Programación dinámica

Definiciones

Aho, Ullman (Fundatios of Computer Science):

El término Programación Dinámica procede de una teoría de control desarrollada por R. E. Bellman en 1950. Es una técnica para el llenado de una tabla usada para sustituir la implementación recursiva directa.

Rawlins (compared to what?):

Sustituye un re-cálculo recursivo con una prosaica asignación destructiva.

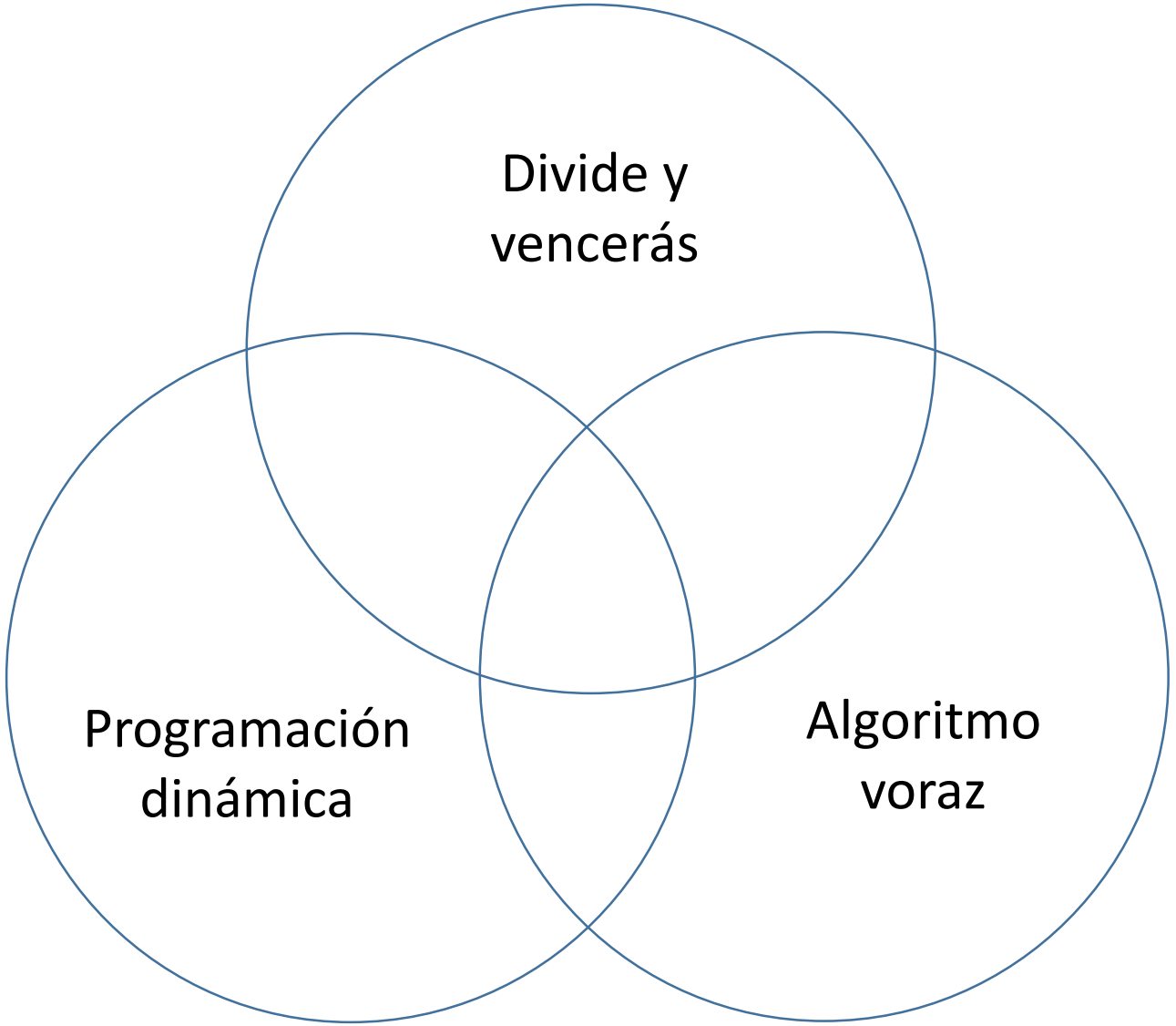
Hopcroft (The Design and Analysis of Computer Algorithms):

Esencialmente la Programación Dinámica calcula la solución de todos los sub-problemas y las guarda en una tabla.

Weiss (Data Structures and Algorithm Analysis):

Almacena en una lista todos los valores pre-calculados para evitar una llamada recursiva para algún sub-problema previamente resuelto.

Contexto



Elementos

Elementos de la Programación Dinámica

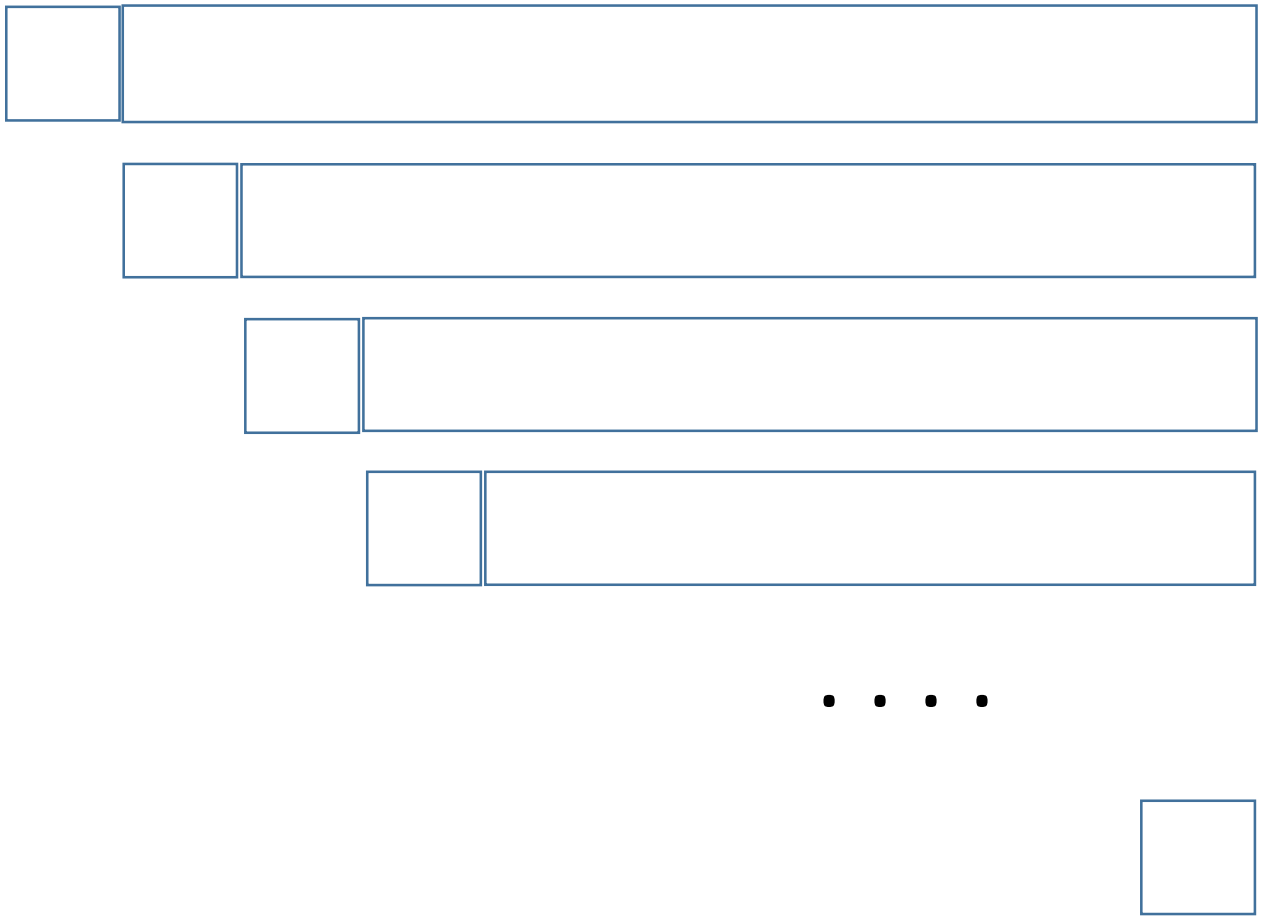
- Subestructura óptima
  Requiere programación ascendente (bottom-up)
  Busca la solución óptima para las particiones más pequeñas del problema.

- Traslape (Overlapping)
  Identifica las sub-tareas repetitivas y genera una solución única.

Ejemplo

Problema: Obtener el elemento mínimo de un vector.

Estrategia: Divide el vector en cabeza y resto.



Ejemplo

```
char menor(char * pal)
{
    if(*(pal + 1) == '\0')
       return(*pal);
    else
    if(*pal < menor(pal + 1))
       return(*pal);
    else
       return(menor(pal + 1));
}
```

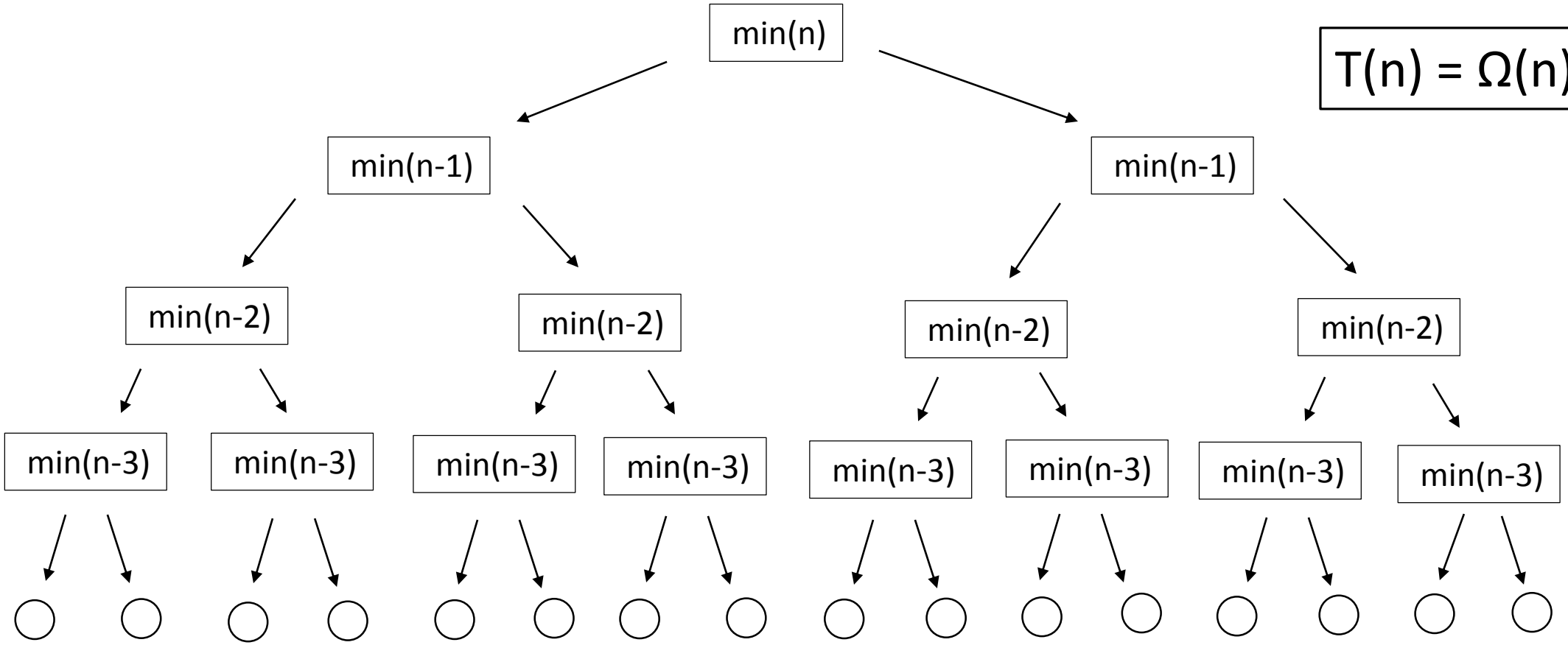
Ejemplo

```
char menor(char * pal)
{
    if(*(pal + 1) == '\0')
       return(*pal);
    else
    if(*pal < menor(pal + 1) )
       return(*pal);
    else
       return( menor(pal + 1) );
}
```

Árbol de llamadas recursivas

$T(n) = O(2^n)$

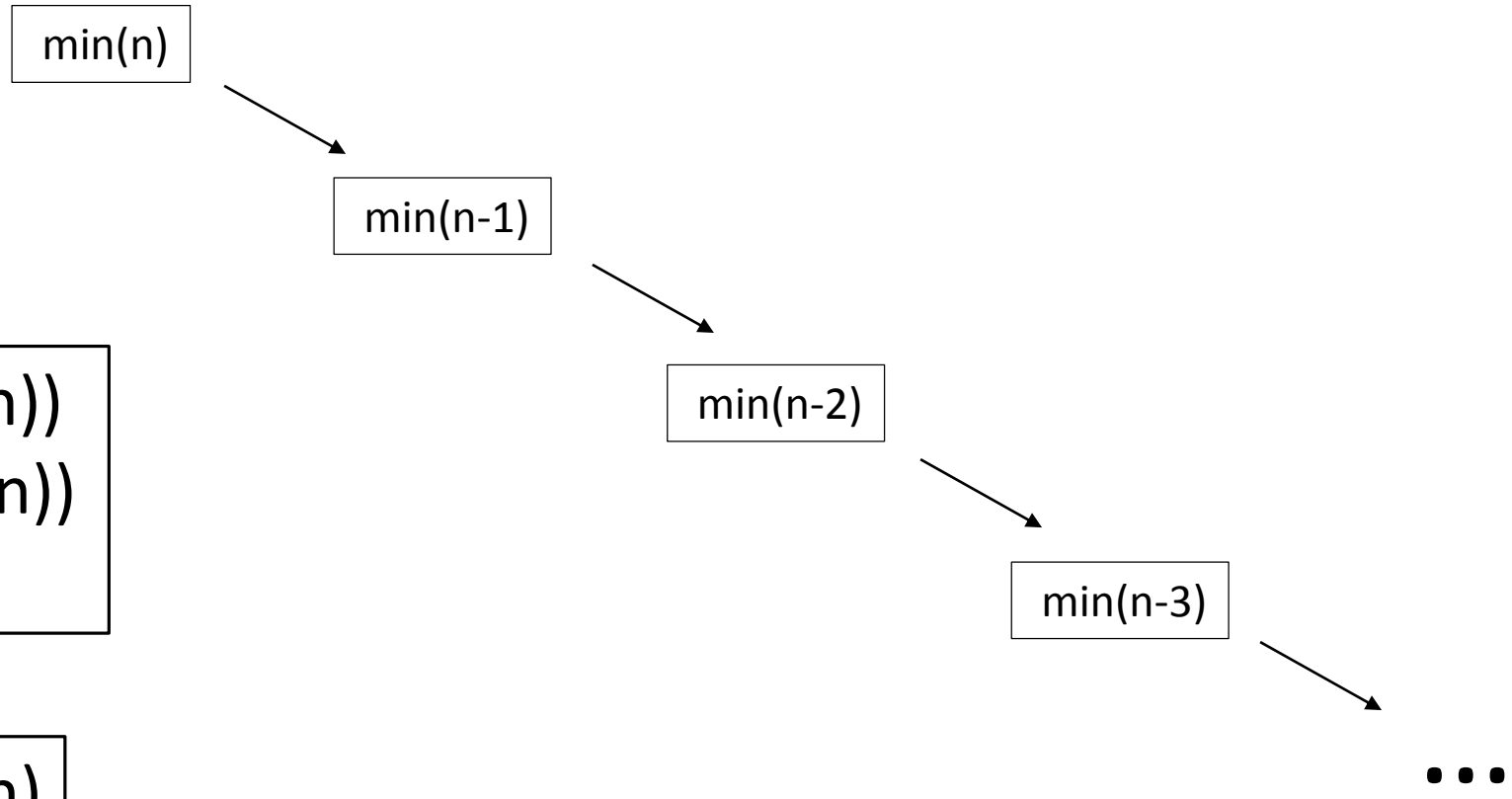
$T(n) = \Omega(n)$



Ejemplo

```
char menor(char * pal)
{
    char c;
    if(*(pal + 1) == '\0')
       return(*pal);
    else
    if(*pal < (c = menor(pal + 1)))
       return(*pal);
    else
       return(c);
}
```

Secuencia de llamadas recursivas

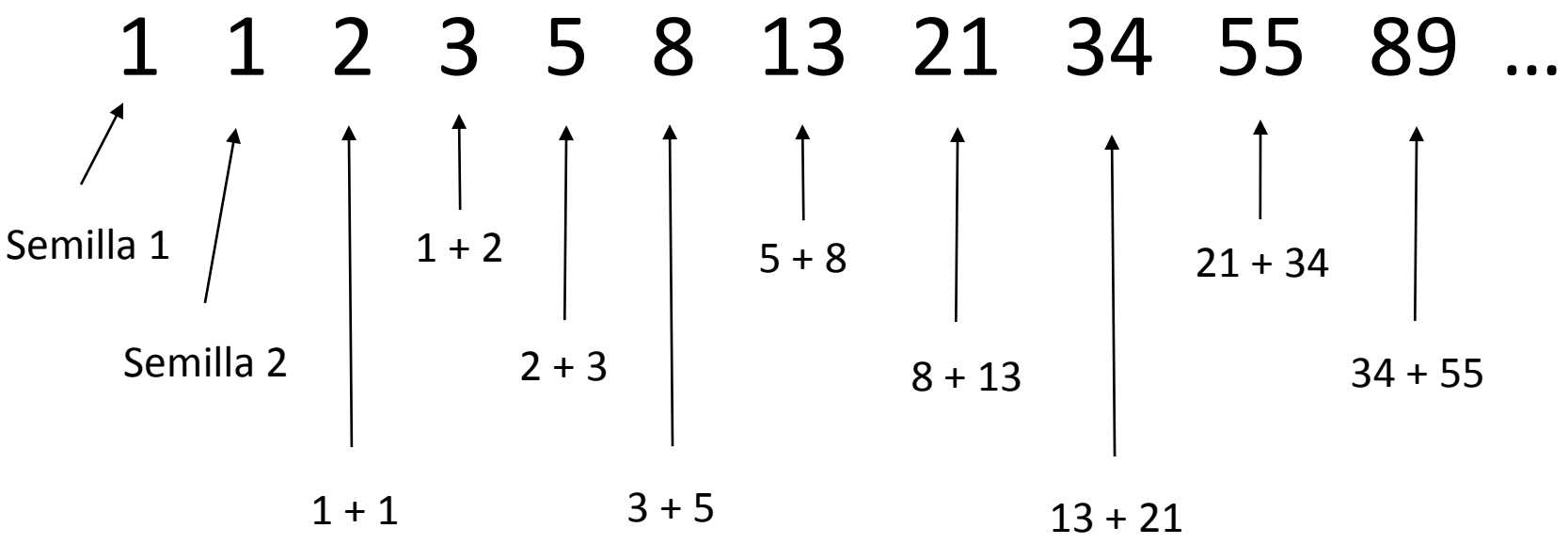


$T(n) = O(f(n))$
$T(n) = \Omega(g(n))$
$f(n) = g(n)$

$T(n) = \Theta(n)$

Aplicación

Secuencia de Fibonacci



Aplicación

```
int fibo(int i)
{
    if((i == 2) || (i == 1))
       return(1);
    else
       return( fibo(i - 2) + fibo(i - 1) );
}
```

Aplicación

```
int fibo(int i)
{
    if((i == 2) || (i == 1))
       return(1);
    else
       return( fibo(i - 2) + fibo(i - 1) );
}
```

Aplicación

```
int fibo(int i, int j[])
{
    int k = 0;
    if((i == 1) || (i == 2))
       return(1);
    else
    if(i == 3)
    {
       j[0] = 1;
       j[1] = 1;
    }
    else
    {
       k = fibo(i - 1, j);
       j[0] = j[1];
       j[1] = k;
    }
    return(j[0] + j[1]);
}
```

Conclusiones

El uso de la recursividad habilita la posibilidad de mantener la transparencia referencial en el trasvase de una definición matemática a través de un algoritmo y hasta la implementación.

En el contexto de la Programación Dinámica el uso de la asignación destructiva degrada la transparencia referencial, pero dicha pérdida es compensada con una implementación más eficiente en tiempo de ejecución.

Eficiencia

Transparencia Referencial