $k$  cintas y alfabeto  $\Sigma$ :

- ▶ **Paso de  $M$ :** Ejecutar una instrucción de la función de transición
- ▶  **$tiempo_M(w)$ :** Número de pasos ejecutados por  $M$  con entrada  $w \in \Sigma^*$
- ▶ **Tiempo de funcionamiento  $M$  en el peor caso:**

$$t_M(n) = \max\{ tiempo_M(w) \mid w \in \Sigma^* \text{ y } |w| = n \}$$

# MT con $k$ cintas: Ejemplo

Construya una MT  $M$  con dos cintas que funcione en tiempo  $O(n)$  y acepte el lenguaje  $L = \{w \in \{0, 1\}^* \mid w \text{ es un palíndromo}\}$ .

**Solución:** Definimos  $M = (Q, \Sigma, \Gamma, q_0, \delta, F)$  de la siguiente forma:

- ▶  $Q = \{q_0, q_c, q_r, q_v, q_f\}$
- ▶  $\Sigma = \{0, 1\}$
- ▶  $\Gamma = \{0, 1, B\}$
- ▶  $F = \{q_f\}$

# MT con $k$ cintas: Ejemplo

- Función  $\delta$  es definida de la siguiente forma:

|                     |     |                                         |                     |     |                                        |
|---------------------|-----|-----------------------------------------|---------------------|-----|----------------------------------------|
| $\delta(q_0, B, B)$ | $=$ | $(q_f, B, B, \square, \square)$         | $\delta(q_r, 1, 0)$ | $=$ | $(q_r, 1, 0, \leftarrow, \square)$     |
| $\delta(q_0, 0, B)$ | $=$ | $(q_c, 0, 0, \rightarrow, \rightarrow)$ | $\delta(q_r, 1, 1)$ | $=$ | $(q_r, 1, 1, \leftarrow, \square)$     |
| $\delta(q_0, 1, B)$ | $=$ | $(q_c, 1, 1, \rightarrow, \rightarrow)$ | $\delta(q_r, B, 0)$ | $=$ | $(q_v, B, 0, \rightarrow, \square)$    |
| $\delta(q_c, 0, B)$ | $=$ | $(q_c, 0, 0, \rightarrow, \rightarrow)$ | $\delta(q_r, B, 1)$ | $=$ | $(q_v, B, 1, \rightarrow, \square)$    |
| $\delta(q_c, 1, B)$ | $=$ | $(q_c, 1, 1, \rightarrow, \rightarrow)$ | $\delta(q_v, 0, 0)$ | $=$ | $(q_v, 0, 0, \rightarrow, \leftarrow)$ |
| $\delta(q_c, B, B)$ | $=$ | $(q_r, B, B, \leftarrow, \leftarrow)$   | $\delta(q_v, 1, 1)$ | $=$ | $(q_v, 1, 1, \rightarrow, \leftarrow)$ |
| $\delta(q_r, 0, 0)$ | $=$ | $(q_r, 0, 0, \leftarrow, \square)$      | $\delta(q_v, B, B)$ | $=$ | $(q_f, B, B, \square, \square)$        |
| $\delta(q_r, 0, 1)$ | $=$ | $(q_r, 0, 1, \leftarrow, \square)$      |                     |     |                                        |

# Complejidad en distintos modelos

¿Es posible aceptar más rápido si se usa cintas adicionales?

- ▶ ¿Tenemos entonces que especificar bajo que modelo estamos resolviendo un problema?

Sea  $L = \{w \in \{0, 1, \#\}^* \mid w \text{ es un palíndromo}\}$

- ▶  $L$  es aceptado por una MT con dos cintas en tiempo  $O(n)$
- ▶ ¿Puede ser  $L$  aceptado en tiempo lineal por una MT con una cinta?

## Proposición

*Sea  $L = \{w \in \{0, 1, \#\}^* \mid w \text{ es un palíndromo}\}$  y  $M$  una MT con una cinta. Si  $L = L(M)$ , entonces  $M$  funciona en tiempo  $\Omega(n^2)$ .*

**Demostración:** Suponga que  $L = L(M)$ , donde  $M$  es una MT con una cinta.

- Sea  $Q$  el conjunto de estados de  $M$

# Una cota inferior en un modelo de computación

Sin pérdida de generalidad, suponemos que  $M$  siempre recorre toda la palabra de entrada.

- ¿Por qué podemos suponer esto?

Para  $w \in \{0, 1, \#\}^*$ , sea  $w^r$  la palabra obtenida al escribir  $w$  en el sentido inverso.

Defina  $L_n$  como el siguiente lenguaje (para  $n > 0$  divisible por 4):

$$L_n = \{w\#^{\frac{n}{2}}w^r \mid w \in \{0, 1\}^{\frac{n}{4}}\}$$

Nótese que  $L_n \subseteq L$

# Complejidad en distintos modelos

Sea  $w \in L_n$  y  $\frac{n}{4} \leq i \leq \frac{3n}{4}$ . Además, sea  $C_i(w)$  la secuencia de estados  $[q_1, \dots, q_k]$  en que se encuentra  $M$  después de moverse entre las posiciones  $i$  e  $i + 1$  (en cualquiera de las dos direcciones) en la ejecución que tiene a  $w$  como entrada.

Definimos  $C(w) = \{C_i(w) \mid \frac{n}{4} \leq i \leq \frac{3n}{4}\}$

# Complejidad en distintos modelos

## Lema

Si  $w_1, w_2 \in L_n$  y  $w_1 \neq w_2$ , entonces  $C(w_1) \cap C(w_2) = \emptyset$

**Demostración:** Suponga que el lema es falso. Entonces existen  $i, j \in \{\frac{n}{4}, \dots, \frac{3n}{4}\}$  tales que  $C_i(w_1) = C_j(w_2)$ .

Sean  $u_1$  y  $u_2$  las palabras formadas por los primeros  $i$  símbolos de  $w_1$  y los últimos  $n - j$  símbolos de  $w_2$ , respectivamente.

Dado que  $C_i(w_1) = C_j(w_2)$ , se tiene que  $u_1 u_2$  es aceptado por  $M$ .

► ¿Cómo se demuestra esto?

Pero  $u_1 u_2$  no es un palíndromo, por lo que tenemos una contradicción.



# Complejidad en distintos modelos

Para  $w \in L_n$ , sea  $s_w$  la secuencia más corta en  $C(w)$

$$\blacktriangleright S_n = \{s_w \mid w \in L_n\}$$

Por el lema sabemos que  $s_{w_1} \neq s_{w_2}$  si  $w_1 \neq w_2$

$$\blacktriangleright \text{Por lo tanto: } |S_n| = |L_n| = 2^{\frac{n}{4}}$$

Sea  $m$  el largo de la secuencia mas larga en  $S_n$

$\blacktriangleright$  Cantidad de posibles secuencias de largo a lo más  $m$ :

$$\sum_{i=0}^m |Q|^i = \frac{|Q|^{m+1} - 1}{|Q| - 1}$$

# Complejidad en distintos modelos

De lo anterior concluimos que:  $\frac{|Q|^{m+1}-1}{|Q|-1} \geq 2^{\frac{n}{4}}$

- ¿Por qué?

Se tiene entonces que  $m$  es  $\Omega(n)$

- Por lo tanto existe  $w_0 \in L_n$  para el cual  $|s_{w_0}|$  es  $\Omega(n)$

Entonces: Todas las secuencias en  $C(w_0)$  son de largo  $\Omega(n)$

Conclusión: Con entrada  $w_0$ , la máquina  $M$  toma tiempo  $\Omega(n^2)$

- Puesto que  $M$  tiene que generar  $\frac{n}{2}$  secuencias de estados de largo  $\Omega(n)$



# ¿Qué hemos aprendido?

Al diseñar un algoritmo debemos considerar el modelo de computación sobre el cual será implementado.

- ▶ ¿Qué operaciones podemos realizar en el modelo?

Al analizar la complejidad computacional de un algoritmo también debemos considerar el modelo de computación.

- ▶ ¿Qué operaciones consideramos al analizar la complejidad de un algoritmo?

# ¿Qué hemos aprendido?

En general, el análisis de la complejidad de un algoritmo se realiza considerando un tipo particular de entradas.

- ▶ El peor caso es muy utilizado, pero también podemos considerar el caso promedio

Al estudiar un problema debemos tener en cuenta que cotas inferiores se puede demostrar para su complejidad

- ▶ Estas cotas inferiores dependen del modelo de computación considerado

# ¿Qué consideraciones adicionales debemos tener?

Debemos considerar modelos de computación que representen el funcionamiento de una arquitectura de computadores.

- ▶ Por ejemplo, debemos considerar acceso directo a los datos y la diferencia de costo entre el uso de memoria principal y secundaria
- ▶ Las máquinas de Turing pueden no ser apropiadas en este sentido.  
¿Por qué las usamos entonces?

Vamos a considerar dos ejemplos que nos servirán para ilustrar los puntos anteriores: ordenación y la multiplicación de números enteros

# Ordenación de una lista

El siguiente es un algoritmo clásico para ordenar una lista  $L$  de números enteros (de menor a mayor).

**InsertionSort**( $L[1 \dots n]$ : lista de números enteros)

**for**  $i := 2$  **to**  $n$  **do**

$j := i - 1$

**while**  $j \geq 1$  **and**  $L[j] > L[j + 1]$  **do**

$aux := L[j]$

$L[j] := L[j + 1]$

$L[j + 1] := aux$

$j := j - 1$

**return**  $L$

# Análisis de la complejidad de insertion sort

Consideramos la comparación como la operación a contar, la cual tiene costo 1.

Dada una lista  $L$  con  $n$  números enteros.

- ▶ ¿Cuál es el peor caso del algoritmo?
- ▶ ¿Cuántas comparaciones realiza el algoritmo en el peor caso, como una función de  $n$ ?

¿Qué otras operaciones son realizadas por el algoritmo? ¿Cambia el orden del algoritmo si las consideramos?

# Una versión mejorada de insertion sort

Suponiendo que  $L[1 \cdots (i - 1)]$  ya ha sido ordenada (de menor a mayor), el paso básico del algoritmo es encontrar la posición donde debería ser ubicado  $L[i]$

- ▶ Además el algoritmo debe colocar  $L[i]$  en esta posición

Para disminuir el tiempo de ejecución del algoritmo podríamos utilizar búsqueda binaria para encontrar la posición correcta para  $L[i]$

# Una versión mejorada de insertion sort

Consideramos nuevamente la comparación como la operación a contar.

Dado que búsqueda binaria realiza  $O(\log_2(i))$  comparaciones para encontrar la posición correcta para  $L[i]$ , el algoritmo mejorado es de orden  $O(n \cdot \log_2(n))$

- Esto puede ser deducido utilizando lo siguiente:

$$\begin{aligned}\sum_{i=1}^n \log_2(i) &= \log_2 \left( \prod_{i=1}^n i \right) \\ &= \log_2(n!) \\ &\leq \log_2(n^n) \\ &= n \cdot \log_2(n)\end{aligned}$$

# ¿Una versión mejorada de insertion sort?

¿Es más eficiente el algoritmo que utiliza búsqueda binaria?

- ▶ ¿Ve algún problema en este algoritmo?

Un problema con este algoritmo: ¿Cómo colocar de manera eficiente  $L[i]$  en la posición correcta en  $L[1 \dots (i - 1)]$  ?

- ▶ Un algoritmo ingenuo toma tiempo lineal en este paso, por lo que el tiempo total del algoritmo sería  $O(n^2)$ , el mismo orden que para insertion sort

# ¿Una versión mejorada de insertion sort?

## Dos lecciones importantes

- ▶ Una operación puede ser cambiada por otra en un algoritmo, esto debe tenerse en cuenta al momento de analizar su complejidad.
- ▶ El uso de estructuras de datos es fundamental para implementar de manera eficiente las operaciones requeridas por un algoritmo.

# Suma de números enteros

Sean  $a$  y  $b$  dos números enteros con  $n \geq 1$  dígitos cada uno.

Primero queremos obtener  $c = a + b$

Vamos a utilizar el algoritmo usual para calcular  $c$

- ▶ Consideramos la **suma de dos dígitos**, la **comparación de dos dígitos** y la **resta de un número con a lo más dos dígitos con uno de un dígito** como las operaciones a contar, cada una con costo 1. ¿Tiene sentido suponer que todas tienen el mismo costo?
- ▶ ¿Cuál es el peor caso para el algoritmo usual? ¿Cuántas operaciones realiza el algoritmo en este caso?
- ▶ ¿Cuántos dígitos puede tener  $c$ ?

# Multiplicación de números enteros

Ahora queremos obtener  $d = a \cdot b$

Primero vamos a utilizar el algoritmo usual para calcular  $d$

- ▶ Consideramos la suma y la multiplicación de dígitos como las operaciones a contar, ambas con costo 1. ¿Tienes sentido suponer que ambas operaciones tienen el mismo costo?
- ▶ ¿Cuál es el peor caso para el algoritmo usual? ¿Cuántas operaciones realiza el algoritmo en este caso?
- ▶ ¿Cuántos dígitos puede tener  $d$ ?

# El algoritmo de multiplicación de Karatsuba

Suponga que  $a$  y  $b$  son dos números con  $n$  dígitos cada uno, donde  $n$  es una potencia de 2.

Podemos representar  $a$  y  $b$  de la siguiente forma:

$$\begin{aligned}a &= a_1 \cdot 10^{\frac{n}{2}} + a_2 \\ b &= b_1 \cdot 10^{\frac{n}{2}} + b_2\end{aligned}$$

Tenemos entonces que:

$$a \cdot b = a_1 \cdot b_1 \cdot 10^n + (a_1 \cdot b_2 + a_2 \cdot b_1) \cdot 10^{\frac{n}{2}} + a_2 \cdot b_2$$

# El algoritmo de multiplicación de Karatsuba

Para calcular  $a \cdot b$  entonces debemos calcular las siguientes multiplicaciones:

1.  $a_1 \cdot b_1$
2.  $a_1 \cdot b_2$
3.  $a_2 \cdot b_1$
4.  $a_2 \cdot b_2$

Obtenemos entonces un algoritmo recursivo

- Para resolver el caso de largo  $n$  realizamos 4 llamadas para los casos de largo  $\frac{n}{2}$

¿Cómo llamamos a este tipo de algoritmos?

- ¿Cómo medimos su complejidad?
- ¿Con las 4 llamadas anteriores obtenemos un algoritmo más eficiente que el algoritmo usual de multiplicación?

# La idea clave en el algoritmo de Karatsuba

Para calcular  $a \cdot b$  realizamos las siguientes multiplicaciones:

1.  $c_1 = a_1 \cdot b_1$
2.  $c_2 = a_2 \cdot b_2$
3.  $c_3 = (a_1 + a_2) \cdot (b_1 + b_2)$

Tenemos entonces que:

$$a \cdot b = c_1 \cdot 10^n + (c_3 - (c_1 + c_2)) \cdot 10^{\frac{n}{2}} + c_2$$

¿Cuántas operaciones realiza este algoritmo?

# El tiempo de ejecución del algoritmo de Karatsuba

$T(n)$ : número de operaciones realizadas por el algoritmo de Karatsuba para dos números de entrada con  $n$  dígitos cada uno en el peor caso

Para determinar el orden de  $T(n)$  utilizamos una **ecuación de recurrencia**:

$$T(n) = \begin{cases} 1 & n = 1 \\ 3 \cdot T(\frac{n}{2}) + e \cdot n & n > 1 \end{cases}$$

# El tiempo de ejecución del algoritmo de Karatsuba

¿Qué supuestos realizamos al formular esta ecuación?

- ▶  $n$  es una potencia de 2
- ▶  $(a_1 + a_2)$  y  $(b_1 + b_2)$  tienen  $\frac{n}{2}$  dígitos cada uno

¿Qué representa la constante  $e$ ? Utilizamos  $e$  para codificar el número de operaciones necesarias para:

- ▶ Calcular  $(a_1 + a_2)$ ,  $(b_1 + b_2)$ ,  $(c_1 + c_2)$  y  $(c_3 - (c_1 + c_2))$
- ▶ Construir  $a \cdot b$  a partir de  $c_1$ ,  $c_2$  y  $(c_3 - (c_1 + c_2))$ , lo cual puede tomar tiempo lineal en el peor caso. ¿Por qué?

# Resolviendo una ecuación de recurrencia

¿Cómo resolvemos la ecuación de recurrencia para  $T(n)$

Como  $n = 2^k$  tenemos que:

$$T(2^k) = \begin{cases} 1 & k = 0 \\ 3 \cdot T(2^{k-1}) + e \cdot 2^k & k > 0 \end{cases}$$

# Resolviendo una ecuación de recurrencia

Para resolver la ecuación simplemente la desarrollamos:

$$\begin{aligned}T(2^k) &= 3 \cdot T(2^{k-1}) + e \cdot 2^k \\&= 3 \cdot (3 \cdot T(2^{k-2}) + e \cdot 2^{k-1}) + e \cdot 2^k \\&= 3^2 \cdot T(2^{k-2}) + 3 \cdot e \cdot 2^{k-1} + e \cdot 2^k \\&= 3^2 \cdot (3 \cdot T(2^{k-3}) + e \cdot 2^{k-2}) + 3 \cdot e \cdot 2^{k-1} + e \cdot 2^k \\&= 3^3 \cdot T(2^{k-3}) + 3^2 \cdot e \cdot 2^{k-2} + 3 \cdot e \cdot 2^{k-1} + e \cdot 2^k \\&= 3^3 \cdot T(2^{k-3}) + e \cdot 2^k \cdot \left( \left( \frac{3}{2} \right)^2 + \frac{3}{2} + 1 \right) \\&\dots \\&= 3^i \cdot T(2^{k-i}) + e \cdot 2^k \cdot \sum_{j=0}^{i-1} \left( \frac{3}{2} \right)^j\end{aligned}$$

# Resolviendo una ecuación de recurrencia

Considerando  $i = k$  obtenemos:

$$T(2^k) = 3^k \cdot T(1) + e \cdot 2^k \cdot \sum_{j=0}^{k-1} \left(\frac{3}{2}\right)^j$$

Tenemos entonces que:

$$\begin{aligned} T(2^k) &= 3^k + e \cdot 2^k \cdot \left( \frac{\left(\frac{3}{2}\right)^k - 1}{\frac{3}{2} - 1} \right) \\ &= 3^k + 2 \cdot e \cdot 2^k \cdot \left( \frac{3^k - 2^k}{2^k} \right) \\ &= 3^k + 2 \cdot e \cdot (3^k - 2^k) \\ &= 3^k \cdot (1 + 2 \cdot e) - 2 \cdot e \cdot 2^k \end{aligned}$$

Considerando que  $k = \log_2(n)$  obtenemos:

$$T(n) = 3^{\log_2(n)} \cdot (1 + 2 \cdot e) - 2 \cdot e \cdot n$$

# Resolviendo una ecuación de recurrencia

Finalmente tenemos que:

$$\begin{aligned} 3^{\log_2(n)} &= 2^{\log_2(3^{\log_2(n)})} \\ &= 2^{\log_2(n) \cdot \log_2(3)} \\ &= (2^{\log_2(n)})^{\log_2(3)} \\ &= n^{\log_2(3)} \end{aligned}$$

De lo cual concluimos que  $T(n) = n^{\log_2(3)} \cdot (1 + 2 \cdot e) - 2 \cdot e \cdot n$

- Vale decir,  $T(n)$  es  $O(n^{\log_2(3)})$

¿Es válido este resultado para todo  $n$ ? ¿Qué sucede si eliminamos los supuestos?

- Vamos a estudiar algunas herramientas para contestar este tipo de preguntas, usted va a resolver el caso del algoritmo de Karatsuba