Complejidad

- **Introducción**
- Esquema general
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Trata-bles e Intratables
- Clases de problema
  - P, NP y NP-completos
- EJEMPLOS 1, 2, 3
  - Resumen

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

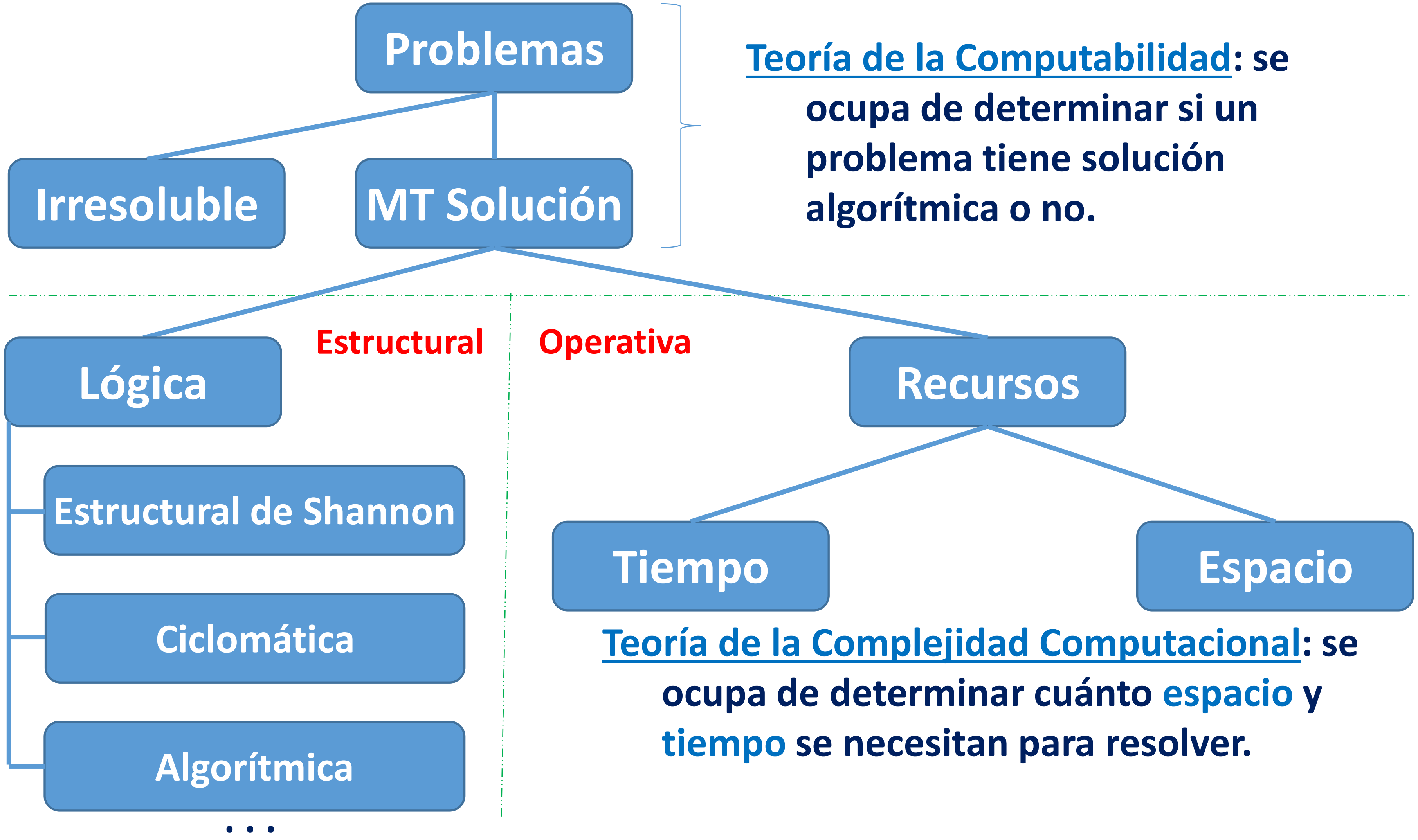
COMPLEJIDAD DE LOS PROBLEMAS

- Definimos en la Unidad 6, la **Máquina de Turing** como una máquina abstracta **MT = (Σ, Γ, Q, q_0, A, *f* , ƀ)** que se utiliza para **reconocer lenguajes** y **ejecutar procedimientos**.
- La **Tesis de Turing-Church** conjetura que **una MT es capaz de computar cualquier función calculable**, estableciendo el límite de lo computable.
- Hoy la Ciencia de la Computación considera que un **problema es computable si y sólo sí puede construirse una MT que lo resuelva**: **una solución**.

Ahora, una vez establecida la **computabilidad** de un problema, queda como una cuestión importante: **¿es la mejor solución?**

Complejidad
- Introducción
- **Esquema general**
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Tratables e Intratables
- Clases de problema
  - P, NP y NP-completos
- EJEMPLOS 1, 2, 3
  - Resumen

COMPLEJIDAD DE LOS PROBLEMAS



Teoría de la Computabilidad: se ocupa de determinar si un problema tiene solución algorítmica o no.

Teoría de la Complejidad Computacional: se ocupa de determinar cuánto espacio y tiempo se necesitan para resolver.

Complejidad

- Introducción
- Esquema general
- **Concepto**
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Trata-bles e Intratables
- Clases de problema
  - P, NP y NP-completos
- EJEMPLOS 1, 2, 3
  - Resumen

COMPLEJIDAD DE LOS PROBLEMAS

Estudiar la **complejidad de los problemas** resulta subjetivo, ya que un problema parece menos complejo a medida que quien lo resuelve adquiere nuevos conocimientos y experiencia.

Solemos asociar **complejidad** con **complicación** pero suelen ser cosas distintas, e inclusive la palabra **complejidad** tiene diversos significados en las distintas ciencias y disciplinas.

Si bien deseamos estudiar la **complejidad de los problemas**, en realidad lo que se aborda es el estudio de la **complejidad de las soluciones** que hemos podido construir para esos problemas, en el convencimiento de que problemas complejos requieren soluciones complejas **(Análisis de Algoritmos)**.

COMPLEJIDAD DE LAS SOLUCIONES

En **Informática**, a mi criterio, podemos hablar en principio de **dos tipos** de **complejidad de las soluciones**, cuyo estudio e importancia obedecerán a los distintos objetivos que se persigan:

- Complejidad Lógica: Mide la **sencillez** de la solución que hemos podido construir para el problema; qué tan **fácil** resulta de **entender** y, consecuentemente, de **modificar**. **Importante en una empresa de Sistemas.**
- Complejidad de Recursos: Mide **cuántos recursos insume** la ejecución de la solución que hemos construido; los recursos críticos suelen ser el **tiempo** que demora la solución en dar el resultado y el **espacio** de memoria necesario para alojarla. **Importante en Computación y en Sistemas.**

Complejidad

COMPLEJIDAD DE LAS SOLUCIONES

MT Solución — Una o varias soluciones al problema planteado

- **Estructural**: **Lógica** — Sencillez, mantenibilidad y elegancia de la solución
  - **Estructural de Shannon** — $C(MT) = |Q| x |\Gamma|$ ¡Nro. de flechas!
  - **Ciclomática** — Ingeniería de Software
  - **Algorítmica** — Teoría de la Aleatoriedad
- **Operativa**: **Recursos** — Que insume la solución
  - **Tiempo** — T(n) # de movimientos
  - **Espacio** — E(n) # de celdas usadas
  - n=largo(α)
  - Teoría de la complejidad computacional

Queremos analizar la **complejidad de las soluciones** tratando de ser lo más **precisos** y **objetivos** posible. Además buscamos que nuestro estudio sea **reproducible** y **comparable**, para que pueda ser validado por todos.

Complejidad

- Introducción
- Esquema general
- **Concepto**
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Tratables e Intratables
- Clases de problema
  - P, NP y NP-completos
- EJEMPLOS 1, 2, 3
  - Resumen

COMPLEJIDAD DE LAS SOLUCIONES

Para lograr **precisión**, **objetividad** y **reproducibilidad** se recomienda:

- Utilizar un **modelo de computación uniforme** y **bien conocido** para expresar la solución: **Máquina de Turing determinista de una cinta**.
- Un problema puede tener más de una **Máquina de Turing** que lo resuelva. Se considerará que la **complejidad del problema** está dada por la **complejidad de la mejor MT solución construida**.
- En el caso de la **complejidad computacional**, medida por los **recursos** que insume la solución, claramente estos recursos (**tiempo** y **espacio**) dependen de los datos de entrada en cada corrida. Por ello:
  - Del mejor, peor y caso promedio, se tomará el **peor caso**.
  - Se establecerá **cómo cambian** los recursos insumidos **en función del largo los datos de entrada** como una medida de la complejidad, esto es, se busca **una función *f*(n)**.

COMPLEJIDAD DE LAS SOLUCIONES

Antes de ver la **complejidad computacional** en máquinas de Turing, comentaremos sobre algunos métodos para establecer una **complejidad lógica**: la complicación intrínseca de la solución.

- **Lógica**
  - **Estructural de Shannon**
  - Ciclomática
  - Algorítmica

Complejidad Estructural de Shannon: Shannon propuso una medida de complejidad para la máquina de Turing, que mide la cantidad de conexiones en su grafo (≅ número de flechas):

$$\textbf{Complejidad} = |Q| \times |\Gamma|$$

También demostró equivalencias interesantes sobre las máquinas de Turing:

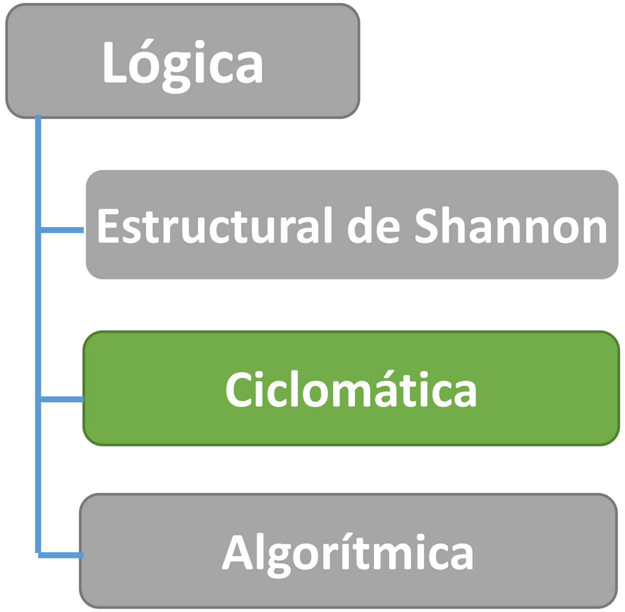
Teorema 1: Cualquier **MT** con $|\Gamma|=m$ y $|Q|=n$ puede ser simulada por una **MT'** con **sólo dos estados** y **4.m.n+m símbolos**.

Teorema 2: Cualquier **MT** con $|\Gamma|=m$ y $|Q|=n$ puede ser simulada por una **MT'** con **sólo dos símbolos** y **menos de 8.m.n estados**.

COMPLEJIDAD DE LAS SOLUCIONES

Antes de ver la **complejidad computacional** en máquinas de Turing, comentaremos sobre algunos métodos para establecer una **complejidad lógica**: la complicación intrínseca de la solución.

FUERA DE APUNTE



Complejidad Ciclomática: En la década de 1970 se desarrolló la programación estructurada y se especificó como una posible medida de la complejidad de los algoritmos, el **número de regiones cerradas** en un diagrama de flujo estructurado, esto es, en el mapa de su grafo plano. Sabemos por Eüler que si **G = (N, S)** es un multigrafo plano, su mapa tendrá:

Regiones Cerradas = |S| - |N| + 1

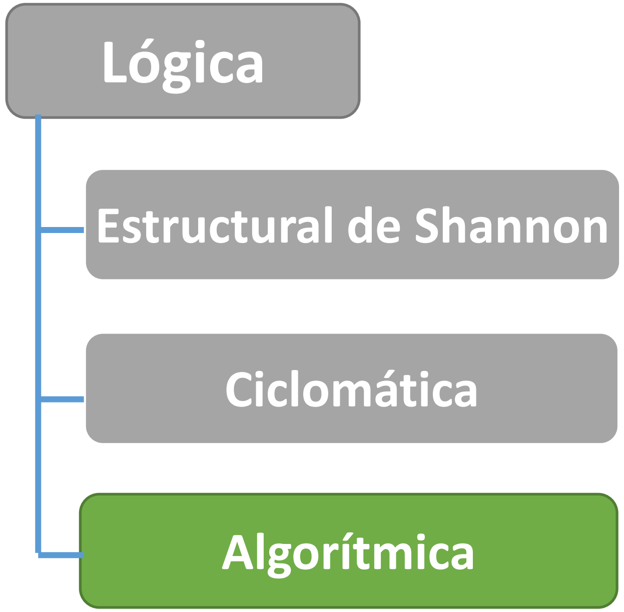
En Ingeniería de Software, esta medida de complejidad se denomina hoy **complejidad ciclomática**.

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

COMPLEJIDAD DE LAS SOLUCIONES

Antes de ver la **complejidad computacional** en máquinas de Turing, comentaremos sobre algunos métodos para establecer una **complejidad lógica**: la complicación intrínseca de la solución.

FUERA DE APUNTE



Complejidad Algorítmica: Otra propuesta para medir la complejidad de un problema, es la de Gregory Chaitin que la propone como ***"el largo del programa más corto escrito para resolver el problema"***.

Esta medida tiene no pocos inconvenientes, ya que el largo del programa varía según el lenguaje y el compilador utilizado para construirlo.

Siguiendo las ideas del matemático ruso Kolmogorov, aunque Gregory lo negaría, esta visión también tiene mucho que ver con el concepto de aleatoriedad y genera aún controversias (ver el número Ω de Chaitin).

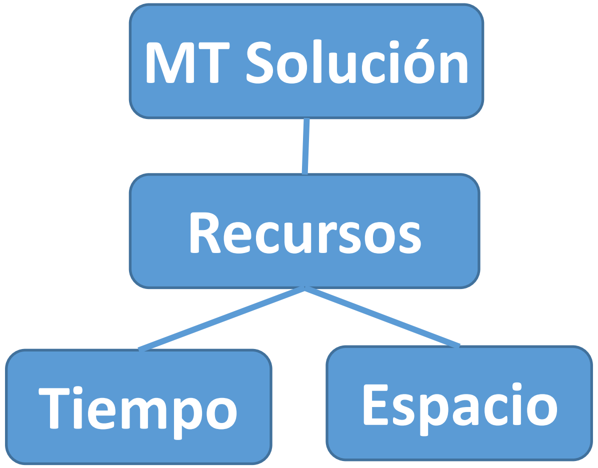
Complejidad

- Introducción
- Esquema general
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- **Complejidad Computacional**
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Trata-bles e Intratables
- Clases de problema
  - P, NP y NP-completos
- EJEMPLOS 1, 2, 3
  - Resumen

SINTAXIS Y SEMÁNTICA DE LOS LENGUAJES

COMPLEJIDAD COMPUTACIONAL

En las Ciencias de la Computación, el enfoque dado a la complejidad es la **Teoría de la Complejidad Computacional**, que identifica a la complejidad con **los recursos** (**tiempo** y **espacio**) **requeridos** por la solución (operativa).



Ya se indicó el uso de la **Máquina de Turing** como un modelo formal y conocido de computación en el cual desarrollar las soluciones a los problemas.

También se señaló que lo que se busca es **cómo cambian** los recursos utilizados para la ejecución de la **MT** según el largo de los datos de entrada: **una función**.

Por ejemplo, un algoritmo para ***"ordenar un arreglo de diez números de menor a mayor"***, tardará más tiempo y ocupará más espacio al querer operar con un arreglo de **cien mil números**. **Pero el algoritmo es el mismo, por lo cual su complejidad no debería cambiar !!!**

COMPLEJIDAD COMPUTACIONAL

Complejidad Temporal: La **complejidad temporal** de una MT es una función **T(n)** que determina la cantidad movimientos (o unidades elementales de tiempo) que utiliza en resolver un problema con datos de entrada α de largo $|\alpha|=n$.

Complejidad Espacial: La **complejidad espacial** de una MT es la función **E(n)** que determina la cantidad de celdas de cinta (su memoria) utilizadas para resolver un problema con datos de entrada α de largo $|\alpha|=n$.

Si bien estas dos medidas son independientes, como el cabezal de una MT siempre se mueve de a una celda, en general:

$$E(n) \leq T(n) + 1$$

COMPLEJIDAD COMPUTACIONAL

Órdenes de Complejidad: Dado un problema computable, se pueden construir varias **MT** distintas que lo resuelvan; para ellas, habrá distintas complejidades **T(n)** y **E(n)**, pero salvo errores, en general serán todas ellas de similar **orden**.

El **orden** de una función ***f***, se denota con la notación **O grande**:

$$f(n) \in O(g(n)) \leftrightarrow \exists n_0 \in N,\ k \in R \ /\ \forall n \geq n_0 : f(n) \leq k \,.\, g(n)$$

esto es, la función ***f*** **es de orden *g*** si está acotada superiormente por ***k.g*** (**k** constante) a partir de un valor $\mathbf{n_0}$ en adelante.

De la definición resulta $\frac{f(n)}{g(n)} \leq k$, lo que dice que ambas funciones **crecen con la misma tasa** o velocidad, desde algún $\mathbf{n_0}$

COMPLEJIDAD COMPUTACIONAL

Órdenes de Complejidad (continuación)

Podemos ganar aún más en **objetividad**. Si bien un problema puede tener varias **MT** que lo resuelvan con sus **T(n)** y **E(n)**, al tener ellas similar orden, podremos hablar del **orden de la complejidad** de la solución:

$O(1)$:	Constante
$O(\log n)$:	Logarítmica
$O(n)$:	Lineal
$O(n.\log n)$:	n.log n
$O(n^2)$:	Cuadrática
$O(n^3)$:	Cúbica
…	
$k>1, O(n^k)$:	Polinómica
$k>1, O(k^n)$:	Exponencial
$O(n!)$:	Factorial

Y aún podemos ganar más **objetividad** y **generalidad** al pensar que cualquier solución con complejidad **POLINÓMICA** o inferior se comportará al funcionar en forma **eficiente**.

Problemas Tratables

Más allá de polinómicas tendremos:

Problemas Intratables

…

COMPLEJIDAD COMPUTACIONAL

Se estima que el número de partículas en el universo es $2{,}2 \times 10^{79}$

Órdenes de Complejidad (continuación)

$\lvert\alpha\rvert$=n→	1	10	100	1000
O(1)	1	1	1	1
$O(\log_2 n)$	0	3,32	6,64	9,97
O(n)	1	10	100	1.000
$O(n^2)$	1	100	10.000	1.000.000
$O(n^3)$	1	1.000	1.000.000	1.000.000.000
…	…	…	…	…
$O(2^n)$	2	1.024	$1{,}27 \times 10^{30}$	$1{,}07 \times 10^{301}$
O(n!)	1	3.628.800	$9{,}33 \times 10^{157}$	$4{,}02 \times 10^{2567}$

Complejidad

- Introducción
- Esquema general
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Trata-bles e Intratables
- **Clase de problema**
  - **P, NP y NP-completos**
- EJEMPLOS 1, 2, 3
  - Resumen

COMPLEJIDAD COMPUTACIONAL

Clase de Problemas: En 1971, Stephen Cook de la Universidad de Toronto, propuso una clasificación de los problemas en **clases de complejidad** que generaliza las ideas anteriores:

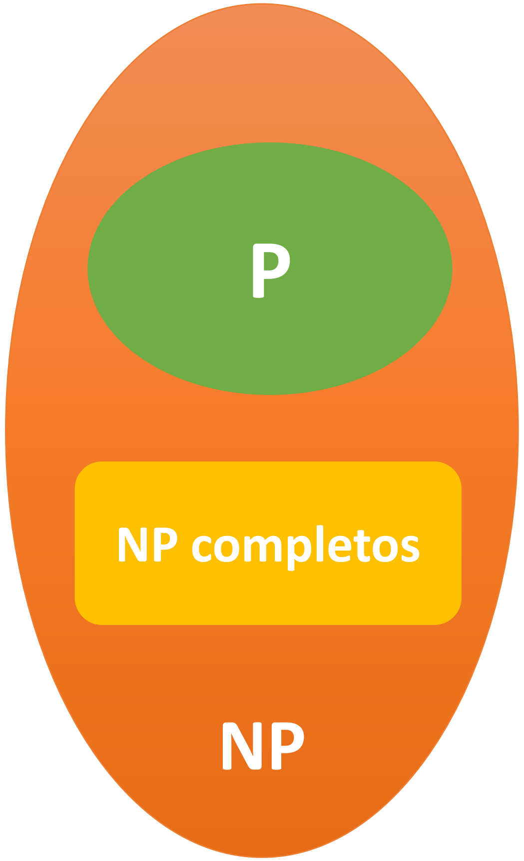
CLASE P: problemas que pueden resolverse con una **máquina de Turing Determinista** en **tiempo polinómico** o menor.

CLASE NP: problemas que pueden resolverse con una **máquina de Turing NO Determinista** en **tiempo polinómico** o menor.

CLASE NP-completos: subproblemas de la clase **NP** a los que cualquier otro problema de **NP** puede **reducirse** (conversión) en tiempo polinómico.

Pueden consultarse las fichas 20, 21 y 29 de la asignatura AED.

Clase de Problemas (continuación)



- Como la **MTD** es un caso particular de la **MTND**, es claro que **P** $\subseteq$ **NP**.
- Por definición, también **NP-completos** $\subseteq$ **NP**. El primer problema NP-completo, formulado por Cook, es el de satisfacibilidad lógica: **SAT**
- Uno de los problemas más importantes, **aún no resuelto**, de las matemáticas y la computación es el determinar si también **NP** $\subseteq$ **P**, o sea :

¿ P = NP ?

- Esto está en **investigación** actualmente y se han definido **muchas otras clases de complejidad**.

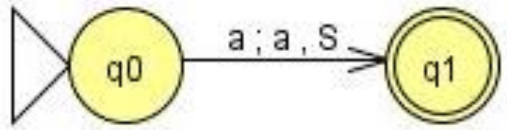
COMPLEJIDAD COMPUTACIONAL

Ejemplo 1: Construir una MT que reconozca cadenas de {a, b}* que inicien con la letra “**a**”.

Estrategia de resolución: Consideremos como ejemplo: **ƀƀabbbabaƀƀ**

Al leer el primer símbolo de la cadena de entrada, la MT ya puede saber si debe aceptar o rechazar la cadena !!! Entonces:

1) Lee el símbolo de entrada y si es “**a**” pasa a un estado de aceptación y **Para**.



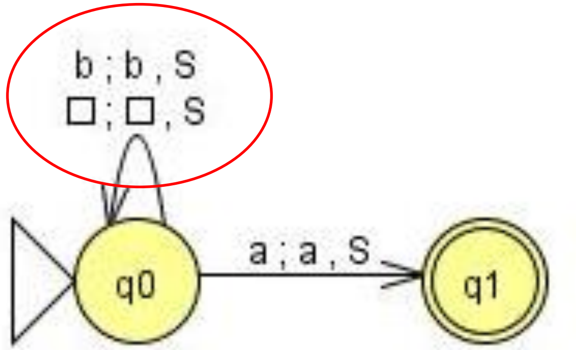
Sencillo !!!

Pero, ¿y si no la acepta? También lo hace en un solo movimiento. Como se definió la máquina, no tiene transición en **q0** con “**b**” o **blanco**, por lo cual si es ésta la situación la MT **cancela**, lo que dice que ha rechazado.

Podría completarse la función con las entradas que llevan a cancelar, pero en general no se hace para dejar claro sólo el camino a aceptación.

COMPLEJIDAD COMPUTACIONAL

Ejemplo 1: (continuación)



MT completada con las transiciones que no llevan a la aceptación.

MT1 = ({a, b}, {a, b, ƀ}, {q0, q1}, q0, {q1}, *f*, ƀ}

La **complejidad temporal** no depende del largo **|α|=n** de la cadena de entrada, y en un solo movimiento acepta o rechaza cualquier α∈**{a, b}***.

$$T(n) = 1 \Rightarrow T(n) \text{ es } O(1)\text{: constante}$$

La **complejidad espacial** responde al largo de la cadena que debe poder ser almacenada en la cinta:

$$E(n) = n \Rightarrow E(n) \text{ es } O(n)\text{: lineal}$$

La **complejidad estructural** de Shannon es: $|Q| \times |\Gamma| = 2 \times 3 = 6$

Complejidad

- Introducción
- Esquema general
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Tratables e Intratables
- Clases de problema
  - P, NP y NP-completos
- **EJEMPLOS** 1, **2**, 3
  - Resumen

COMPLEJIDAD COMPUTACIONAL

Ejemplo 2: Construir una MT que procese cadenas de {a, b}* agregando un **asterisco** al final de ella por cada letra "**a**" que contenga.

Estrategia de resolución: Consideremos como ejemplo: **ƀƀabbabaƀƀ**

1) Lee el símbolo de entrada y: **ƀƀAbbaba∗ƀ**

   1.1) Si es "**a**" lo marca con "**A**" y avanza hasta el primer blanco a la derecha, colocando en ese lugar "∗". Luego vuelve a la izquierda hasta encontrar la marca "**A**", la cambia por "**a**" y mueve el cabezal a la derecha. **ƀƀabbaba∗ƀ**

   1.2) Si es "**b**", la deja como está y mueve el cabezal a la derecha.

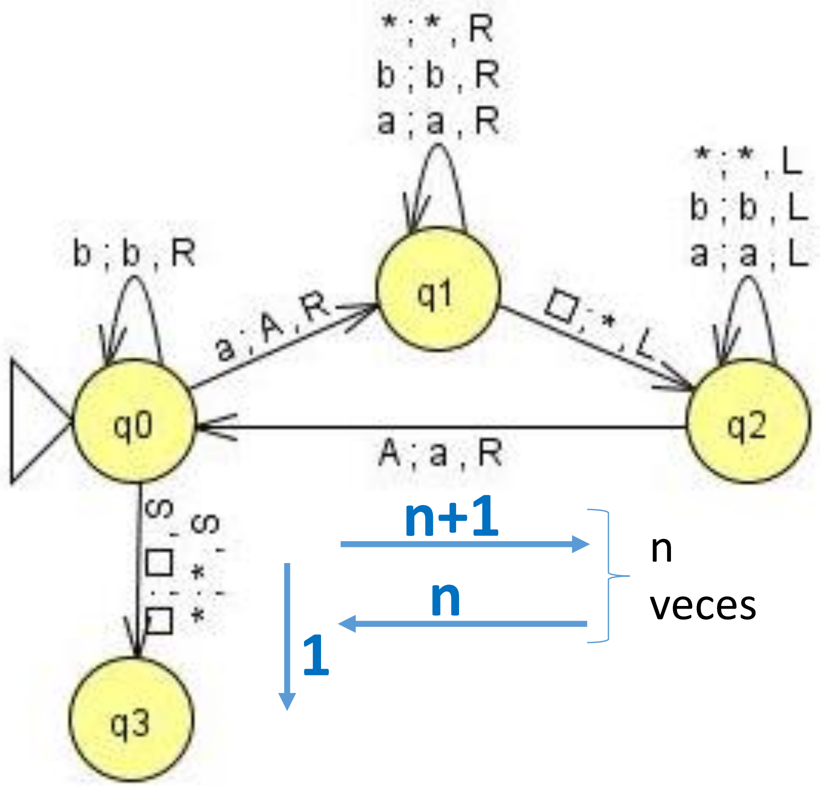
2) Repite el paso **1** hasta que encuentre un **blanco** o un **asterisco**, y en ese caso **Para**. **ƀƀabbaba∗∗∗ƀ**

Mejor caso: Si solo hay "**b**" recorre toda la cadena y **Para**.

Peor caso..: Si solo hay "**a**" la recorrerá **|entrada|** veces de ida y vuelta.

COMPLEJIDAD COMPUTACIONAL

Ejemplo 2: MT2 = ({a, b}, {a, b, A, ∗, ƀ}, {q0, q1, q2, q3}, q0, {}, *f*, ƀ}



La **complejidad temporal** se obtiene para el **peor caso** (todas letras **a**), contando los movimientos realizados en función del largo **|α|=n** de la entrada.

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

$$T(n) = 2n^2 + n + 1 \Rightarrow T(n) \text{ es } O(n^2)\text{: cuadrática}$$

Menos Sencillo !!!

La **complejidad espacial** será el doble de la entrada por los asteriscos agregados

$$E(n) = 2n \Rightarrow E(n) \text{ es } O(n)\text{: lineal}$$

La **complejidad estructural** de Shannon es: $|Q| \times |\Gamma| = 4 \times 5 = 20$

COMPLEJIDAD COMPUTACIONAL

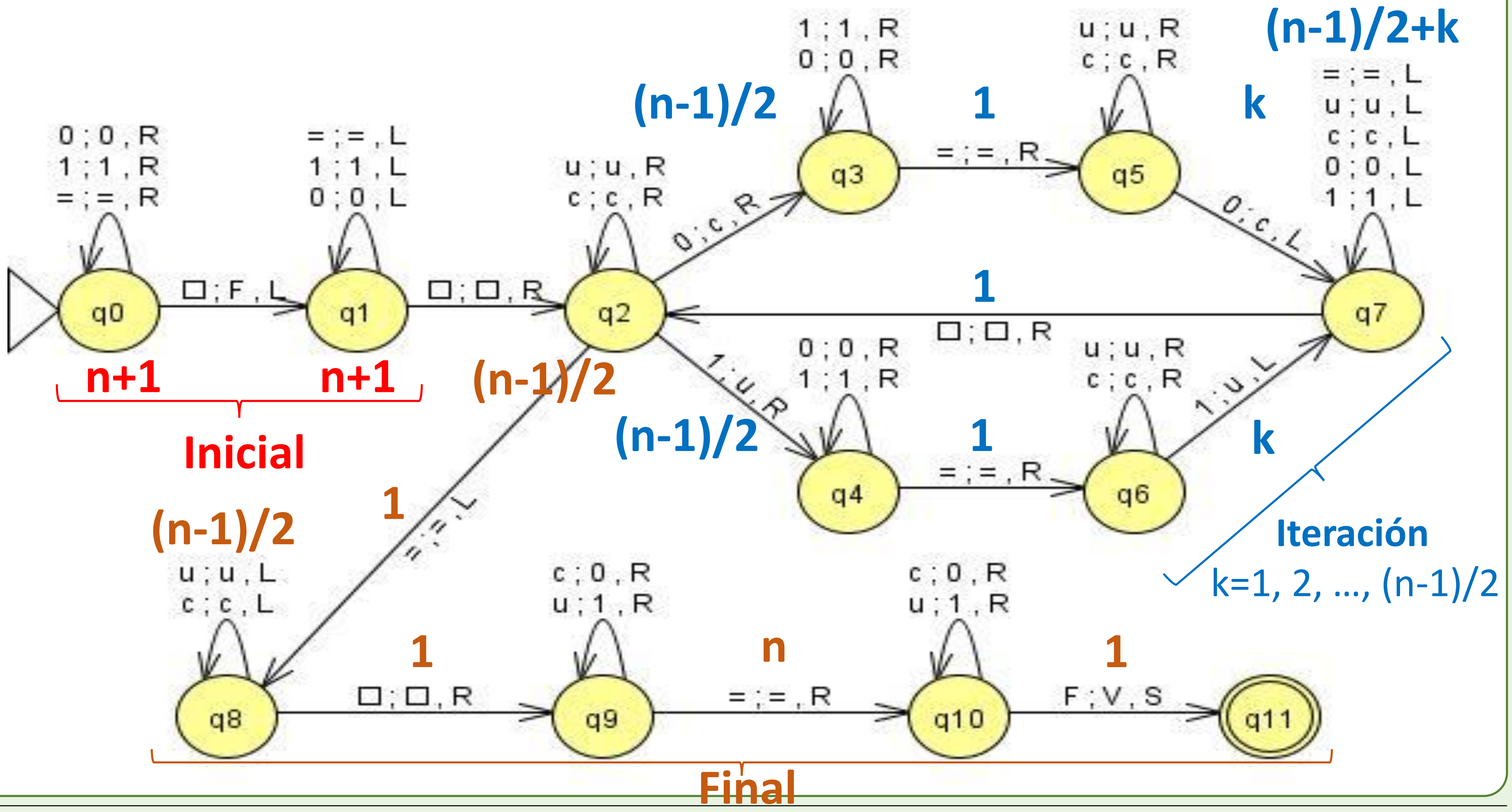
Ejemplo 3: Construir una MT que reconozca cadenas de largo al menos tres compuestas por dos cadenas binarias separadas por el símbolo "**=**", si y sólo si, las dos cadenas son iguales. Responde: **V** o **F** al final derecho.

Estrategia de resolución: Consideremos como ejemplo: **bb101=101bb**

1) Recorrer la cadena y poner **F** al final.	**bb101=101Fb**
2) Volver al principio.	**bb101=101Fb**
3) Si lee **0** marcar con **c** y si lee **1** marcar con **u**, y DER.	**bbu01=101Fb**
4) Buscar el primer dígito luego del **=** y comparar.	**bbu01=101Fb**
5) Si lee el mismo símbolo marcar e IZQ.	**bbu01=u01Fb**
6) Volver hasta el blanco de la izquierda y DER.	**bbu01=u01Fb**
7) Repetir 3, 4, 5 y 6, salteando las marcas.	**bbucu=ucuFb**
8) Si al repetir el paso 3 se encuentra **=**, casi terminó.	**bbucu=ucuFb**
9) Volver al principio, cambiar las marcas y al final	**bb101=101Fb**
10) Reemplazar la **F** por **V** y **Parar**.	**bb101=101Vb**

COMPLEJIDAD COMPUTACIONAL

Ejemplo 3: **MT3 = ({0, 1, =}, {0, 1, =, u, c, V, F, ƀ}, {q0, …, q11}, q0, {q11}, *f*, ƀ}**



Complejidad
- Introducción
- Esquema general
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Tratables e Intratables
- Clases de problema
  - P, NP y NP-completos
- **EJEMPLOS** 1, 2, **3**
  - Resumen

COMPLEJIDAD COMPUTACIONAL

Ejemplo 3: La **complejidad temporal** es cuántos movimientos realiza:

T(n) =	**(n+1) + (n+1) +**	**Inicial**
	((n-1)/2 + 1 + 1 + ((n-1)/2 + 1 + 1) +	**1ª iteración**
	((n-1)/2 + 1 + 2 + ((n-1)/2 + 2 + 1) +	**2ª iteración**
	((n-1)/2 + 1 + 3 + ((n-1)/2 + 3 + 1) +	**3ª iteración**
	…	**…**
	((n-1)/2 + 1 + (n-1)/2 + ((n-1)/2 + (n-1)/2 + 1) +	**(n-1)/2 iteración**
	((n-1)/2 + 1 + (n-1)/2 + 1 + n + 1)	**Final**

Pero 2.(1+2+3+…+(n-1)/2) = 2.[((n-1)/2).((n-1)/2+1)]/2 según fórmula de Gauss.

= ((n-1)/2) . ((n-1)/2+1)

T(n) =	**(n+1) + (n+1) +**	**Inicial**
	((n-1)/2) . ((n-1)/2+1+((n-1)/2+1+(n-1)/2+1)) +	**Todas las iteraciones**
	((n-1)/2 + 1 + (n-1)/2 + 1 + n + 1)	**Final**

Complejidad

- Introducción
- Esquema general
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Tratables e Intratables
- Clases de problema
  - P, NP y NP-completos
- **EJEMPLOS** 1, 2, **3**
  - Resumen

COMPLEJIDAD COMPUTACIONAL

Ejemplo 3: (continuación)

T(n)	= (n+1) + (n+1) +	Inicial
	((n-1)/2) . ((n-1)/2+1+((n-1)/2+1+(n-1)/2+1)) +	Todas las iteraciones
	((n-1)/2 + 1 + (n-1)/2 + 1 + n + 1)	Final
T(n)	= 2n + 2 + ((n-1)/2) . (n+2+(n-1)/2) + 2n + 2	Total
T(n)	= 2n + 2 + ((n-1)/2) . (2n/2+4/2+(n-1)/2) + 2n + 2	Total
T(n)	= 2n + 2 + (n-1)/2) . ((3n+3)/2) + 2n + 2	Total
T(n)	= 2n + 2 + (n-1) . (3n+3)/4 + 2n + 2	Total
T(n)	= $4n + 4 + 3n^2/4 + 3n/4 – 3n/4 – 3/4$	Total
T(n)	= $\frac{3}{4} n^2 + 4n + 13/4$	Total Final

$$T(n) = \frac{3}{4}n^2 + 4n + \frac{13}{4}$$ Total Final

No tan Sencillo !!!

Complejidad

- Introducción
- Esquema general
- Concepto
- Comp. Lógica
  - Estructural
  - Ciclomática
  - Algorítmica
- Complejidad Computacional
  - Comp. Temporal y Espacial
  - Notación O grande
  - Órdenes de complejidad
  - Problemas Tratables e Intratables
- Clases de problema
  - P, NP y NP-completos
- **EJEMPLOS** 1, 2, **3**
  - Resumen

COMPLEJIDAD COMPUTACIONAL

Ejemplo 3: (continuación)

Luego de todo ese trabajo de deducción, la cantidad de movimientos necesarios para aceptar una cadena de largo **n**, esto es la **complejidad temporal** es:

$$T(n) = \frac{3}{4}n^2 + 4n + \frac{13}{4} \Rightarrow \text{T(n) es O(n}^2\text{): cuadrático}$$

La **complejidad espacial** es la cantidad de celdas de cinta utilizadas; en este caso será el largo de la cadena de entrada **n** más el blanco de la izquierda leído y, finalmente, más la posición de la respuesta **V** o **F**:

$$E(n) = n + 2 \Rightarrow \text{E(n) es O(n): lineal}$$

La **complejidad estructural** de Shannon es:

$$|Q| \times |\Gamma| = 12 \times 8 = 96$$

COMPLEJIDAD COMPUTACIONAL

Ejemplo 3: (continuación)

La **dificultad** para hallar la complejidad temporal de este tercer ejemplo, nos lleva a pensar **otros métodos** para determinarla.

Otra forma de determinar la complejidad temporal de una MT es utilizar un **simulador**. Para este ejemplo se cargó la solución en JFLAP y se corrió con **distintos largos de cadena**, generando la siguiente tabla de valores:

Largo	Movimientos
3	22
5	42
7	68
9	100

Valores de JFLAP

Suponiendo la solución de tiempo cúbico:

$$T(n) = A \cdot n^3 + B \cdot n^2 + C \cdot n + D$$

se puede armar el sistema de ecuaciones:

$$A \cdot 3^3 + B \cdot 3^2 + C \cdot 3 + D = 22$$
$$A \cdot 5^3 + B \cdot 5^2 + C \cdot 5 + D = 42$$
$$A \cdot 7^3 + B \cdot 7^2 + C \cdot 7 + D = 68$$
$$A \cdot 9^3 + B \cdot 9^2 + C \cdot 9 + D = 100$$

COMPLEJIDAD COMPUTACIONAL

Ejemplo 3: (continuación)

$$A \cdot 3^3 + B \cdot 3^2 + C \cdot 3 + D = 22$$
$$A \cdot 5^3 + B \cdot 5^2 + C \cdot 5 + D = 42$$
$$A \cdot 7^3 + B \cdot 7^2 + C \cdot 7 + D = 68$$
$$A \cdot 9^3 + B \cdot 9^2 + C \cdot 9 + D = 100$$

Al resolver este sistema de 4 ecuaciones con 4 incógnitas se obtiene:

$$A=0,\ \ B=3/4,\ \ C=4,\ \ D=13/4$$

El hecho de obtener **A=0**, indica que el **polinomio no es cúbico** como se supuso sino **cuadrático**, lo que coincide nuestro anterior análisis.

Siempre se debe utilizar un polinomio de **mayor grado al que se intuya que es una solución**, para asegurarse.

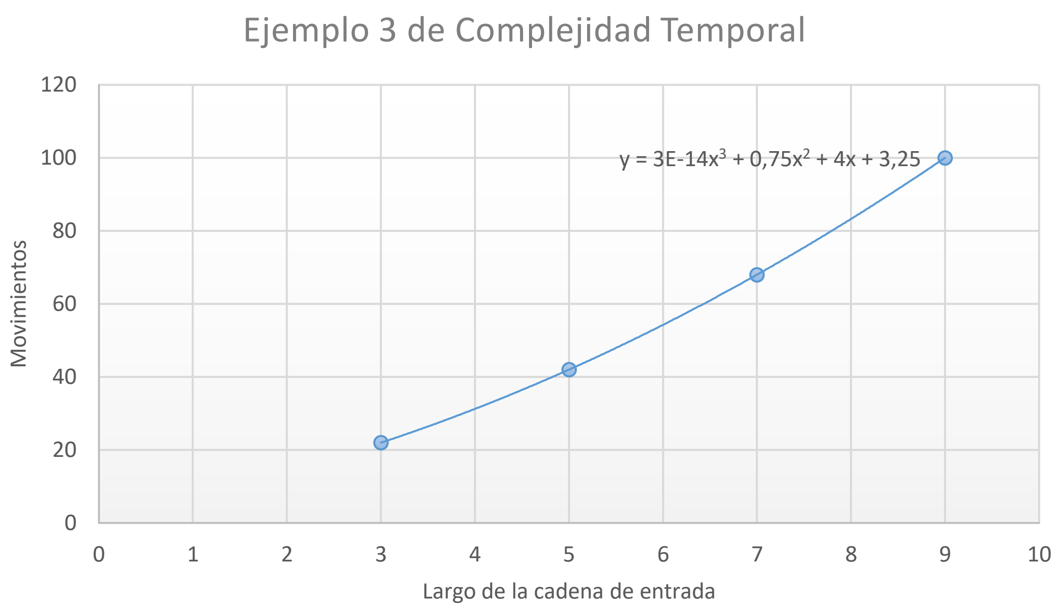
Por supuesto si la solución no es polinómica, este método solo dará una **aproximación local**.

COMPLEJIDAD COMPUTACIONAL

Ejemplo 3: (continuación)

Otra forma de determinar la complejidad temporal de una MT es utilizar una **planilla de cálculo** como Excel para trabajar los datos arrojados por el **simulador**. Entonces se solicita que grafique los valores como puntos y luego que los **aproxime por una curva polinómica de tercer grado**.

Largo	Movimientos
3	22
5	42
7	68
9	100
Valores de JFLAP	



COMPLEJIDAD COMPUTACIONAL

Resumen de Ejemplos:

#	Temporal		Espacial		Estructural
1	$T(n) = 1$	O(1)	E(n) = n	O(n)	6
2	$T(n) = 2n^2 + n + 1$	O(n^2)	E(n) = 2n	O(n)	20
3	$T(n) = \frac{3}{4}n^2 + 4n + \frac{13}{4}$	O(n^2)	E(n) = n + 2	O(n)	96

Como puede verse, el tercer ejemplo necesitó más trabajo de diseño y de cálculo que el segundo, aunque su **complejidad temporal** y **espacial son del mismo orden**. Sin embargo la **complejidad estructural** muestra al menos una diferente **complicación**.