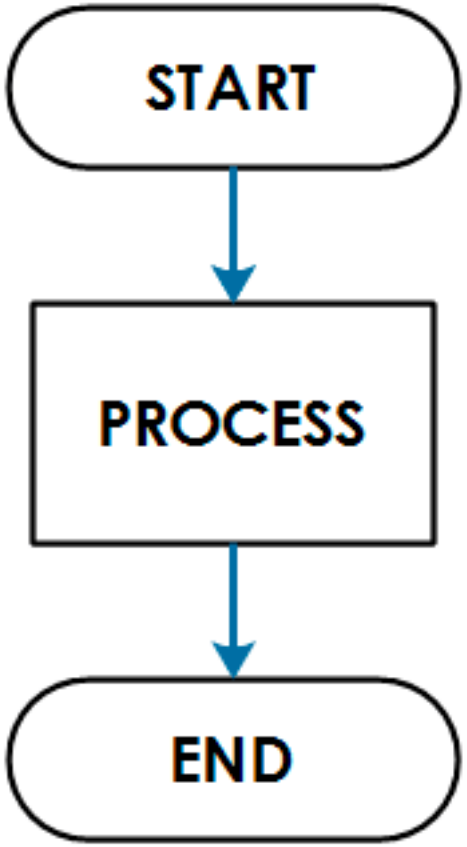
# Recursión y Backtracking

Guillermo Álvarez

# Agenda

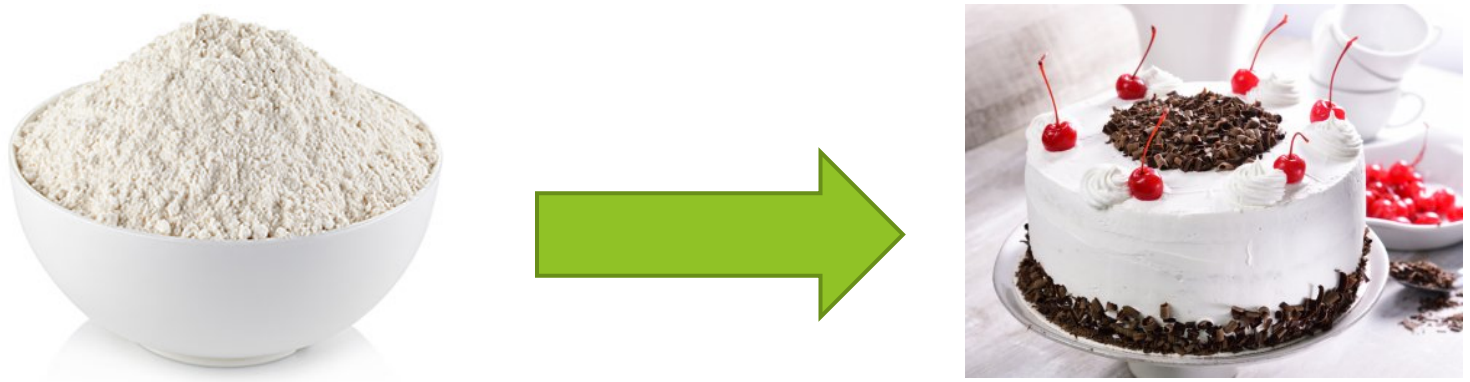
- ¿Qué es un Algoritmo?
- Recursión
- Reintento

# ¿Qué es un Algoritmo?



# ¿Qué es un Algoritmo?

- Es un procedimiento computacional bien definido que toma una entrada(conjunto de valores), los procesa y produce una salida.



- Se puede ver como una herramienta para resolver un problema computacional bien definido.

# ¿Porqué estudiar algoritmos?


# Fibonacci

$$F_n = \begin{cases} 0, & n = 0\,, \\ 1, & n = 1\,, \\ F_{n-1} + F_{n-2}, & n > 1\,. \end{cases}$$

Crecimiento de la función

$$F_n \geq 2^{n/2} \text{ for } n \geq 6\,.$$

# Fibonacci Idea Inicial

- Calcular el valor de Fn

- Entrada: Un entero n >= 0
- Salida: Fn

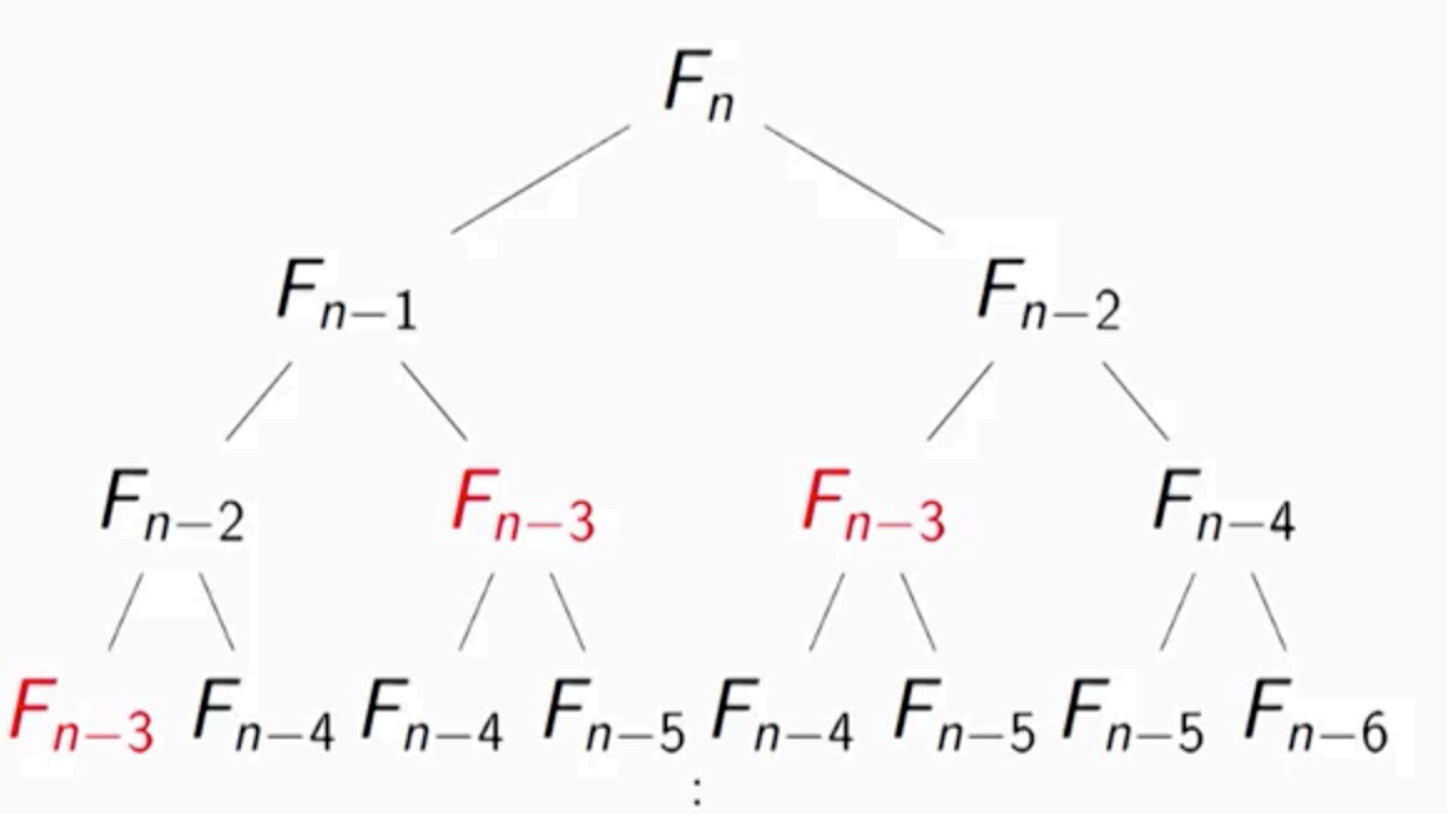
```
FibRecurs(n)
if n ≤ 1:
  return n
else:
  return FibRecurs(n − 1) + FibRecurs(n − 2)
```

# Tiempo de Ejecución

$$T(n) = \begin{cases} 2 & \text{if } n \leq 1 \\ T(n-1) + T(n-2) + 3 & \text{else.} \end{cases}$$

$$T(100) \approx 1.77 \cdot 10^{21}$$

# Árbol Recursivo



# Fibonacci Mejorado

```
FibList(n)

create an array F[0...n]
F[0] ← 0
F[1] ← 1
for i from 2 to n:
    F[i] ← F[i − 1] + F[i − 2]
return F[n]
```

# Recursión

# Recursión

- Una técnica que es muy usada en varios algoritmos es dividir la tarea en un número de piezas mas pequeñas, para resolver estas de forma individual y luego combinarlas las respuestas para responder la tarea original. Esto se denomina como el método de ’Dividir y Conquistar’.

- Cuando las subtareas son diferentes, esto lleva a diferentes subrutinas. Cuando estas son instancias del problema original, hablamos de algoritmos recursivos.

# Reducción

- La reducción es la técnica utilizada más común en el diseño de algoritmos. Reducir un problema X en otro problema Y significa escribir un algoritmo para X que usa un algoritmo para Y como una caja negra o subrutina.

- La correctitud del algoritmo que resulta no puede depender de ninguna manera en como el algoritmo para Y funciona. Lo único que podemos asumir es que la caja negra resuelve el problema Y de forma correcta.

# Simplificar y Delegar

- La recursión se puede ver como una forma especial de reducción, descrita de la siguiente forma:
  - Si la instancia del problema es pequeña o muy sencilla, resolver el problema directamente.
  - Si no, reduzca el problema a uno o mas instancias del mismo problema.

# Argumento Circular

- Deseamos entrar a una casa, pero la puerta esta cerrada y la llave dentro de la misma. Una solución podría ser.
  - *Si puedo entrar a la casa, podría obtener la llave, entonces podría abrir la puerta y podría entrar a la casa.*

- ¿Es una solución recursiva válida?

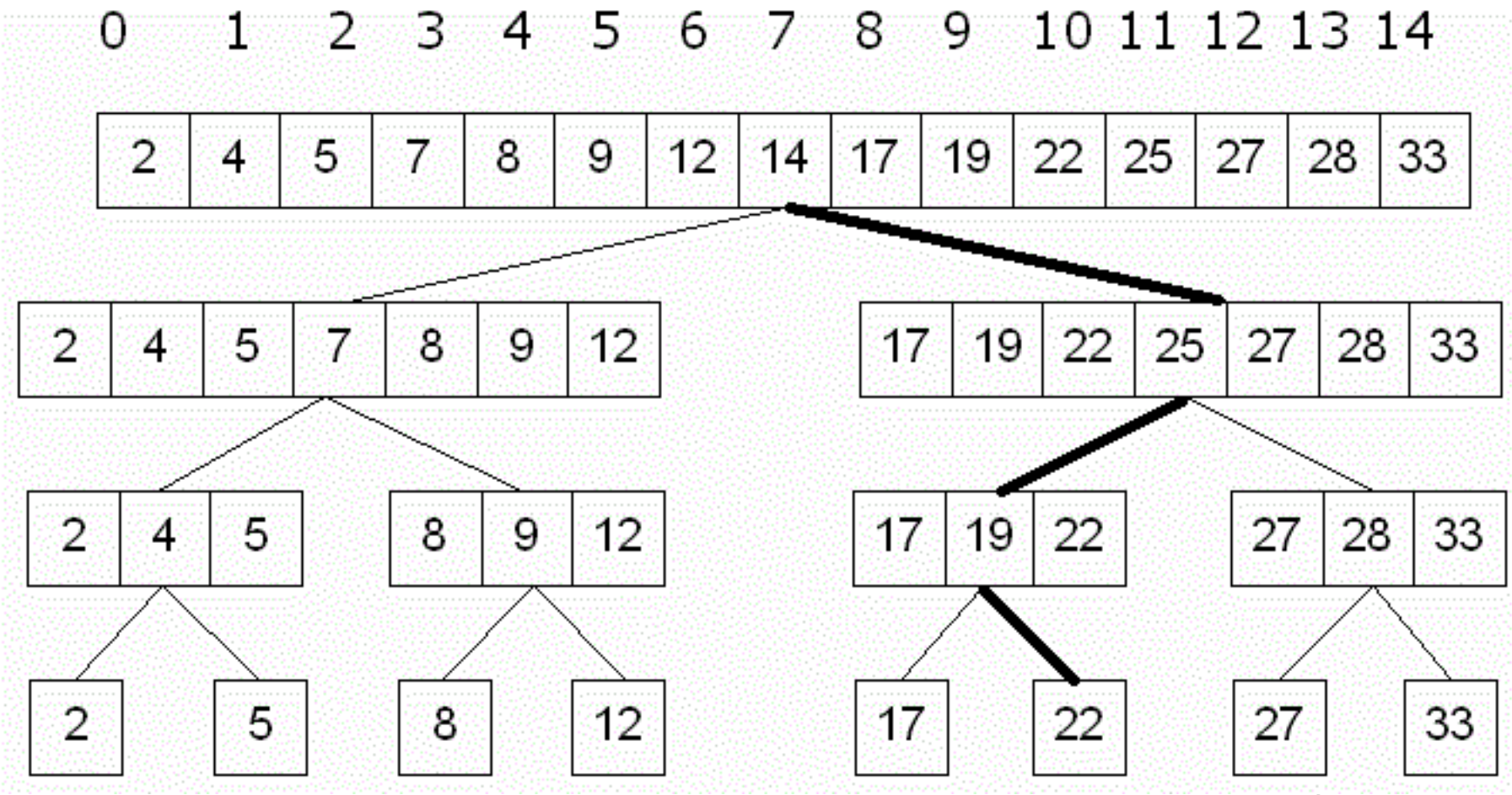
# Ejemplo 1:

- Considere una hilera de casas. El problema consiste en entrar dentro de una casa especificada previamente. Cada casa en la hilera es una instancia separada del mismo problema. Cada casa es mas grande que la siguiente.

- Su tarea es entrar a la casa más grande, adicionalmente la llave de la casa bloqueada esta en la casa del tamaño más pequeño cercana.

# Torres de Hanoi

# Búsqueda Binaria



# Búsqueda Binaria

- Entrada: Una arreglo A de n números enteros ordenados de manera ascendente y un valor k que se desea buscar.

- Salida: Un booleano indicando si el valor k está contenido en el arreglo A.

# Potenciación

¿Habrá otra forma de calcular la potenciación?

```
SlowPower(a, n):
    x ← a
    for i ← 2 to n
        x ← x · a
    return x
```

# Potenciación Rápida

- Se basa en la siguiente propiedad

$$a^n = a^{\lfloor n/2 \rfloor} \cdot a^{\lceil n/2 \rceil}.$$

$$a^{15} = a^7 \cdot a^7 \cdot a;\ a^7 = a^3 \cdot a^3 \cdot a;\ a^3 = a \cdot a \cdot a$$

# Potenciación Rápida

```
FastPower(a, n):
    if n = 1
        return a
    else
        x ← FastPower(a, ⌊n/2⌋)
        if n is even
            return x · x
        else
            return x · x · a
```

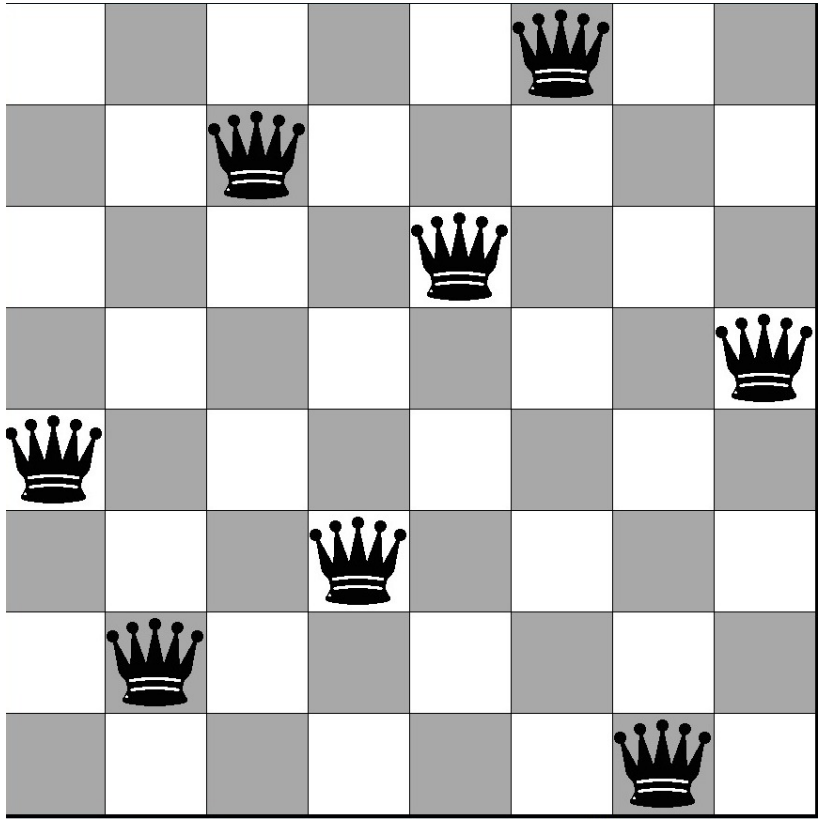
- Reintento

# Reintento

- También conocido como backtracking, es una estrategia de algoritmos recursivos que intenta construir una solución a un problema computacional de manera incremental.

- Cuando el algoritmo necesita decidir entre múltiples alternativas para el siguiente componente de la solución, sencillamente intenta con todas las posibles opciones de manera recursiva.

# Problema de las 8 reinas



# Problema de las 8 reinas

```
RecursiveNQueens(Q[1..n], r):
  if r = n + 1
      print Q
  else
      for j ← 1 to n
          legal ← True
          for i ← 1 to r − 1
              if (Q[i] = j) or (Q[i] = j + r − i) or (Q[i] = j − r + i)
                  legal ← False
          if legal
              Q[r] ← j
              RecursiveNQueens(Q[1..n], r + 1)
```

# Subset Sum

- Dado un conjunto X de enteros positivos y un entero T deseado, ¿Hay algún subconjunto de elementos en X que sumados me den el valor de T?

- X = {8, 6, 7, 5, 3, 10, 9} y T = {15}

# Subset Sum

- Casos Base:
  - T = 0
  - T < 0 OR T ≠ 0 pero el conjunto X es vacío

- Considere un elemento $x \in X$. Existe un subconjunto de X que suma T si y solo si una de las siguientes afirmaciones es cierta.

  - Hay un subconjunto de X que incluye a x y cuya suma es T.
  - Hay un subconjunto de X que excluye a x y cuya suma es T.

# Subset Sum

```
SubsetSum(X[1..n], T):
  if T = 0
      return True
  else if T < 0 or n = 0
      return False
  else
      return (SubsetSum(X[1..n−1], T) ∨ SubsetSum(X[1..n−1], T − X[n]))
```