# ¿Qué es un algoritmo?

# Definición:

Un algoritmo es un conjunto de instrucciones que resuelven un problema dado paso a paso y sin generar ambigüedades.

# Problema:
# Preparar una quesadilla

# Pasos a seguir:

1.- Conseguir el sartén

2.- Conseguir fuente de calor

A) ¿tenemos aceite?

I. Si sí, ponerlo en el sartén

Ii. Si no, ¿queremos comprar aceite?

1. Si sí, vamos y lo compramos
2. Si no, no preparamos omelette

3.- Prender el fuego y cocinar

# Curso Básico de Algoritmos y Estructuras de datos

# Lenguajes de programación

# Lenguajes máquina y ensamblador

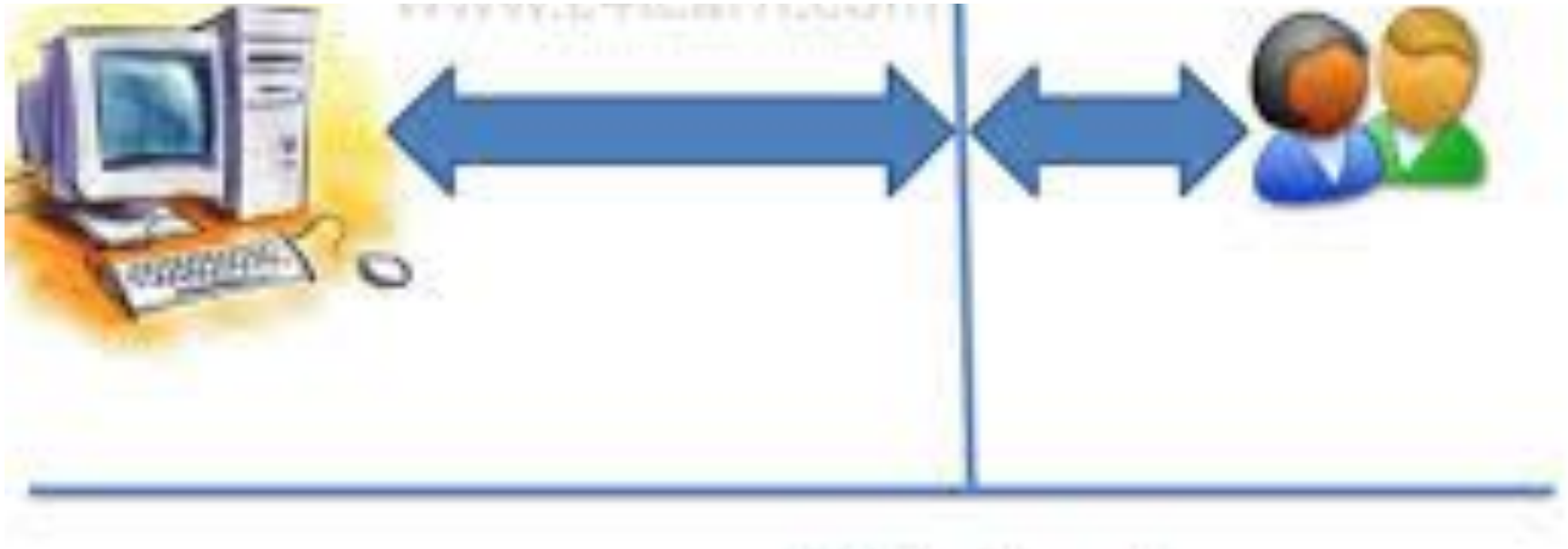
# 8086 INSTRUCTION SET

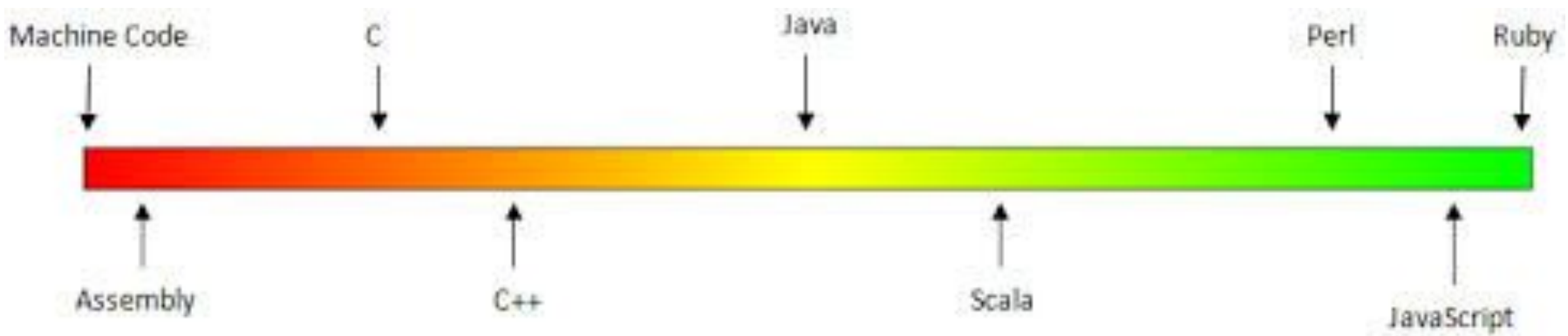
| OPCODE | | DESCRIPTION |
|---|---|---|
| AAA | | ASCII adjust addition |
| AAD | | ASCII adjust division |
| AAM | | ASCII adjust multiply |
| AAS | | ASCII adjust subtraction |
| ADC | dt,sc | Add with carry |
| ADD | dt,sc | Add |
| AND | dt,sc | Logical AND |
| CALL | proc | Call a procedure |
| CBW | | Convert byte to word |
| CLC | | Clear carry flag |
| CDL | | Clear direction flag |
| CLI | | Clear interrupt flag |
| CMC | | Complement carry flag |
| CMP | dt,sc | Compare |
| CMPS | [dt,sc] | Compare string |
| CMPSB | " | " bytes |
| CMPSW | " | " words |
| CWD | | Convert word to double word |
| DAA | | Decimal adjust addition |
| DAS | | Decimal adjust subtraction |
| DEC | dt | Decrement |
| DIV | sc | Unsigned divide |
| ESC | code,sc | Escape |
| HLT | | Halt |
| IDIV | sc | Integer divide |
| IMUL | sc | Integer multiply |
| IN | ac,port | Input from port |
| INC | dt | Increment |
| INT | type | Interrupt |
| INTO | | Interrupt if overflow |
| IRET | | Return from interrupt |
| JA | slabel | Jump if above |
| JAE | slabel | Jump if above or equal |
| JB | slabel | Jump if below |
| JBE | slabel | Jump if below or equal |
| JC | slabel | Jump if carry |
| JCXZ | slabel | Jump if CX is zero |
| JE | slabel | Jump if equal |
| JG | slabel | Jump if greater |
| JGE | slabel | Jump if greater or equal |
| JL | slabel | Jump if less |
| JLE | slabel | Jump if less or equal |
| JMP | label | Jump |
| JNA | slabel | Jump if not above |
| JNAE | slabel | Jump if not above or equal |
| JNB | slabel | Jump if not below |
| JNBE | slabel | Jump if below or equal |
| JNC | slabel | Jump if no carry |
| JNE | slabel | Jump if not equal |
| JNG | slabel | Jump if not greater |
| JNGE | slabel | Jump if not greater or equal |
| JNL | slabel | Jump if not less |
| JNLE | slabel | Jump if not less or equal |
| JNZ | slabel | Jump if not zero |
| JNO | slabel | Jump if not overflow |
| JNP | slabel | Jump if not parity |
| JNS | slabel | Jump if not sign |
| JO | slabel | Jump if overflow |
| JPO | slabel | Jump if parity odd |
| JP | slabel | Jump if parity |
| JPE | slabel | Jump if parity even |
| JS | slabel | Jump if sign |
| JZ | slabel | Jump if zero |
| LAHF | | Load AH from flags |
| LDS | dt,sc | Load pointer using DS |
| LEA | dt,sc | Load effective address |
| LES | dt,sc | Load pointer using ES |
| LOCK | | Lock bus |
| LODS | [sc] | Load string |
| LODSB | " | " bytes |
| LODSW | " | " words |
| LOOP | slabel | Loop |
| LOOPE | slabel | Loop if equal |
| LOOPZ | slabel | Loop if zero |
| LOOPNE | slabel | Loop if not equal |
| LOOPNZ | slabel | Loop if not zero |
| MOV | dt,sc | Move |
| MOVS | [dt,sc] | Move string |
| MOVSB | " | " bytes |
| MOVSW | " | " words |
| MUL | sc | Unsigned multiply |
| NEG | dt | Negate |
| NOP | | No operation |
| NOT | dt | Logical NOT |
| OR | dt,sc | Logical OR |
| OUT | port,ac | output to port |
| POP | dt | Pop word off stack |
| POPF | | Pop flags off stack |
| PUSH | sc | Push word onto stack |
| PUSHF | | Push flags onto stack |
| RCL | dt,cnt | Rotate left through carry |
| RCR | dt,cnt | Rotate right through carry |
| REP | | Repeat string operation |
| REPE | | Repeat while equal |
| REPZ | | Repeat while zero |
| REPNE | | Repeat while not equal |
| REPNZ | | Repeat while not zero |
| RET | [pop] | Return from procedure |
| ROL | dt,cnt | Rotate left |
| ROR | dt,cnt | Rotate right |
| SAHF | | Store AH into flags |
| SAL | dt,cnt | Shift arithmetic left |
| SHL | dt,cnt | Shift logical left |
| SAR | dt,cnt | Shift arithmetic right |
| SBB | dt,sc | Subtract with borrow |
| SCAS | [dt] | Scan string |
| SCASB | " | " byte |
| SCASW | " | " word |
| SHR | dt,cnt | Shift logical right |
| STC | | Set carry flag |
| STD | | Set direction flag |
| STI | | Set interrupt flag |
| STOS | [dt] | Store string |
| STOSB | " | " byte |
| STOSW | " | " word |
| SUB | dt,sc | Subtraction |
| TEST | dt,sc | Test (logical AND) |
| WAIT | | Wait for 8087 |
| XCHG | dt,sc | Exchange |
| XLAT | table | Translate |
| XLATB | " | " |
| XOR | dt,sc | Logical exclusive OR |

Notes:
dt - destination
sc - source
label - may be near or far address
slabel - near address

| OPCODE | | DESCRIPTION |
|---|---|---|
| AAA | | ASCII adjust addition |
| AAD | | ASCII adjust division |
| AAM | | ASCII adjust multiply |
| AAS | | ASCII adjust subtraction |
| ADC | dt,sc | Add with carry |
| ADD | dt,sc | Add |
| AND | dt,sc | Logical AND |
| CALL | proc | Call a procedure |
| CBW | | Convert byte to word |
| CLC | | Clear carry flag |
| CDL | | Clear direction flag |
| CLI | | Clear interrupt flag |
| CMC | | Complement carry flag |
| CMP | dt,sc | Compare |
| CMPS | [dt,sc] | Compare string |
| CMPSB | " | " bytes |
| CMPSW | " | " words |
| CWD | | Convert word to double word |
| DAA | | Decimal adjust addition |
| DAS | | Decimal adjust subtraction |
| DEC | dt | Decrement |
| DIV | sc | Unsigned divide |

# Lenguajes de bajo nivel y alto nivel




Machine Code
C
Java
Perl
Ruby
Assembly
C++
Scala
JavaScript

# Metodología para la construcción de un algoritmo

1. **Definición** del problema
2. **Análisis** del problema
3. **Diseño** del algoritmo
4. **Verificación** o pruebas

## Datos a extraer del problema:

- **Entrada**
  - ¿Qué se necesita para realizar los pasos?
- **Salida**
  - ¿Qué se obtiene al final del algoritmo?
- **Tipos de datos**
  - Números: enteros, reales, complejos
  - Texto: letras, palabras, frases
  - Otros

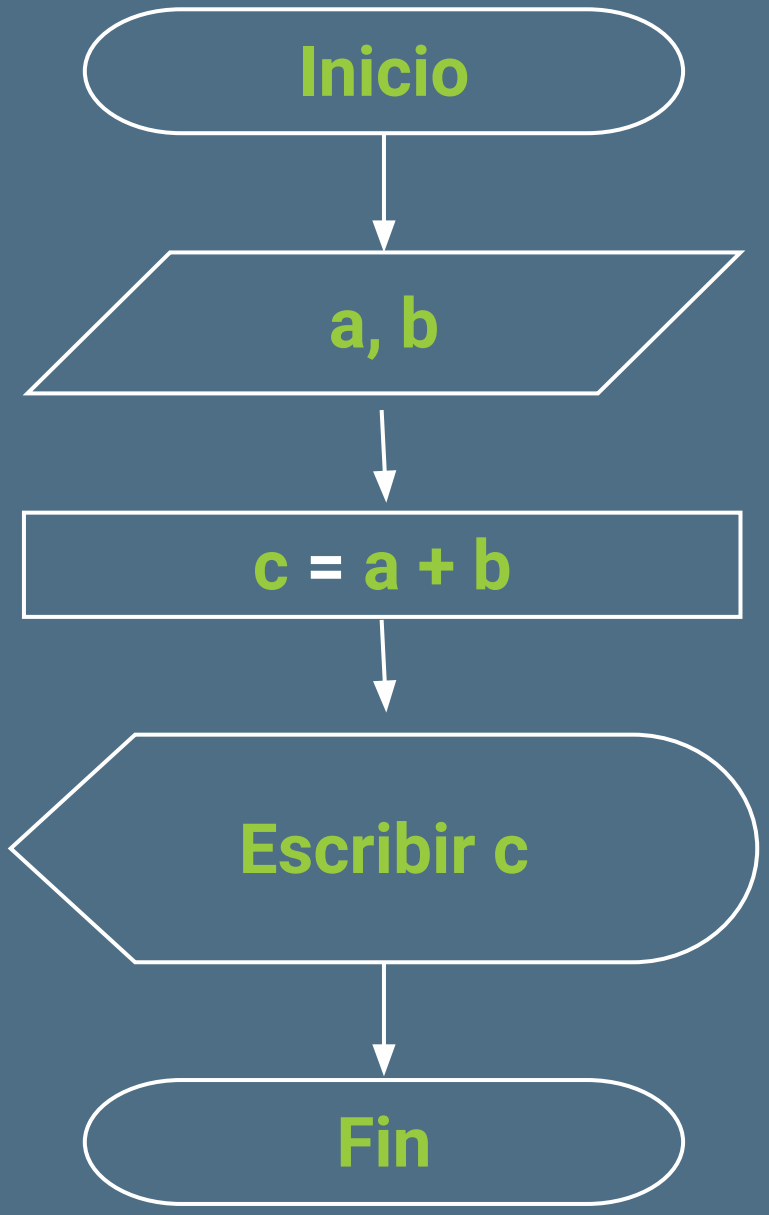
# Calcular la sumatoria de dos números

Platzi

// Entrada de datos

// Proceso

// Salida de información




Platzi

# User Defined Data Types

# Abstract Data Types y las Estructuras de Datos

- Un tipo de dato abstracto (ADT) representa un set particular de comportamientos.
  - Podemos definir con precisión lo que hará un ADT
    - Un stack es una lista que implementa una política “LIFO” en elementos agregados y eliminados.

- Una estructura de datos es más concreta. Típicamente es una técnica o estrategia para implementar una ADT.
  - Por ejemplo, podemos utilizar una Linked List o un Array (estructuras de datos) para implementar un Stack (ADT)

- Algunos de los ADT más comunes que debes conocer como programador preparado son:
  - Stack (pila), Queue (cola), Priority Queue (cola de prioridades), Diccionarios, Trees (árboles) Graphs (grafos).

- Algunas de las estructuras de datos usadas para implementar esos ADTS son:
  - Array, linked list, hash tables.
  - Trees

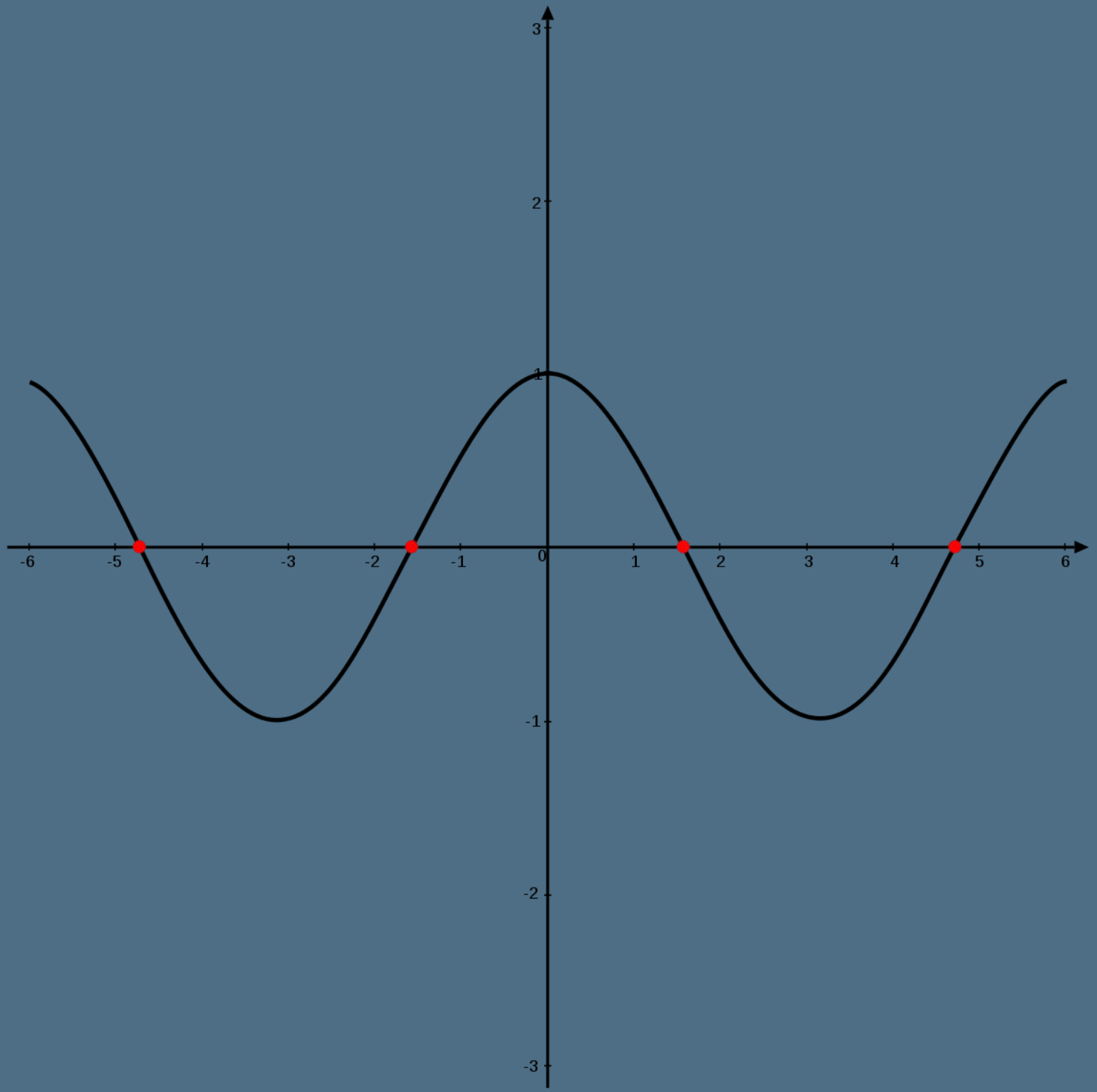
# El objetivo del análisis de algoritmos

Platzi

# Objetivos del análisis de algoritmos

Platzi

# ¿Qué es análisis en tiempo de ejecución?


3
2
1
-1
-2
-3
-6
-5
-4
-3
-2
-1
0
1
2
3
4
5
6

# ¿Cómo comparar algoritmos?

Platzi

# ¿Cómo comparar algoritmos?

- Tiempos de ejecución: no es una buena métrica.
- Número de instrucciones ejecutadas: no es una buena métrica.
- ¿Solución ideal...

# Análisis del ritmo de crecimiento de los algoritmos

Platzi

$$Total\ Cost = cost\_of\_car + cost\_of\_bicycle$$
$$Total\ Cost \approx cost\_of\_car\ (approximation)$$

$$n^4 + 2n^2 + 100n + 500 \approx n^4$$

# Cambiar a vista de Visual Studio para mostrar el ejemplo explicando un insertion sort

# Ratio de crecimiento de los tiempos de ejecución en algoritmos

| Time Complexity | Name |
|---|---|
| 1 | Constant |
| $logn$ | Logarithmic |
| $n$ | Linear |
| $nlogn$ | Linear Logarithmic |
| $n^2$ | Quadratic |
| $n^3$ | Cubic |
| $2^n$ | Exponential |

# Estructuras de control:

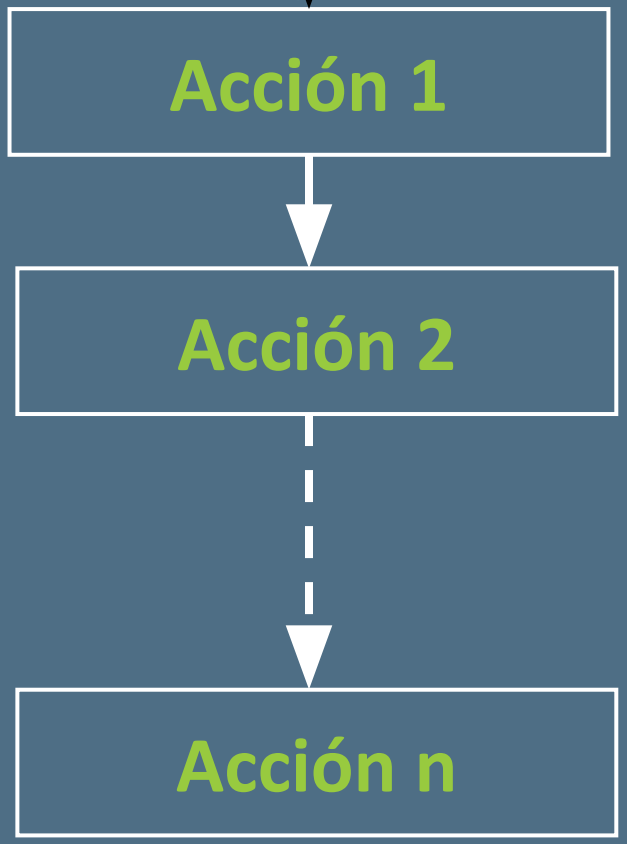
- Secuenciales
- Selectivas
- Repetitivas

# Tipos de datos definidos por el usuario

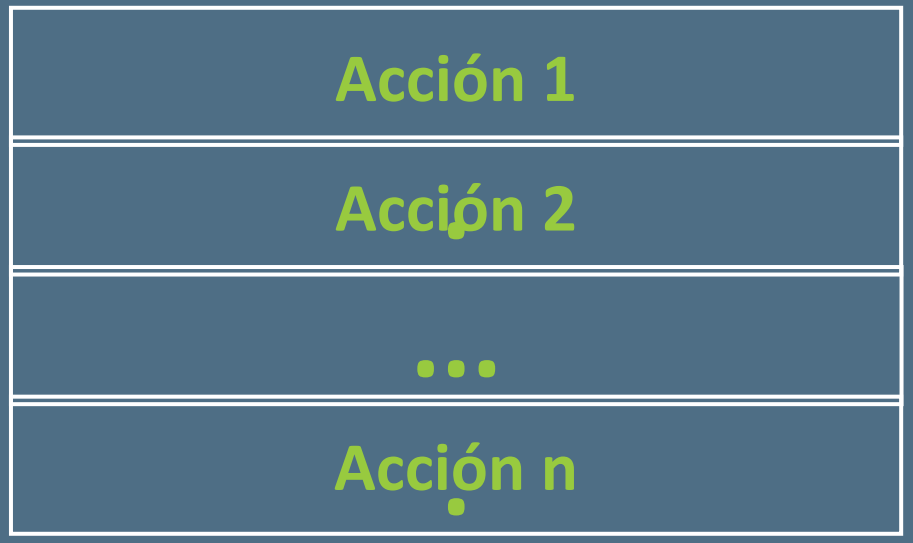
# Estructuras de datos

# Tipos de datos
# abstractos

## Diagrama de flujo

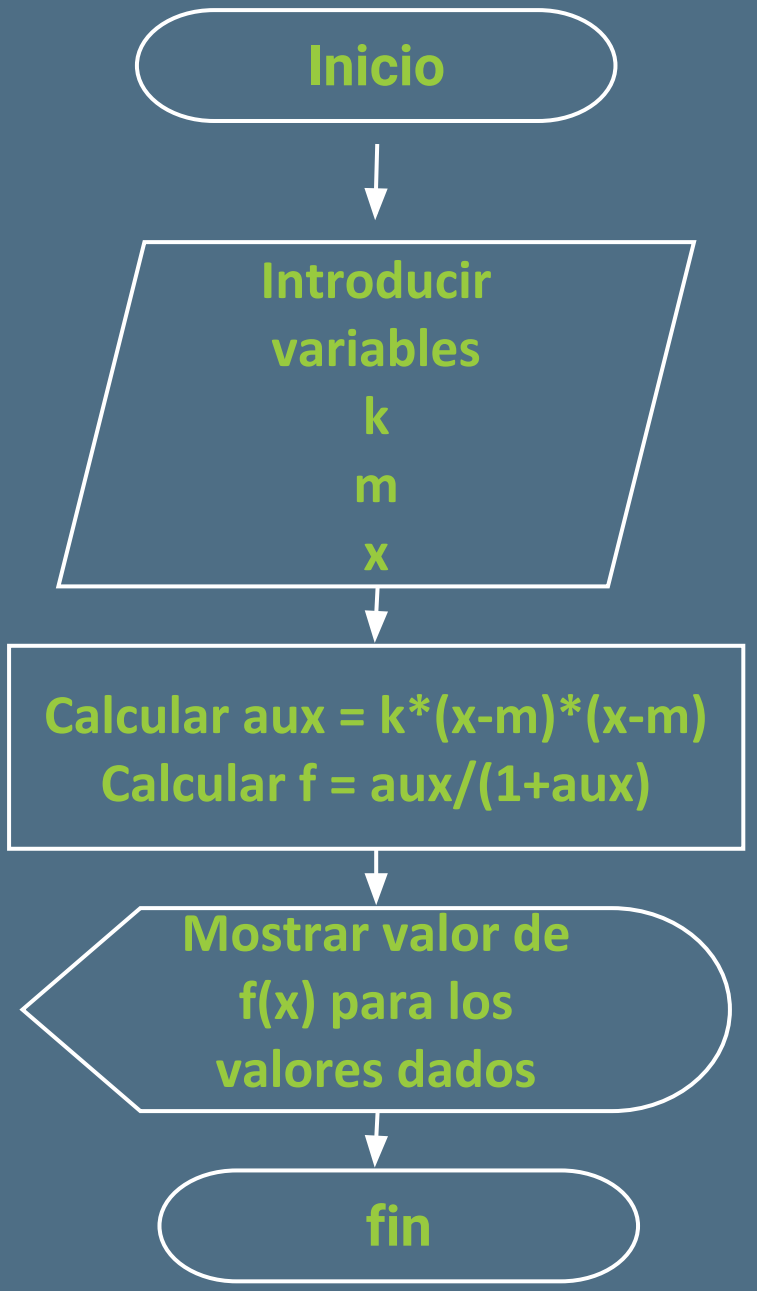



## Diagrama Nassi-Shneiderman



$$f(x) = \frac{k(x-m)^2}{1+k(x-m)^2}$$

## Diagrama de flujo



# Estructuras selectivas:

Si, if

Si - sino, if - else

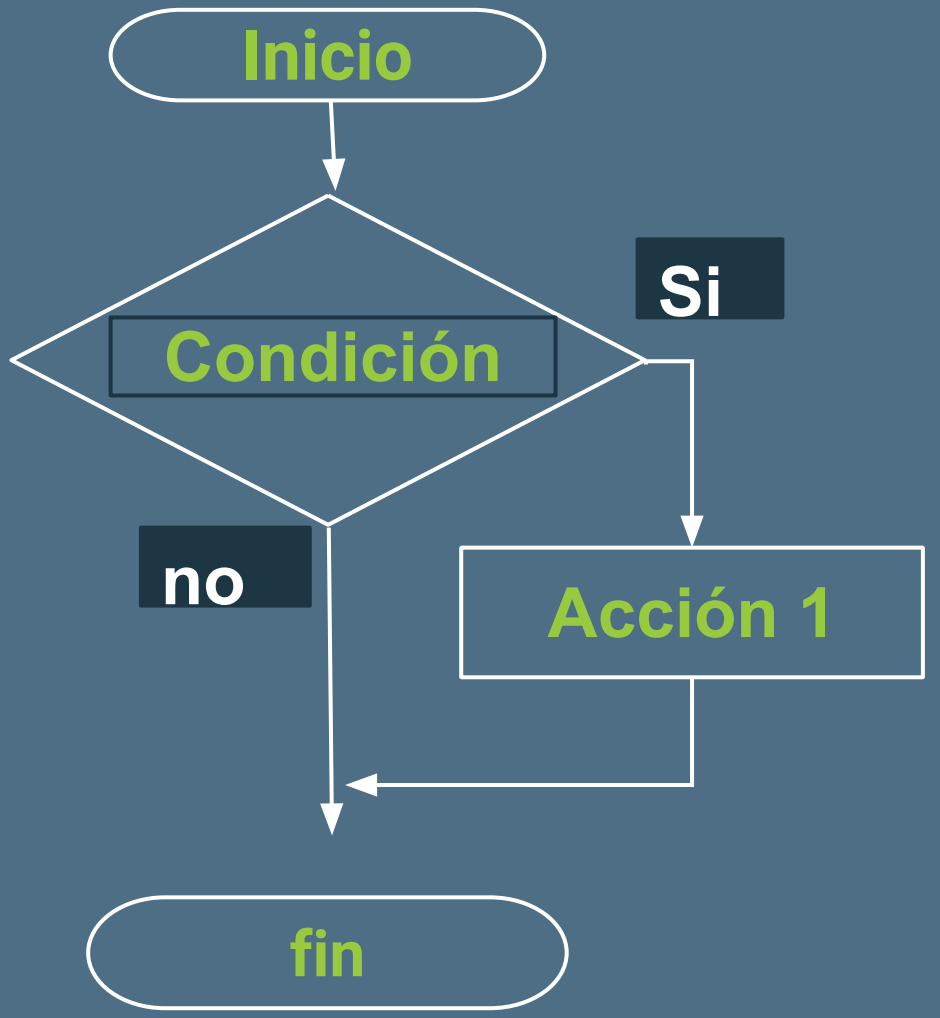
Si, sino entonces… sino,

if, else if, else

## Pseudocódigo

```
Inicio

    Si < Condición >
      entonces <Acción_1>
    Fin_Si
Fin
```

## Diagrama de flujo

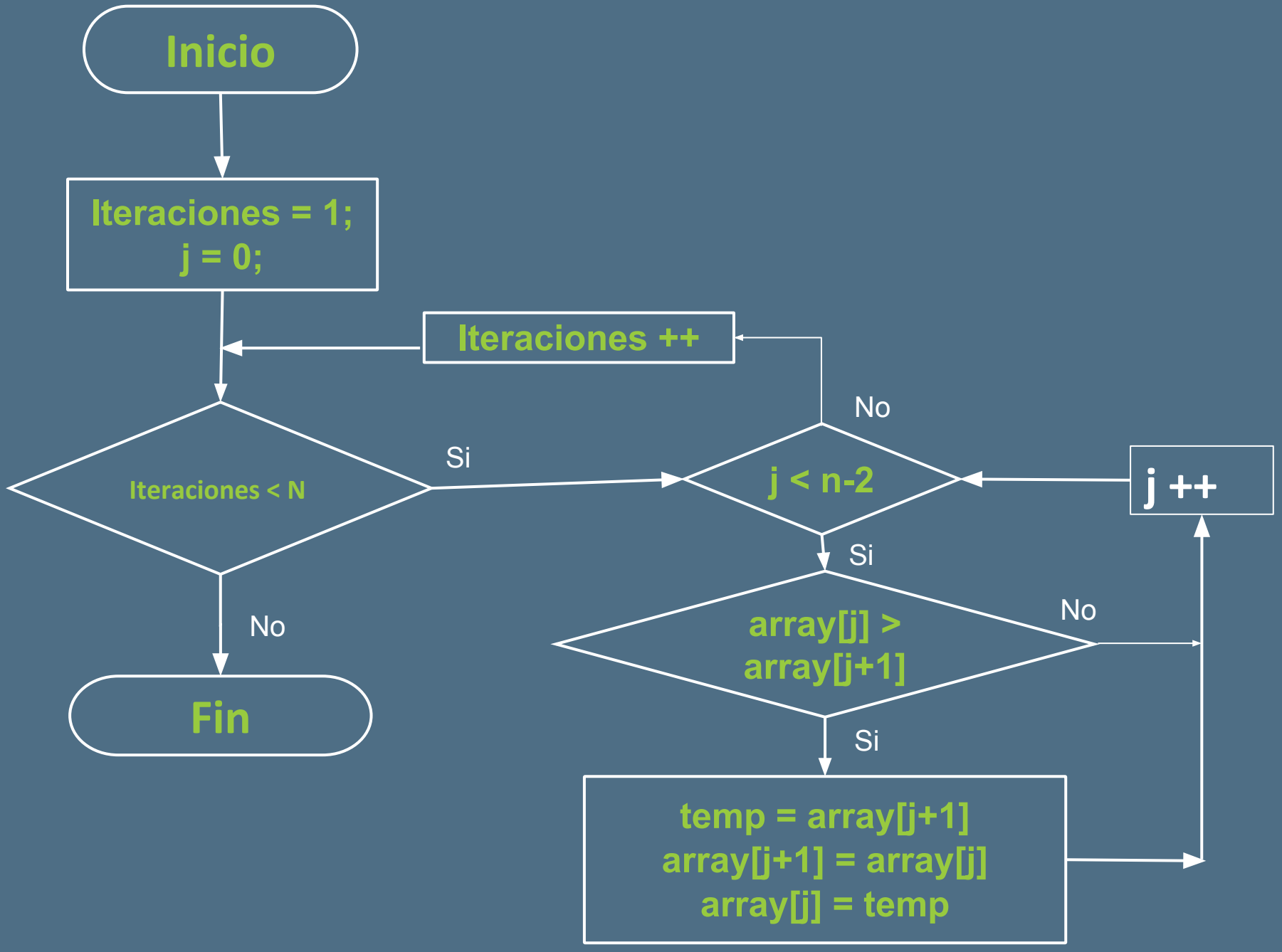


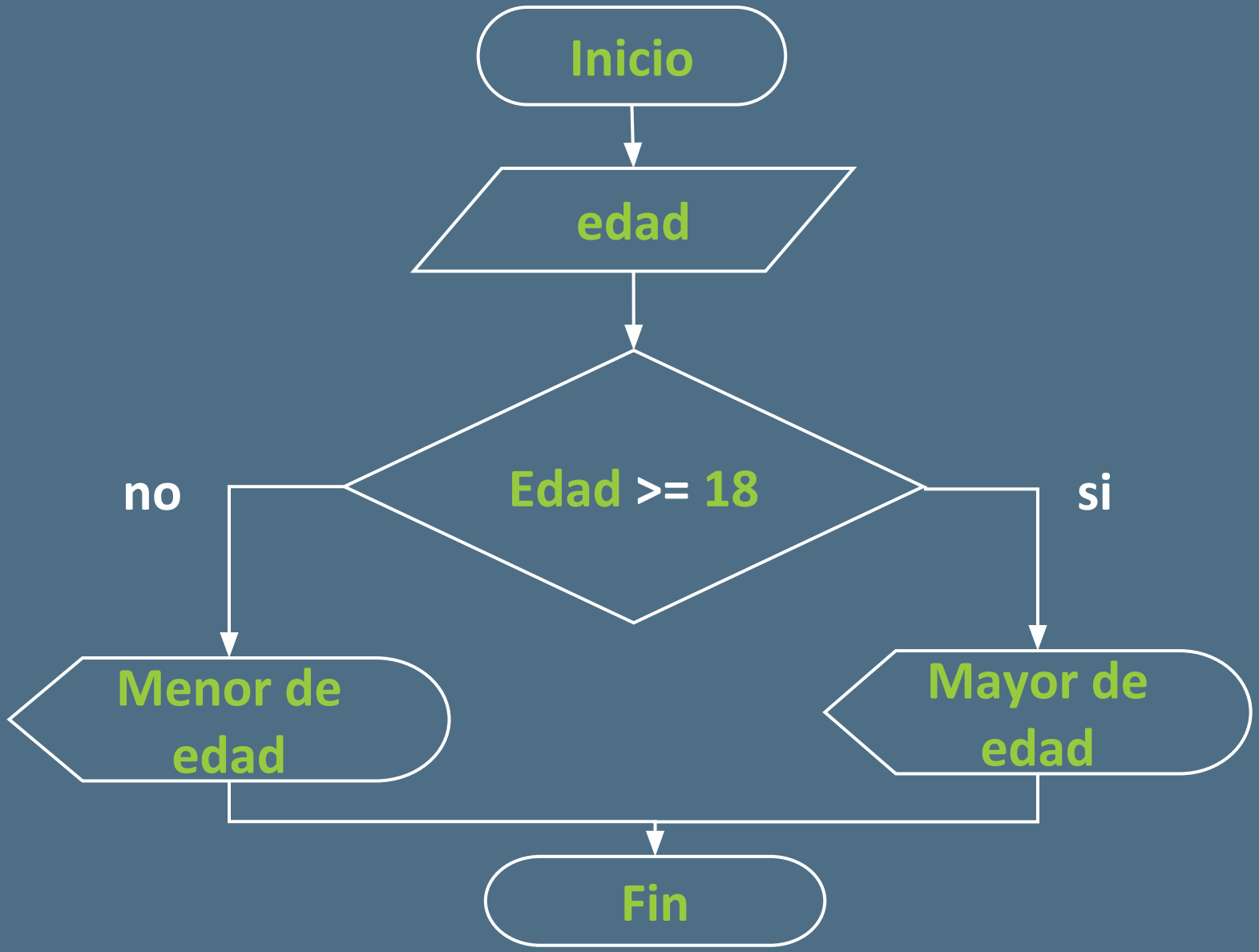
# Estructura de selección doble (si-sino/if-else)

## Diagrama de flujo general


## Pseudocódigo

```
Inicio

Leer EDAD

    Si  EDAD > 18
        entonces  "es  mayor
            de edad"
    Sino "es menor de edad"

    Fin_si

Fin
```

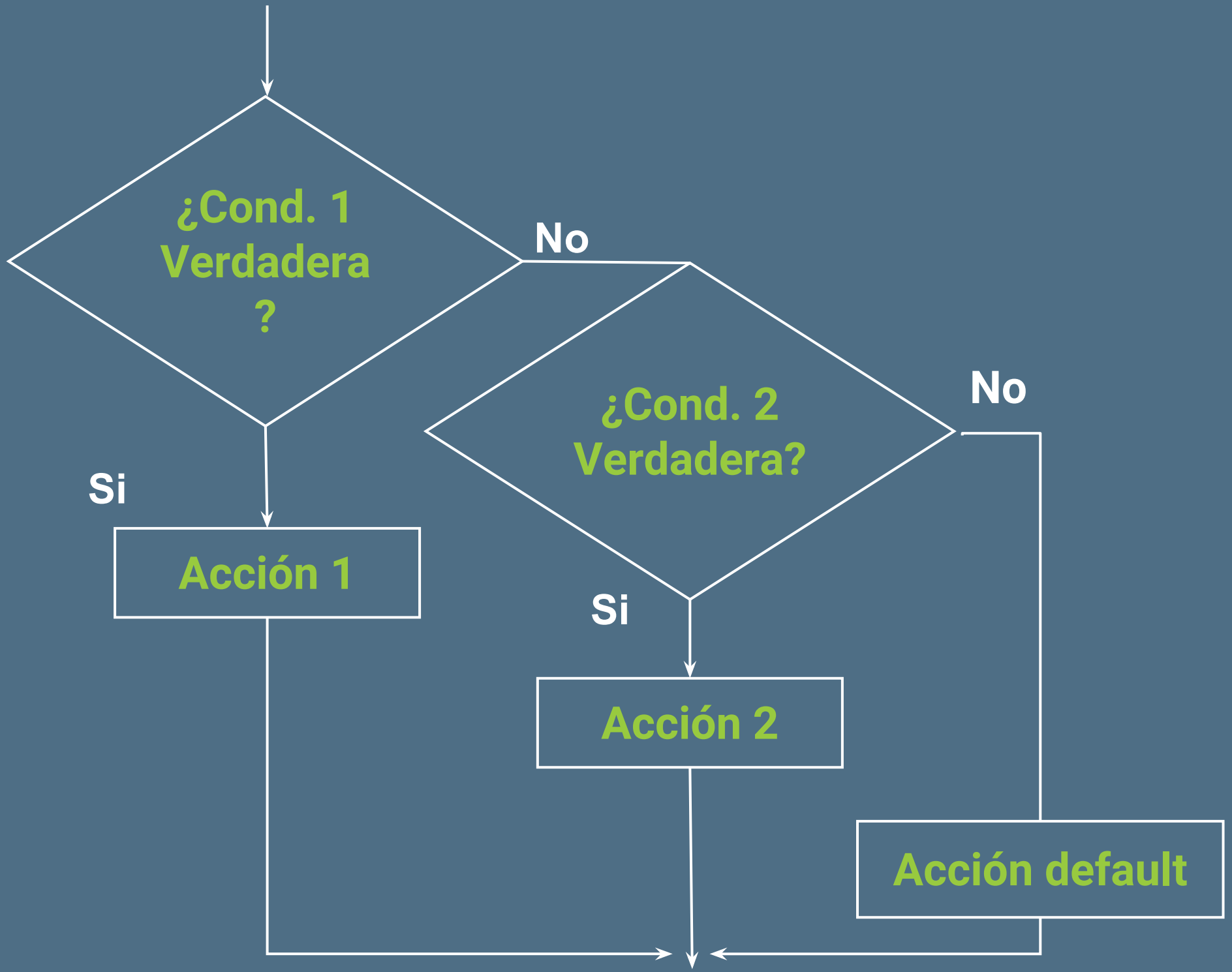

Inicio
Iteraciones = 1;
j = 0;
Iteraciones ++
Iteraciones < N
Si
No
Fin
j < n-2
No
Si
j ++
array[j] >
array[j+1]
No
Si
temp = array[j+1]
array[j+1] = array[j]
array[j] = temp


Inicio
edad
Edad >= 18
no
si
Menor de edad
Mayor de edad
Fin

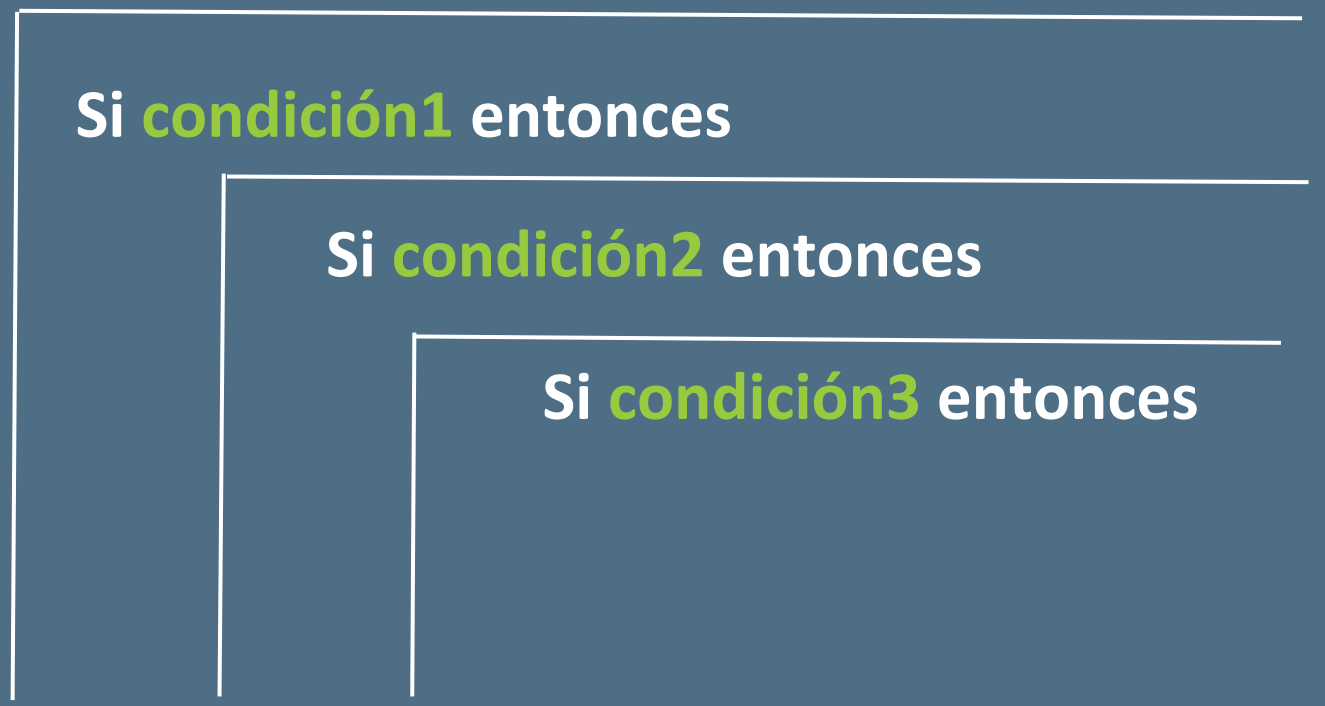
# Estructuras de selección múltiple (si - sino entonces - sino)

Platzi

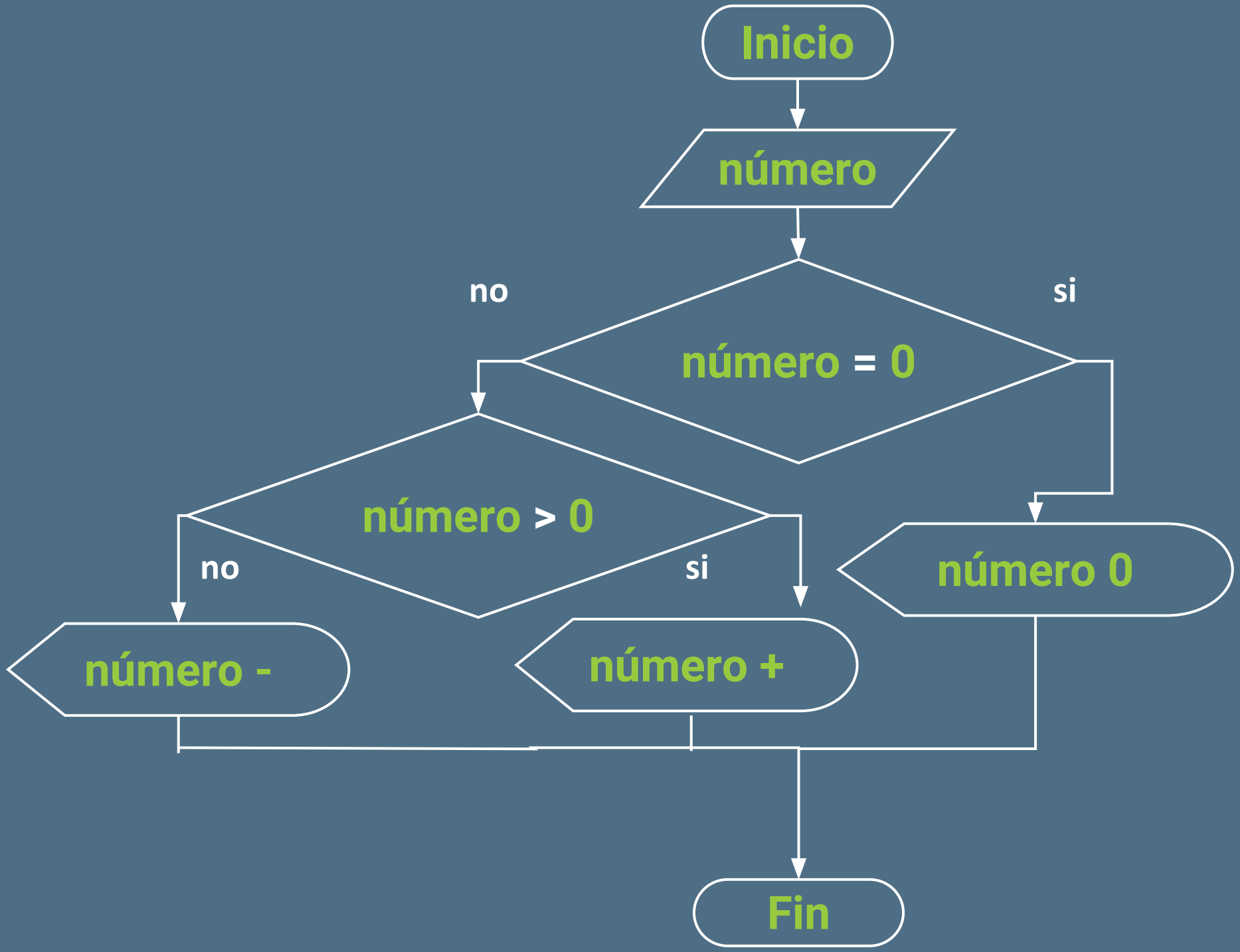
```
SI <condición_1>
Entonces <acción_1 >
      Sino SI <condición_2>
       Entonces <acción_2>
            Sino SI <condición_3>
             Entonces  <acción_3>
                  Sino …
                        …
                        …
                        …

Fin_si
```


¿Cond. 1 Verdadera ?
No
¿Cond. 2 Verdadera?
No
Si
Acción 1
Si
Acción 2
Acción default

# si-entonces / si-entonces-sino




Platzi


Inicio
número
no
si
número = 0
número > 0
no
si
número 0
número -
número +
Fin

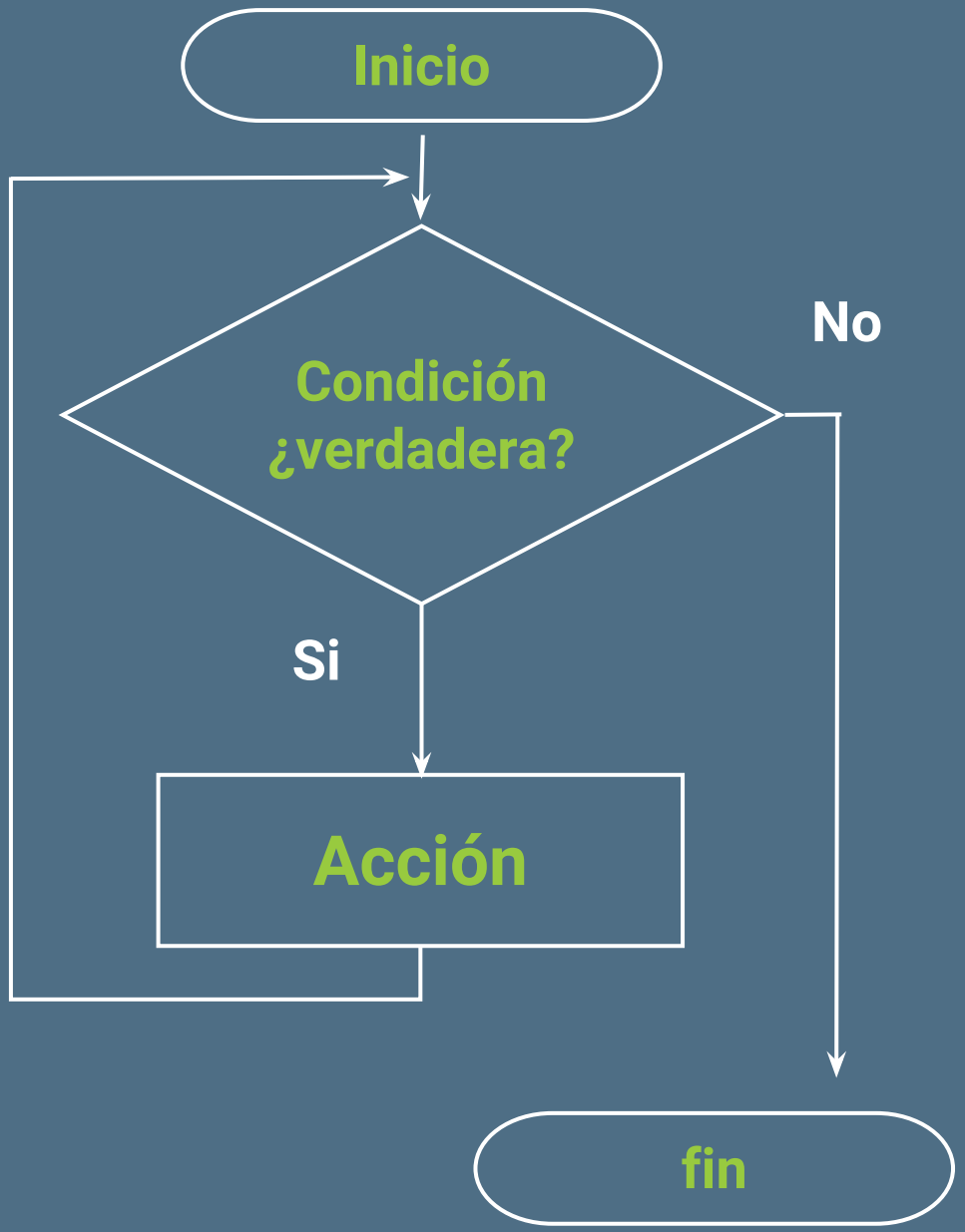
# Estructuras repetitivas:

Mientras, while
Haz mientras, do while
Para, for

Platzi

# Mientras

## Diagrama de flujo

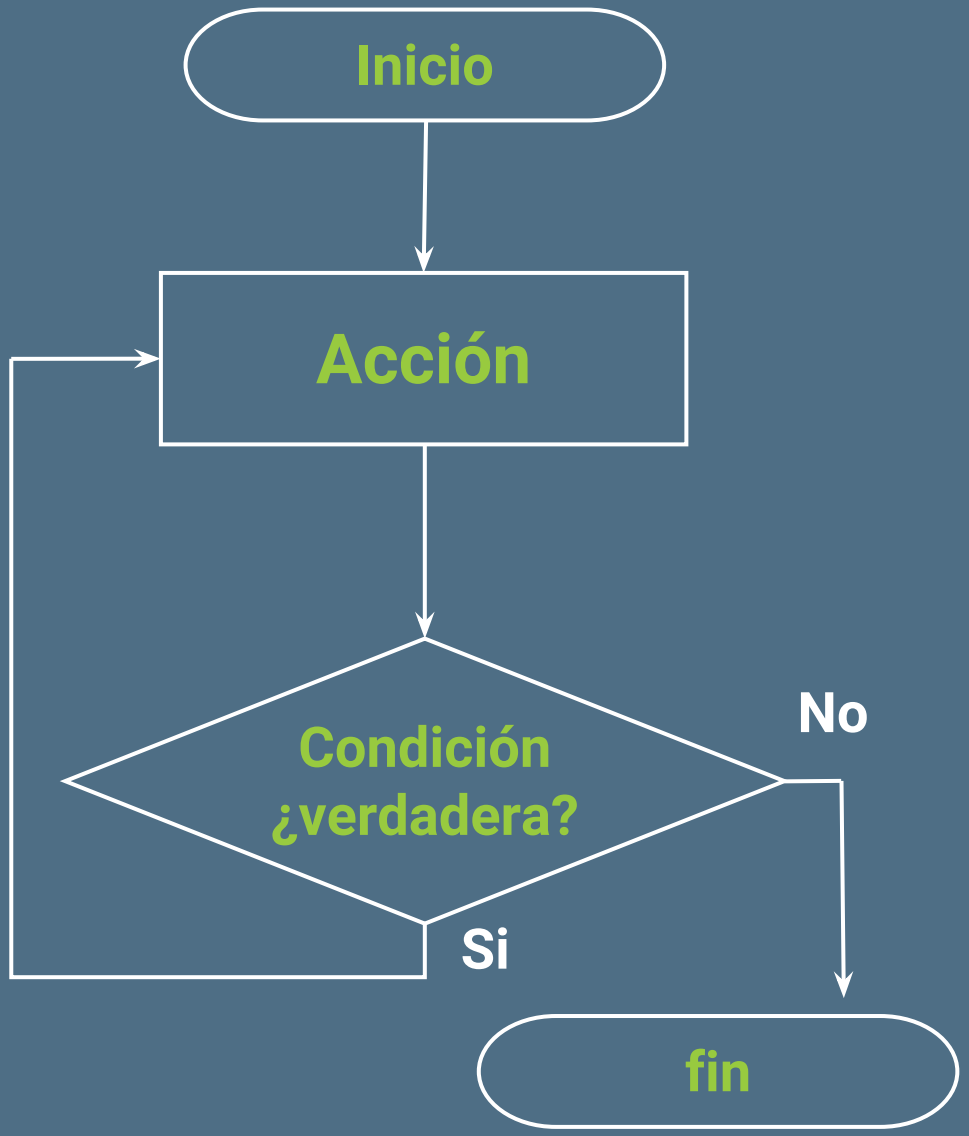



## Pseudocódigo

**Mientras** < Condición >
< Acción >
**Fin_Mientras**

# Haz Mientras

## Diagrama de flujo

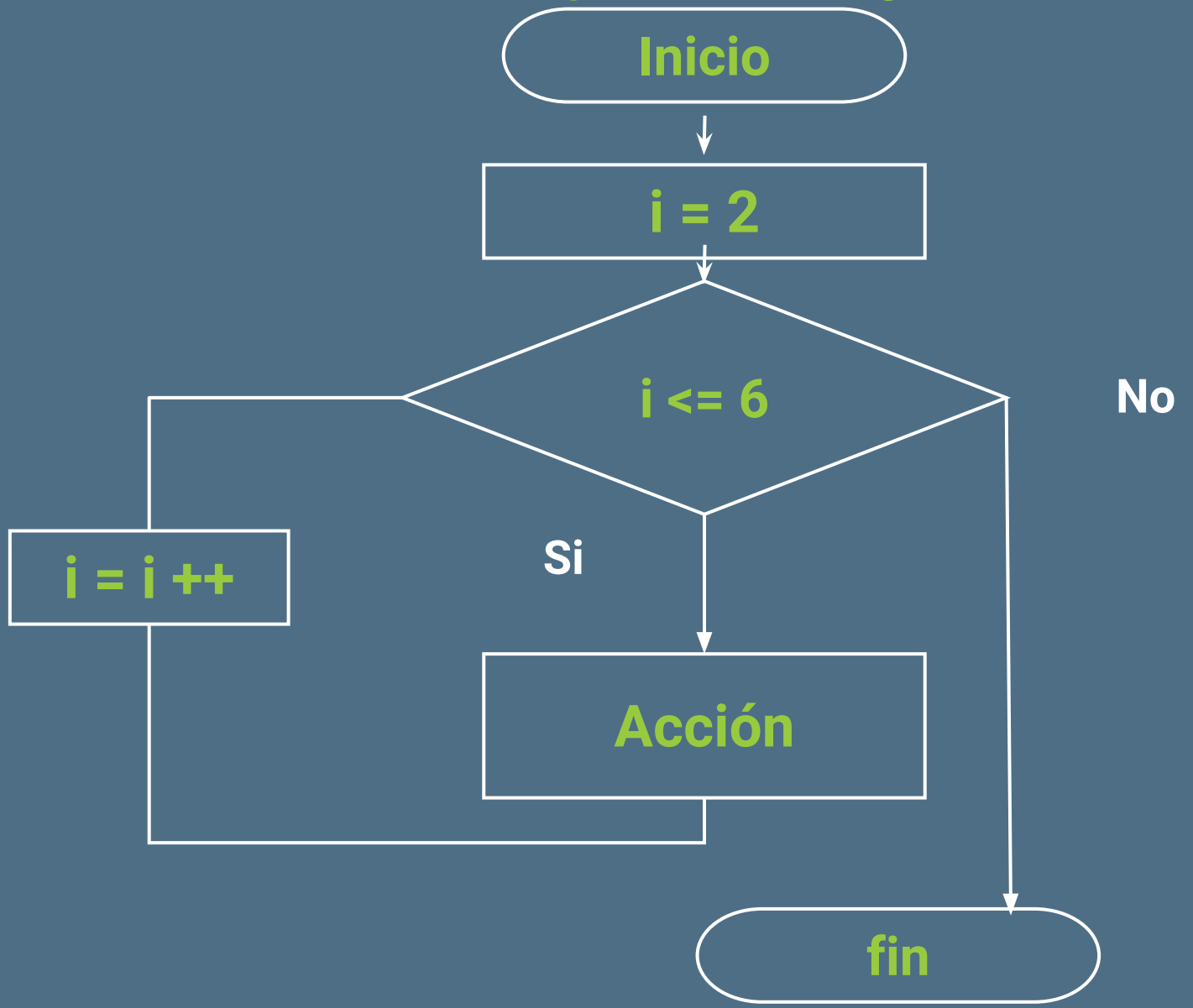



## Pseudocódigo

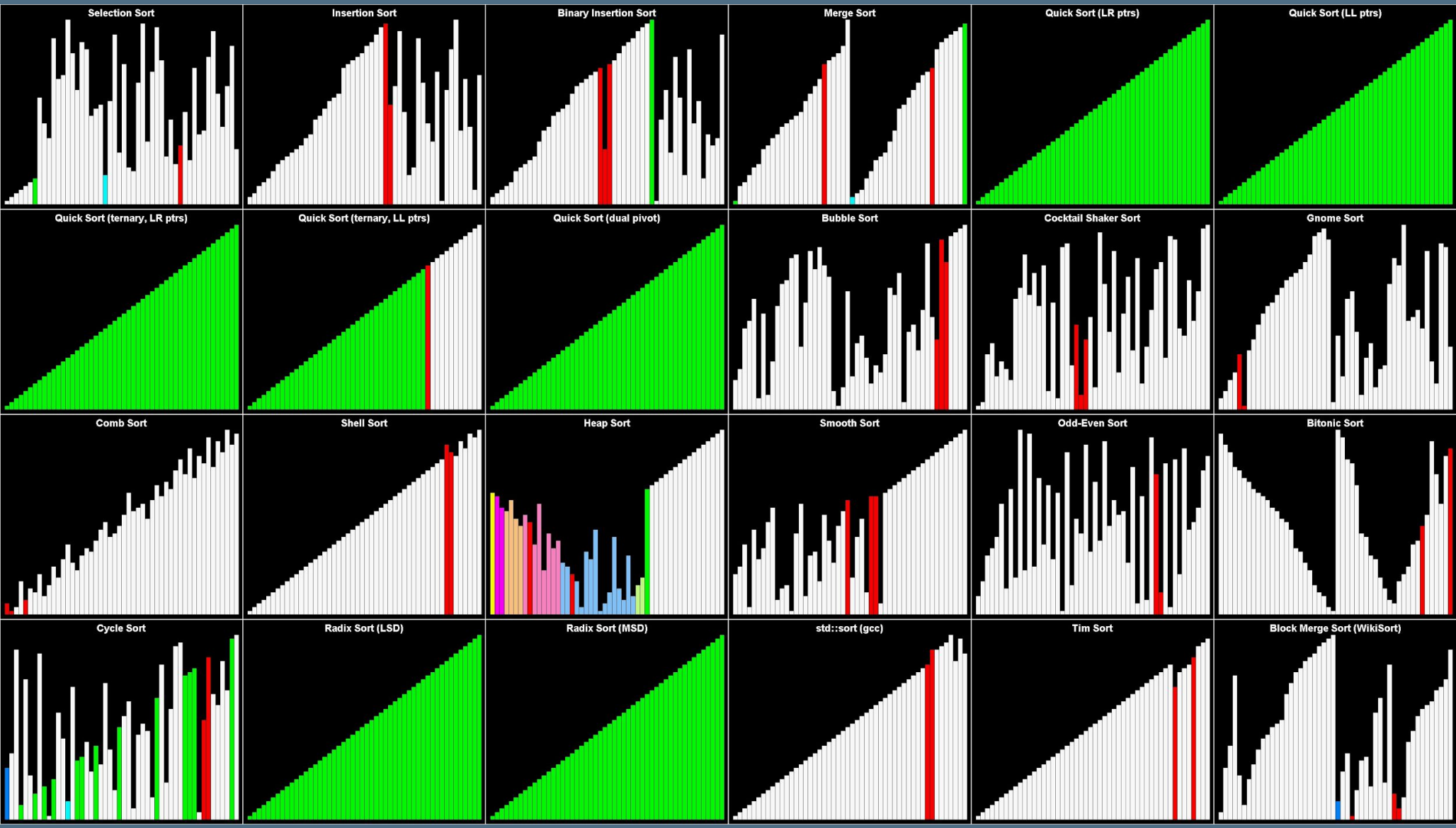
**Inicio**
    **Haz**
    **<** Acción **>**
    **Mientras <** Condición **>**
**Fin**

# Para

## Diagrama de flujo



# Algoritmos de ordenamiento


Selection Sort
Insertion Sort
Binary Insertion Sort
Merge Sort
Quick Sort (LR ptrs)
Quick Sort (LL ptrs)
Quick Sort (ternary, LR ptrs)
Quick Sort (ternary, LL ptrs)
Quick Sort (dual pivot)
Bubble Sort
Cocktail Shaker Sort
Gnome Sort
Comb Sort
Shell Sort
Heap Sort
Smooth Sort
Odd-Even Sort
Bitonic Sort
Cycle Sort
Radix Sort (LSD)
Radix Sort (MSD)
std::sort (gcc)
Tim Sort
Block Merge Sort (WikiSort)

# ¿Qué es un algoritmo de ordenamiento?

**Un algoritmo de ordenamiento colocará los objetos (números o letras) en orden.**

**Un arreglo ordenado, es un arreglo que tiene un orden específicamente definido.**

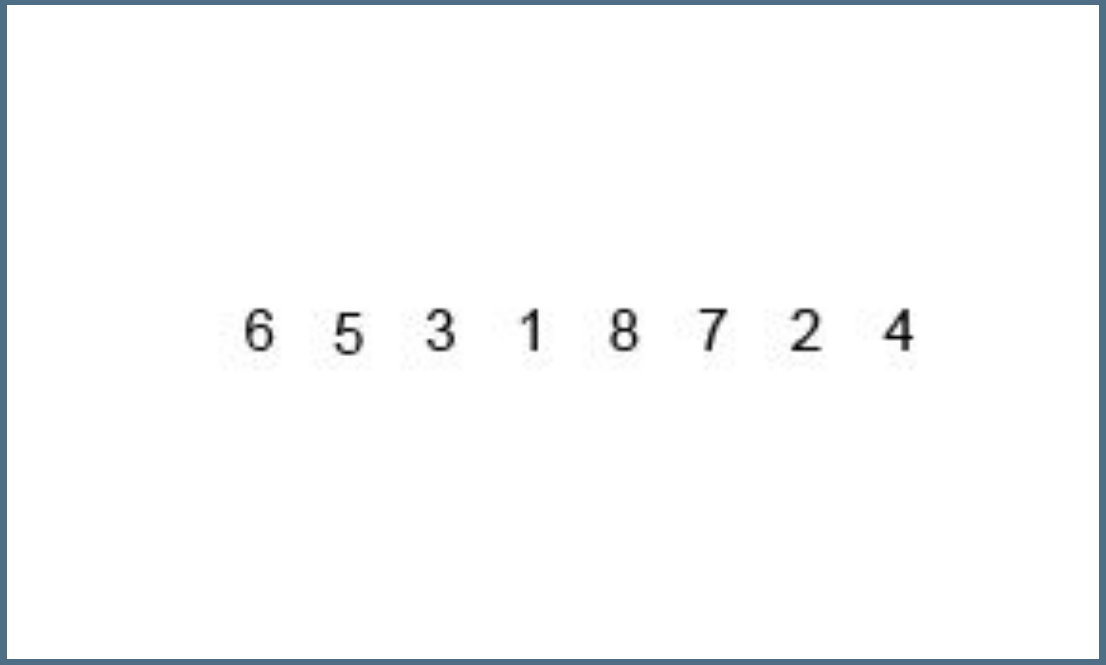
**Por ejemplo:**

**[a, b, c, d] es un arreglo ordenado alfabéticamente.**

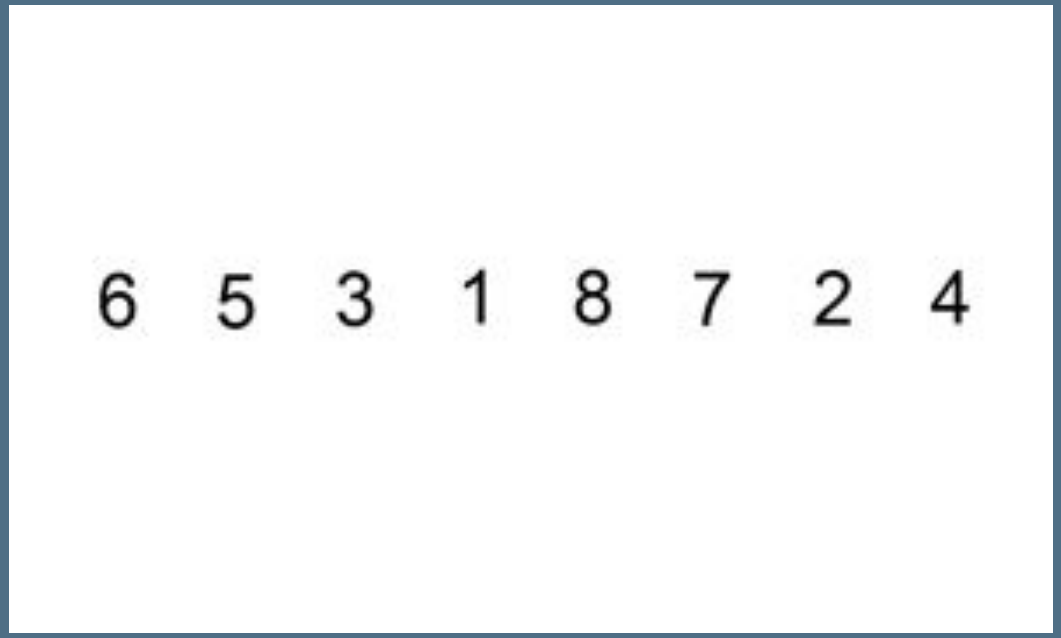
**[1, 2, 3, 4, 5] es un arreglo de números enteros ordenados ascendentemente**

# Algoritmos de ordenamiento más utilizados

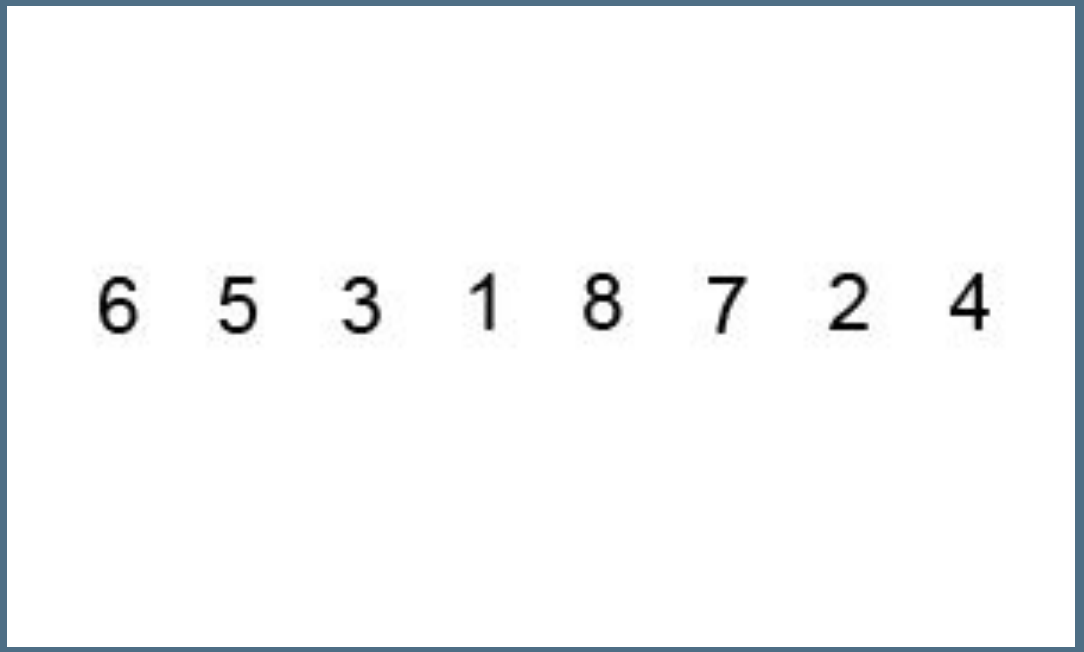
# Merge Sort



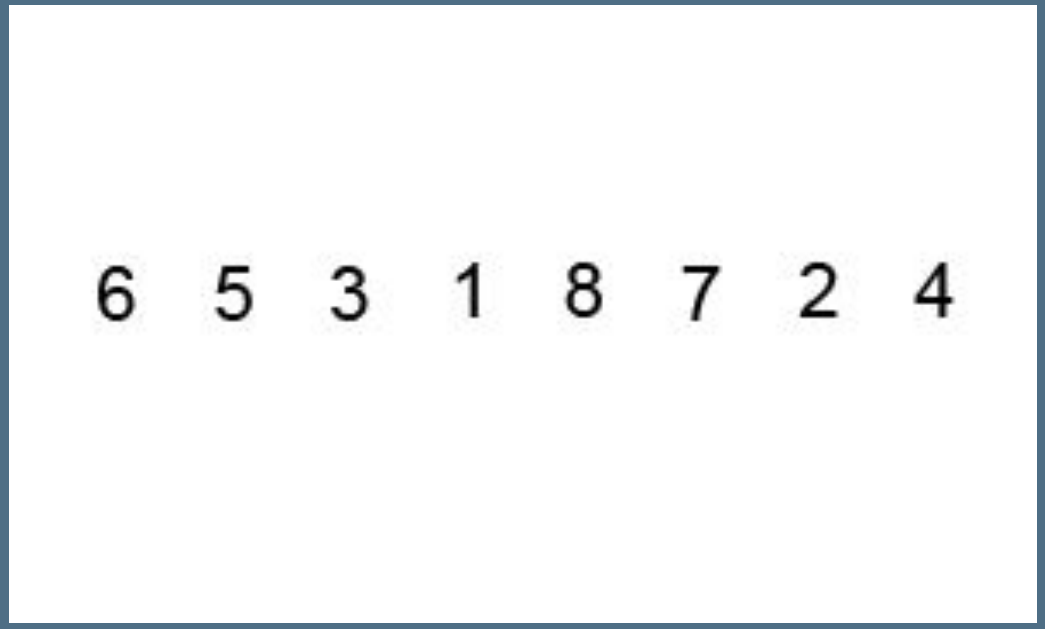
# Insertion Sort



# Bubble Sort



# Quick Sort



S : secuencia de objetos ordenables
N : no. de elementos en S