APELLIDOS, NOMBRE____ NOTA________

Ejercicio_1. (2 puntos)

El algoritmo de ordenación MergeSort puede ser implementado por:

```
MergeSort (A, p, r) /* Ordena un vector A desde p hasta r */
        if p<r {/* Dividir en dos trozos de tamaño igual (o lo más parecido posible), es decir ⌈n/2⌉ y ⌊n/2⌋ */
            q = ⌊(p+r)/2⌋ ;        /* Divide */
             /* Resolver recursivamente los subproblemas */
            MergeSort (A,p,q);          /* Resuelve */
            MergeSort (A,q+1,r) ;    /* Resuelve */
            /* Combinar: mezcla dos listas ordenadas en O(n) */
            Merge (A,p,q,r);      /* Combina*/
            }
```

El algoritmo utiliza para su implementación la función Merge que tiene un coste **Θ(n)**.

Se pide:

a. (0,75 puntos). Calcular la complejidad del algoritmo propuesto por el método de la **ecuación característica**.
b. (0,5 puntos). Calcular la complejidad del algoritmo propuesto por el Teorema maestro.
c. (0,75 puntos). Comparar el algoritmo propuesto con el siguiente (Calcular la complejidad y compararlas):

```
MergeSort_bad (A, p, r) /* Ordena un vector A desde p hasta r */
    if p<r {
        MergeSort_bad (A,p,p);
        MergeSort_bad (A,p+1,r) ;
        Merge (A,p,p,r);
        }
```

NOTA: El Teorema maestro es:

$$T(n) \in \begin{cases} O(n^{\log_b a}) & \text{si } a > b^k \\ O(n^k \cdot \log^{p+1} n) & \text{si } a = b^k \\ O(n^k \cdot \log^p n) & \text{si } a < b^k \end{cases}$$

➢ Solución:

a. Ecuación característica:

$$T(n) = \begin{cases} c_1 & \text{si } n \le 1; \\ 2T(n/2) + c_2 n + c_3 & \text{si } n > 1 \end{cases}$$

- Si $n=2^k$ podemos escribir la ecuación como: $T(2^k) = 2T(2^{k-1}) + 2^k c_2 + c_3$
- Haciendo el cambio de variable $t_k = T(2^k)$, tenemos $t_k = 2\, t_{k-1} + 2^k c_2 + c_3$
  [$p(x)=(x-2)(x-2)(x-1) \Rightarrow r_1=r_2=2$ doble; $r_3=1$]
- Esta es una recurrencia no homogénea, con solución $t_k = c_1 2^k + c_2 k 2^k + c_3 1^k$
- Deshacemos el cambio de variable: $T(n) = c_1 n + c_2 n \log n + c_3 \in \Theta(n \cdot \log n)$

b. Aplicando el teorema maestro $T(n) = aT(n/b) + \Theta(n^k \log^p n)$; a=b=2, k=1, p=0 $\Rightarrow a = b^k$

$\boldsymbol{T(n)} \in O(n^k \cdot \log^{p+1} n) \Rightarrow \boldsymbol{T(n) \in \Theta(n \cdot \log n)}$

c. Ecuación característica:

$$T(n) = \begin{cases} c_1 & \text{si } n \le 1; \\ T(n-1) + c_2 n + c_3 & \text{si } n > 1 \end{cases}$$

- Recurrencia NO homogénea $p(x)=(x-1)(x-1)^2 \Rightarrow r=1$ triple]
- Solución $T(n) = c_1 1^n + c_2 n 1^n + c_3 n^2 1^n \in \Theta(n^2)$

$\Rightarrow \Theta(n \cdot \log n) \subseteq \Theta(n^2)$ y la versión primera es más eficiente.

Ejercicio_2. (3 puntos)

Para aprobar una asignatura el/la estudiante tiene que hacer en total **n** tareas (exámenes, prácticas, trabajos, etc). Para cada una de ellas estima que le llevará cierto tiempo, t_i. Como puede realizarlas a lo largo del curso, las quiere repartir entre las convocatorias de junio, septiembre y diciembre. En cada convocatoria puede sacar **M** unidades de tiempo como **máximo**. Se supone que todas las tareas se deben hacer y que no se pueden fraccionar (cada tarea va a una sola convocatoria). El **objetivo** es conseguir un reparto haciendo las tareas cuanto antes, no dejarlas para el final, es decir, minimizar el tiempo dedicado en la convocatoria de diciembre. En caso de empate en diciembre, minimizar el tiempo en la de septiembre.

a. (1 punto). Diseñar un algoritmo **voraz** para resolver el problema, aunque no se garantice siempre la solución óptima. Proponer y contrastar dos criterios de selección.
   Aplicar el algoritmo al caso: **n** = 5, **M** = 16, **t** = {7, 5, 3, 5, 6}.
b. (1 punto). Resolver el problema mediante **programación dinámica**. Definir la ecuación recurrente, los casos base, las tablas y el algoritmo para rellenarlas. No hay que aplicar el ejemplo.
c. (1 punto). Resolver el problema por **backtracking** usando el esquema iterativo. Indicar cómo es la representación de la solución, la forma del árbol y programar las funciones genéricas del esquema correspondiente.

➢ Solución:

a. Diseñar un **algoritmo voraz** para resolver el problema. Proponer y contrastar dos criterios de selección. Aplicar el algoritmo al caso: **n** = 5, **M** = 13, **t** = {6, 4, 2, 4, 5}.

❖ Estructuras de datos y variables para representar la información:

- **Entrada**: Tiempos de las tareas a realizar, los **candidatos** son las tareas. Una vez seleccionadas en cierto orden, se introducen en la primera convocatoria en la que quepan (junio, septiembre o diciembre). Para decidir el orden de selección hay que proponer y probar dos criterios:
  - seleccionar las tareas de mayor a menor (para ejecutar las más largas antes) y,
  - seleccionar las tareas de menor a mayor (para ejecutar las más posibles antes).
- **Salida**: La **solución** se almacenará en un array **s: array [1..n] de entero**, **s[i]** es la convocatoria en la que se hace la tarea i (1= junio, 2= septiembre, 3= diciembre).
- **Variables auxiliares**:
  - **ord: array [1..n] de entero**, almacenará los tiempos de las tareas ordenados según criterio.
  - **conv: array [1..3] de entero**, almacenará el tiempo acumulado de cada convocatoria.

❖ Algoritmo.

```
TareasVoraz (t: array [1..n] de entero; var s: array [1..n] de entero)
  conv:= (0, 0, 0)
  Ordenar t, almacenando los índices en ord  /* En orden creciente o decreciente */
  para i:= 1, …, n hacer
      si conv[1]+t[ord[i]] ≤ M entonces s[ord[i]]:= 1
      sino
          si conv[2]+t[ord[i]] ≤ M entonces s[ord[i]]:= 2
          sino
              si conv[3]+t[ord[i]] ≤ M entonces s[ord[i]]:= 3
              sino devolver "No se puede encontrar solución"
              fsi
          fsi
      fsi
      conv[s[ord[i]]]:= conv[s[ord[i]]] + t[ord[i]]
  finpara
fTareasVoraz
```

❖ Caso: **n** = 5, **M** = 13, **t** = {6, 4, 2, 4, 5}. Ninguno de los dos garantiza la solución óptima. Aplicando los dos criterios al ejemplo:

- De mayor a menor (6, 5, 4, 4, 2): ord= {1, 5, 2, 4, 3} ⟹ s = [1, 2, 1, 2, 1]; conv = [13, 8, 0]
- De menor a mayor (2, 4, 4, 5, 6): ord= {3, 2, 4, 5, 1} ⟹ s = [2, 1, 1, 1, 2]; conv = [10,11, 0]

b. Resolver el problema mediante **programación dinámica**. No hay que aplicar el ejemplo.

Las **decisiones** son del tipo "dada una tarea **i**, ejecutarla en junio, en septiembre o en diciembre".

- Si la ejecutamos en junio, nos quedan **i**-1 tareas por decidir, y $\mathbf{t_i}$ unidades de tiempo menos en junio. Lo mismo para septiembre y diciembre. La ecuación: **Tareas (i, j, s, d)**, resolverá el problema con las **i** primeras tareas y con **j**, **s** y **d** unidades de tiempo disponibles en junio, septiembre y diciembre, respectivamente.
- Para la **función objetivo** hay que minimizar el tiempo asignado a diciembre, pero en caso de empate hay que considerar el de septiembre. Si (j, s, d) son los tiempos asignados en cada convocatoria, la función objetivo será: **Valor**(j, s, d) = d·M + s.

a. Ecuación recurrente:

Tareas(i, j, s, d) = min {Tareas(i-1, j-t_i, s, d), t_i + Tareas(i-1, j, s-t_i, d), t_i·M + Tareas(i-1, j, s, d-t_i)}

b. Casos base:

Tareas(0, j, s, d) = 0 ; **Tareas(i, j, s, d)** = +∞ si j<0 ó s<0 ó d<0

c. Tabla: **T**:

array[0..n, 0..M, 0..M, 0..M] de entero, siendo T[i, j, s, d] = Tareas(i, j, s, d). El valor final de la función objetivo es T[n, M, M, M].

d. Algoritmo:

```
TareasPD
 para i:= 0 hasta n hacer
   para j:= 0 hasta M hacer
     para s:= 0 hasta M hacer
       para d:= 0 hasta M hacer
         si i==0 entonces T[i, j, s, d]:= 0
         sino
 T[i,j,s,d]:=min(T[i-1,j-t[i],s,d],t[i]+T[i-1,j,s-t[i],d],t[i]*M+T[i-1,j,s,d-t[i]])
        fsi
       fpara
     fpara
   fpara
 fpara
 escribir ("Tiempo en diciembre:", T[n, M, M, M] div M)
 escribir ("Tiempo en septiembre:", T[n, M, M, M] mod M)
fTareasPD
```

c. Resolver el problema por **backtracking** usando el esquema iterativo. Indicar cómo es la representación de la solución, la forma del árbol y programar las funciones genéricas del esquema correspondiente.

a. Representación de la **solución** mediante una tupla **s** = (s_1, s_2, ..., s_n), siendo cada s_i = 1, 2 ó 3, según la tarea **i** se asigne a la convocatoria de junio, septiembre o diciembre, respectivamente.
La forma del árbol de soluciones es 3-ario. Inicialización: **s** = (0,0,.., 0).

b. Variable auxiliar: **conv**: array [0..3] de entero, controla el tiempo asignado a cada convocatoria. La función objetivo será la misma que la considerada en el algoritmo de programación dinámica.

c. Funciones genéricas del esquema iterativo:

operación Inicializar
voa:= +∞
nivel:= 1
s:= (0, 0, ..., 0)
conv:= (0, 0, 0, 0)

operación Generar (nivel, s)
conv[s[nivel]]:= conv[s[nivel]] - t[nivel]
s[nivel]:= s[nivel] + 1
conv[s[nivel]]:= conv[s[nivel]] + t[nivel]

operación Solución (nivel, s)
devolver (nivel==n) AND (conv[s[nivel]]<=M)

operación Criterio (nivel, s)
devolver (nivel<n) AND (conv[s[nivel]]<=M)

operación MasHermanos (nivel, s)
devolver s[nivel] < 3

operación Retroceder (nivel, s)
conv[s[nivel]]:= conv[s[nivel]] – t[nivel]
s[nivel]:= 0
nivel:= nivel - 1

operación Valor (s)
devolver conv[3]*M + conv[2]

Ejercicio_3. (1,5 puntos)

Dado el autómata siguiente,

f	a	b
$\rightarrow$ 0	1	3
1	0	3
2	1	4
* 3	5	5
4	3	3
* 5	5	3

Obtener:

a. (0.5 puntos). El A.F.D. mínimo equivalente.
b. (0.5 puntos). La expresión regular del lenguaje reconocido por el autómata del apartado anterior.
c. (0.5 puntos). La gramática de tipo 3 para el lenguaje.

➢ **Solución:**

a. El algoritmo de minimización obtiene:

Conjunto inicial Q/E_1 = ({0,1, 2,4},{3,5})

Q/E_2 =({0,1}, {2},{4},{3,5})

Q/E_3= Q/E_2

Los estados 2 y 4 son inaccesibles desde el estado inicial $\Rightarrow$ los eliminamos.

Nombrando los estados como A={0,1} y B={3,5}, el A.F.D. mínimo equivalente es:

f	a	b
$\rightarrow$ A	A	B
* B	B	B

b. A partir del AFD mínimo del apartado anterior obtenemos las ecuaciones características:

1. $x_0=ax_0+ bx_1+b$
2. $x_1= ax_1+ bx_1+a+b$

Resolvemos aplicando la regla de inferencia $\boldsymbol{X=\alpha X+\beta \Leftrightarrow X=\alpha^*\beta}$

1. comenzando por la ecuación (2) : $x_1= (a+b)x_1+a+b=(a+b)^*(a+b)$
2. Sustituyendo en (1): $\mathbf{x_0}= ax_0+b(a+b)^*(a+b)+b=a^*(b(a+b)^*(a+b))+b)= a^*b((a+b)^*(a+b))+\lambda)= a^*b(a+b)^*$ (aplicando la propiedad $\alpha^*=\lambda+ \alpha\alpha^*$)

$\mathbf{x_0}$ = Lenguaje reconocido que es $a^*b(a|b)^*$ (cadenas que contienen al menos una *b*).

c. La gramática de tipo 3 asociada al A.F.D. mínimo del apartado a. es:

$A \rightarrow aA \mid bB \mid b$
$B \rightarrow aB \mid bB \mid a \mid b$

Ejercicio_4. (1,5 puntos)

Dado el lenguaje **$(01)^n$** con **n≥0**,

1) (0,75 puntos). Seleccionar, **justificando la respuesta**. el autómata que reconoce el lenguaje indicado.

   a. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(A,0)=B, f(A, λ)= λ, f(C,0)=B, f(B,1)=C, f(B,1)= λ

   b. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(A,0)=B, f(A, λ)=F, f(C,0)=B, f(B,1)=C, f(B,1)=F

   c. AF=[{0,1}, {A,B,C,F}, f, A,{F}] con f(A, B)=0, f(A,F)= λ, f(C,B)=0, f(B,C)=1, f(B,F)=1

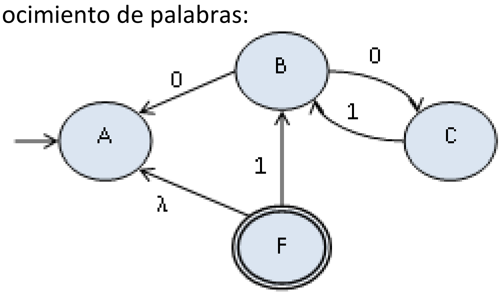
   d. AF=[{0,1}, {A,B,C,F}, f, A, {F}] con f(B,0)=A, f(F, λ)=A, f(B,0)=C, f(C,1)=B, f(F,1)=B

2) (0,75 puntos). Obtener el AFD mínimo equivalente del autómata seleccionado en el apartado anterior.

➢ **Solución:**

1) Seleccionar el autómata que reconoce el lenguaje **$(01)^n$** con **n≥0**.

- La opción "a" no es válida por las transiciones f(A, λ)= λ, f(B,1)=λ, pues la salida de las transiciones no puede ser λ, siempre tiene que ser un estado incluido en Q={A,B,C,F}.
- La opción "c" no es correcta porque todas las transiciones tienen como símbolo de entrada un estado (B, C o F) en vez de un elemento del alfabeto (0 ó 1), y como estado de llegada un símbolo del alfabeto (0 ó 1) en lugar de un estado (A, B, C o F).
- La opción "d" no es válida porque hay varias transiciones que llegan al estado inicial A, pero no hay ninguna que salga de él para empezar el reconocimiento de palabras:

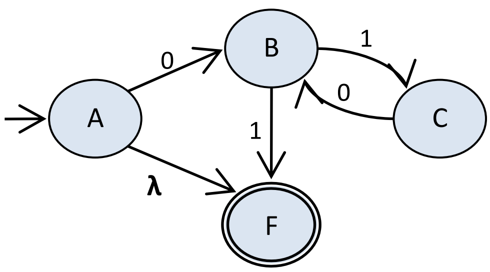


- La opción correcta es la "b", es la única sobre la que se puede comprobar que genera el lenguaje indicado en el enunciado: $(01)^n$ con n≥0. Se verifica que lo cumple porque reconoce λ (para n=0), por la transición f(A, λ)=F; y el resto de transiciones sólo permiten llegar al estado final cuando se ha leído uno o varios pares de "01" consecutivamente.

2) Para obtener el AFD mínimo equivalente del AFND seleccionado se convierte al AFD equivalente y luego al AFD mínimo equivalente:

   2.1. AFND ⇒ AFD equivalente

   Partiendo del AFND = [{0,1}, {A,B,C,F}, f, A, {F}] f(A,0)=B, f(A, λ)=F, f(C,0)=B, f(B,1)=C, f(B,1)=F



	0	1	λ
→ A	{B}		{F}
B		{C, F}	
C	{B}		
* F			

- Q_0 = *λ-clausura*(A) = {A,F}
- *λ-clausura*(f(Q_0={A,F},0))=*λ-clausura(B)={B}=Q_1*
- *λ-clausura*(f(Q_0={A,F},1))=∅=*Q_2*
- *λ-clausura*(f(Q_1={B},0))= ∅=*Q_2*
- *λ-clausura*(f(Q_1={B},1))=*λ-clausura(C,F)={C,F}=Q_3*
- *λ-clausura*(f(Q_2=∅, 0))= ∅=*Q_2*
- *λ-clausura*(f(Q_2=∅, 0))= ∅=*Q_2*
- *λ-clausura*(f(Q_3={C,F},0))= *λ-clausura(B)={B}=Q_1*
- *λ-clausura*(f(Q_3={C,F},1))= ∅=*Q_2*

	0	1
→ * Q_0 = {A,F}	Q_1	Q_2
Q_1 = {B}	Q_2	Q_3
Q_2= ∅	Q_2	Q_2
* Q_3= {C,F}	Q_1	Q_2

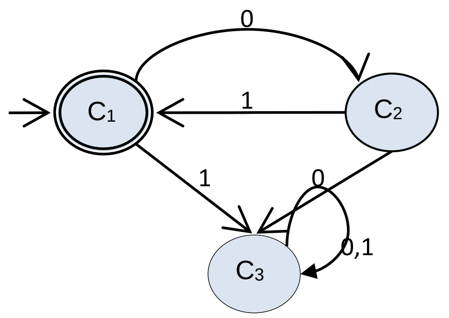
2.2. AFD ⇒ AFD mínimo equivalente:

Cálculo del conjunto cociente:

Q/E_0 = {{Q_0, Q_3},{Q_1,Q_2}}={C_1,C_2}

Q/E_1 = {{Q_0, Q_3},{Q_1},{Q_2}}={C_1,C_2,C_3}= Q/E_2 = Q/E

AFD_{min} = [{0,1}, {C_1,C_2,C_3}, f, C_1, {C_1}], donde f es:



	0	1
→ * C_1	C_2	C_3
C_2	C_3	C_1
C_3	C_3	C_3

Ejercicio_5. (2 puntos)

Dada la gramática:

$S \rightarrow S = A \mid A$

$A \rightarrow A := B \mid B$

$B \rightarrow (S) \mid a \mid b$

a. (0.5 puntos). Comprobar si es LL(1), eliminar la recursividad a la izquierda y obtener la gramática LL(1) equivalente.

b. (0.5 puntos). Convertir la gramática del apartado anterior en un autómata con pila que acepte el mismo lenguaje por pila vacía.

c. (0.5 puntos). Analizar, teniendo en cuenta el principio de preanálisis (lectura de un símbolo de la entrada con anticipación) la entrada "**(a)**".

d. (0.5 puntos). Implementar el seudocódigo de análisis descendente dirigido por la sintaxis para la gramática obtenida LL(1).

- **Solución:**

a. Comprobar si es LL(1).

- Eliminar la recursividad a la izquierda

$$\begin{array}{ll} 1.\ S \rightarrow S = A \\ 2.\ \quad\ \mid A \end{array} \left\{ \begin{array}{l} S \rightarrow A\ S' \\ S' \rightarrow =A\ S' \\ \quad\ \mid \lambda \end{array} \right.$$

$$\begin{array}{ll} 3.\ A \rightarrow A := B \\ 4.\ \quad\ \mid B \end{array} \left\{ \begin{array}{l} A \rightarrow B\ A' \\ A' \rightarrow\ := B\ A' \\ \quad\ \mid \lambda \end{array} \right.$$

5. $B \rightarrow (S)$
6. $\quad \mid a$
7. $\quad \mid b$

- Gramática equivalente:

1. $S \rightarrow A\ S'$
2. $S' \rightarrow = A\ S'$
3. $\quad \mid \lambda$
4. $A \rightarrow B\ A'$
5. $A' \rightarrow\ := B\ A'$
6. $\quad \mid \lambda$
7. $B \rightarrow (S)$
8. $\quad \mid a$
9. $\quad \mid b$

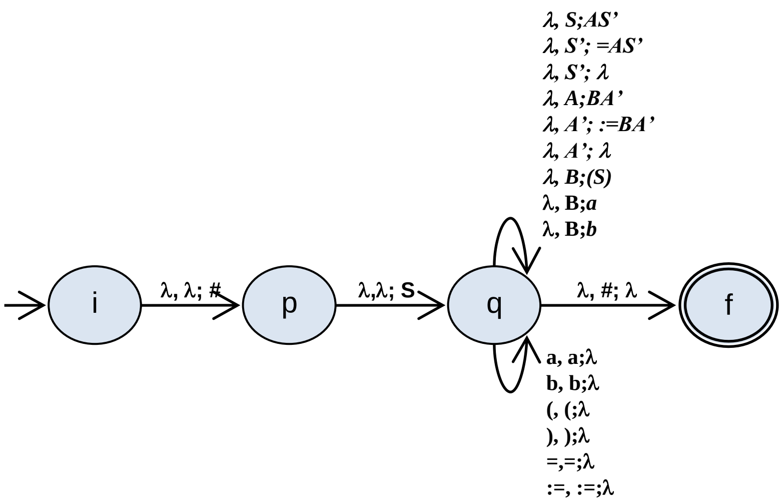
	PRIM	SIG
S	{(, a, b}	{\$,)}
S'	{=, **λ** }	{\$,)}
A	{(, a, b}	{=, \$,)}
A'	{**:=**, **λ** }	{=, \$,)}
B	{(, a, b}	{**:=**, =, \$,)}

- Comprobación de la Condición necesaria y suficiente LL(1):

Primero($= A\ S'$) $\cap$ Siguiente(S') = {=} $\cap$ {\$,)} = $\varnothing$
Primero($:= B\ A'$) $\cap$ Siguiente(A') = {:=} $\cap$ {=, \$,)} = $\varnothing$
Primero((S)) $\cap$ Primero(a) $\cap$ Primero(b) = {(} $\cap$ {a} $\cap$ {b} = $\varnothing$

Por tanto, la gramática equivalente es LL(1).

b. Convertir la gramática en un autómata con pila que acepte el mismo lenguaje por pila vacía.



c. Analizar, teniendo en cuenta el principio de preanálisis, la entrada "**(a)**".

estado	pila	entrada	acción	indeterminación	acción
i	**λ**	(a)$	(i, λ, λ ; p, #)		
p	#	(a)$	(p, λ, λ ; q, S)		
q	S#	(a)$	(q, λ, S ; q, AS')		$S \rightarrow A\,S'$
q	AS'#	(a)$	(q, λ, A ; q, BA')		$A \rightarrow B\,A'$
q	BA'S'#	(a)$	(q, λ, B ; q, (S))		$B \rightarrow (\,S\,)$
q	(S)A'S'#	(a)$	(q, (,(; q,λ)		': Reconocer(´(´);
q	S)A'S'#	a)$	(q, λ, S ; q, AS')		$S \rightarrow A\,S'$
q	AS')A'S'#	a)$	(q, λ, A ; q, BA')		$A \rightarrow B\,A'$
q	BA'S')A'S'#	a)$	(q, λ, B ; q, a)		$B \rightarrow$ a
q	aA'S')A'S'#	a)$	(q, a ,a ; q,λ)		Reconocer(´a´);
q	A'S')A'S'#	)$	(q, λ, A' ; q, λ)	(q, λ, A' ; q, :=BA') ☞ **(q, λ, A' ; q, λ)**	$A' \rightarrow \lambda$
q	S')A'S'#	)$	(q, λ, S' ; q, λ)	(q, λ, S' ; q, =AS') ☞ **(q, λ, S' ; q, λ)**	$S' \rightarrow \lambda$
q	)A'S'#	)$	(q,) ,) ; q,λ)		Reconocer(´)´);
q	A'S'#	$	(q, λ, A' ; q, λ)	(q, λ, A' ; q, :=BA') ☞ **(q, λ, A' ; q, λ)**	$A' \rightarrow \lambda$
q	S'#	$	(q, λ, S' ; q, λ)	(q, λ, S' ; q, =AS') ☞ **(q, λ, S' ; q, λ)**	$S' \rightarrow \lambda$
q	#	$	**(q, λ, #; f, λ)**		
f	**λ**	**λ**	**Aceptar**		

d. Seudocódigo del análisis descendente dirigido por la sintaxis para la gramática LL(1).

1. $S \rightarrow A\ S'$
2. $S' \rightarrow = A\ S'$
3. $\quad | \ \lambda$
4. $A \rightarrow B\ A'$
5. $A' \rightarrow\ := B\ A'$
6. $\quad | \ \lambda$
7. $B \rightarrow (S)$
8. $\quad | \ a$
9. $\quad | \ b$

	PRIM	SIG
S	{(, a, b}	{$,)}
S'	{=, **λ** }	{$,)}
A	{(, a, b}	{=, $,)}
A'	{**:=**, **λ** }	{=, $,)}
B	{(, a, b}	{**:=**, =, $,)}

```
Simbolo SLA; /* Simbolo leido = preanálisis*/
Programa_Principal;
{
  SLA = leer_símbolo();
  S();                          //S = axioma
  Si SLA != $ entonces          // $ = EOF
        Error();
}
```

```
Procedimiento Reconocer (Simbolo terminal) {
  Si (SLA == terminal)
        leer_símbolo();
  sino
        error_sintactico();
}
```

```
Procedimiento S() {
 /* S → A S' */
  A();
  S'():
}

Procedimiento S'() {
/* S' → = A S'  | λ  */
  CASE SLA OF
   '=':        Reconocer(´=´);
               A()
               S'();
   ´)´, ´$´ :  /* nada */;
  else:  error_sintactico();
  END;
}
```

```
Procedimiento A() {
 /* A → B A' */
  B();
  A'():
}

Procedimiento A'() {
/* A' →  := B A' | λ  */
  CASE SLA OF
   ':=':       Reconocer(´:=´);
               B()
               A'();
   ´=´, ´)´, ´$´ :  /* nada */;
  else:  error_sintactico();
  END;
}
```

```
Procedimiento B() {
/* B → (S) | a | b */
  CASE SLA OF
   '(':   Reconocer(´(´);
          S()
          Reconocer(´)´);
   ´a´ :  Reconocer(´a´);
   ´b´ :  Reconocer(´b´);
  else:  error_sintactico();
  END;
}
```