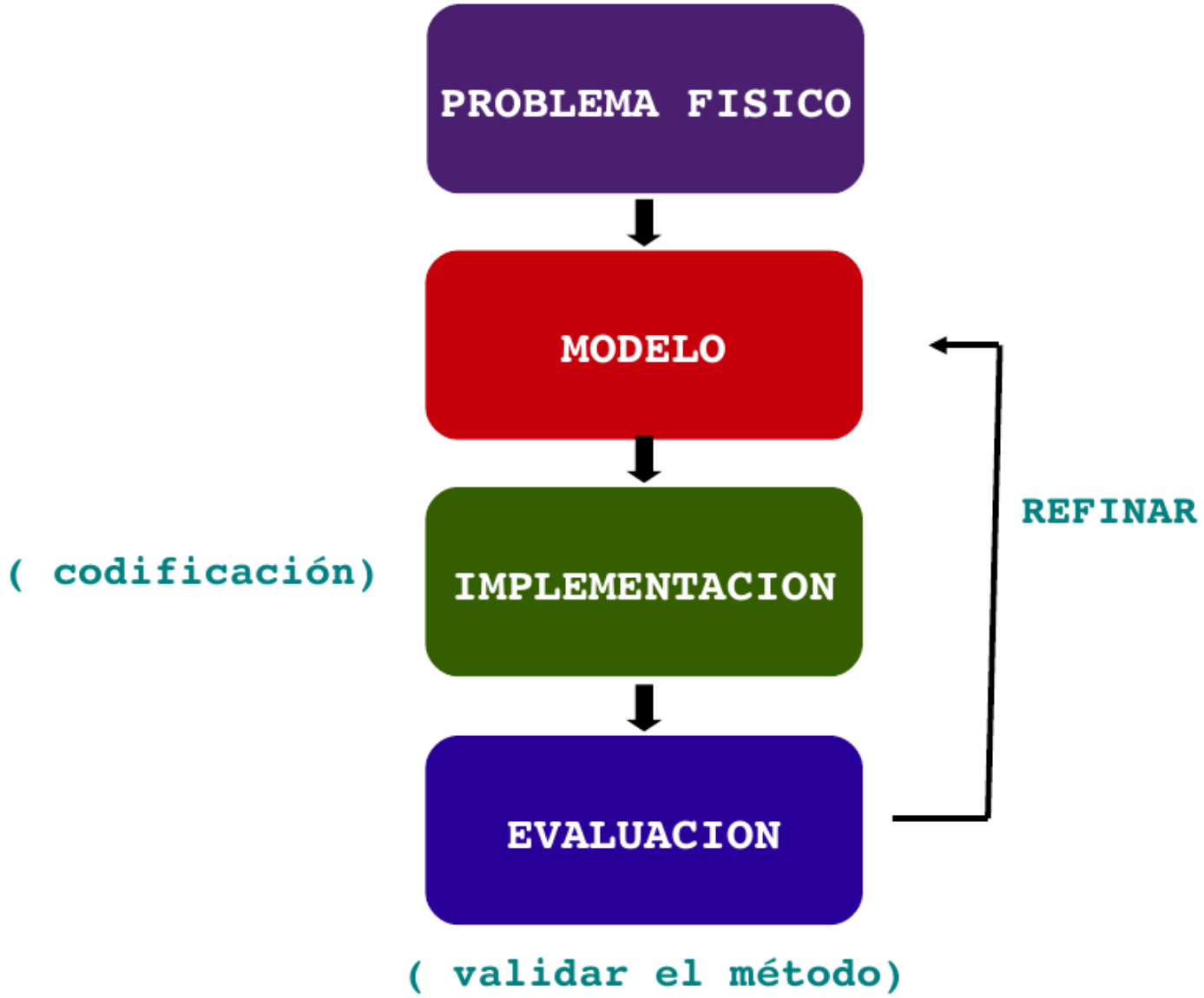
# Optimización

M. Graciela Molina

m.graciela.molina@gmail.com

WTPC workshop en técnicas de programación científica

# El proceso de escribir software científico




WTPC Workshop en técnicas de programación científica

# El proceso de escribir software científico

**Nos acordemos de la clase de Cecilia :)**

## Desarrollo de Software de "convencional"

- Requisitos: cliente y/o manager
- Típica estrategia:
  - Decidir un enfoque general para la implementación
  - Traducir requisitos en tareas/subtareas
  - Usar metodología de admin de proyecto (asegura la implementación, validación y entrega a tiempo)
- Si alcanzo los requisitos: fin del proyecto

## Desarrollo de Software de científico

- *Requisitos no siempre bien definidos.*
- *Limitaciones de matemática de punto flotante, la validación se complica.*
- *La aplicación tal vez sea necesaria una única vez (o pocas veces).*
- *Muy pocos científicos son programadores.*

WTPC workshop en técnicas de programación científica

# El proceso de escribir software científico

Codificamos sin mucha planificación
(que tan difícil puede ser???)

↓

La primera versión que "anda" (resultados correctos?!)

WTPC
workshop en técnicas de programación científica

# El proceso de escribir software científico

Pero cuando el problema crece...

Recién notamos que no son **soluciones óptimas**

WTPC
Workshop en técnicas de programación científica

# Que optimizamos

Debemos pensar en lograr un trade-off entre:

Tiempo de desarrollo, Debugging, Validación,portabilidad, etc.
Y claro el tiempo de ejecución

Importante: El tiempo de CPU es más económico que el tiempo humano !

WTPC
workshop en técnicas de programación científica

# Que optimizamos

Seguramente alguien ya nos resolvió parte del problema !

**No vamos a inventar la rueda**

# Usar librerías optimizadas

Algunas librerías científicas:

- LAPACK — Linear Algebra PACKage (http://www.netlib.org/lapack/)
- GSL - GNU Scientific Library (https://www.gnu.org/software/gsl/)
- C++ Boost (http://www.boost.org/)
- Scipy (https://www.scipy.org/)
- Numpy (http://www.numpy.org/)
- Pandas (http://pandas.pydata.org/)

WTPC workshop en técnicas de programación científica

# Que más podemos hacer

APRENDER SKILLS DE GOOGLEO!
(Evaluar lo que existe)
Seguramente conseguiremos sw de mejor calidad que si intentamos escribirlo nosotros mismos

Y
si tenemos que programar, "ELEGIR" UN LENGUAJE DE PROGRAMACIÓN
(Tener presente qué existe, conocer sus fortalezas y debilidades)

WTPC
workshop en técnicas de programación científica

# Que más hay para optimizar

- Usar aproximaciones cuando sea posible.
- Desarrollar algoritmos más eficientes
- Utilizar estructuras de datos apropiadas
- Obtener hardware más veloz
- Usar o escribir software optimizado para el hardware que se posee. Por ejemplo: aprovechar bibliotecas ya optimizadas, BLAS (www.netlib.org/blas) , LAPACK (www.netlib.org/lapack),etc.
- Paralelizar

# Optimización

- Respecto a los **algoritmos**.
- Respecto a las **estructuras de datos**
- Respecto al **hardware**

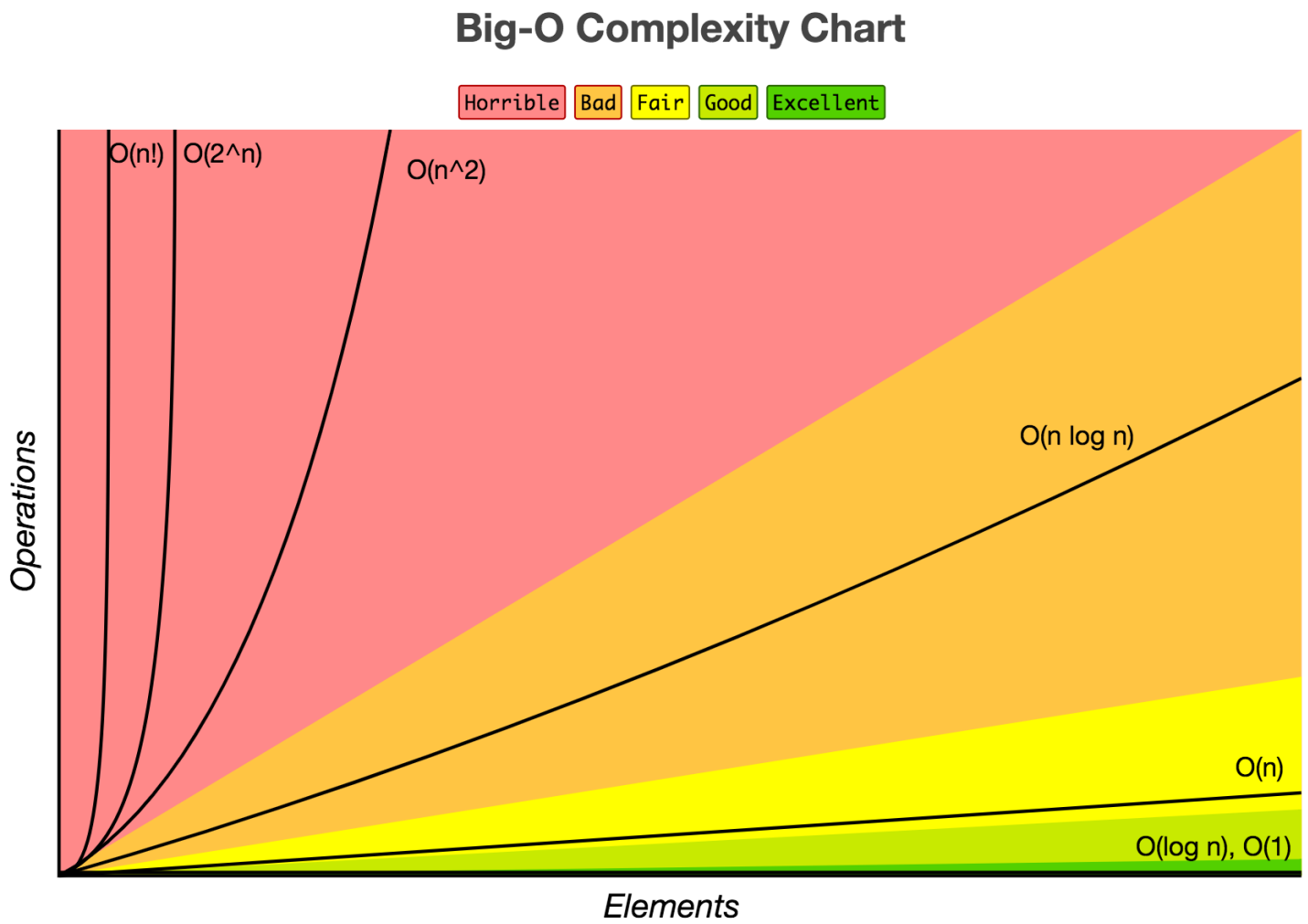
# Cómo elegir el algoritmo adecuado

- **Evaluación teórica:**.
  Complejidad en tiempo y almacenamiento. (Analizar como se comporta el programa a medida que el tamaño de la entrada crece)

- **Evaluación práctica:**
  Realizar mediciones (profiling)

# Algoritmos: evaluación teórica

Complejidad asintótica y notación O grande: Estima de qué manera crecerá el tiempo de ejecución a medida que aumente el tamaño de la entrada.



# Algoritmos: evaluación teórica

| n | O(1) | O(log(n)) | O(n) | O(n log(n)) | $O(n^2)$ | $O(n^3)$ |
|---|---|---|---|---|---|---|
| 10 | const | 3 | 10 | 33 | 100 | 1000 |
| 1000 | const | 10 | 1000 | 9966 | 1E+06 | 1E+09 |
| 100000 | const | 17 | 100000 | 1660964 | 1E+10 | 1E+15 |
| 1000000 | const | 20 | 1000000 | 19931569 | 1E+12 | 1E+18 |

$$O(1) < O(\log\ n) < O(n) < O(n^2) < O(n^3) ... < O(c^n)$$

# Ejemplo: Algoritmos de ordenación

Queremos ordenar un arreglo

$$[\ 9\ 0\ 0\ 4\ 3\ 4\ 1\ 2] \rightarrow [0\ 0\ 1\ 2\ 3\ 4\ 4\ 9]$$

Algoritmo 1

$$[9\ 0\ 0\ 8\ 3\ 4\ 1\ 2] \rightarrow \quad [\mathbf{0}\ \mathbf{9}\ 0\ 8\ 3\ 4\ 1\ 2]$$
$$[0\ \mathbf{0}\ \mathbf{9}\ 8\ 3\ 4\ 1\ 2]$$
$$[0\ 0\ \mathbf{8}\ \mathbf{9}\ 3\ 4\ 1\ 2]$$
$$\ldots$$

Al cabo de n comparaciones, solo se ubicó un elemento.

Necesitamos ahora comparar los n-1 elementos restantes.

# Ejemplo: Algoritmos de ordenación

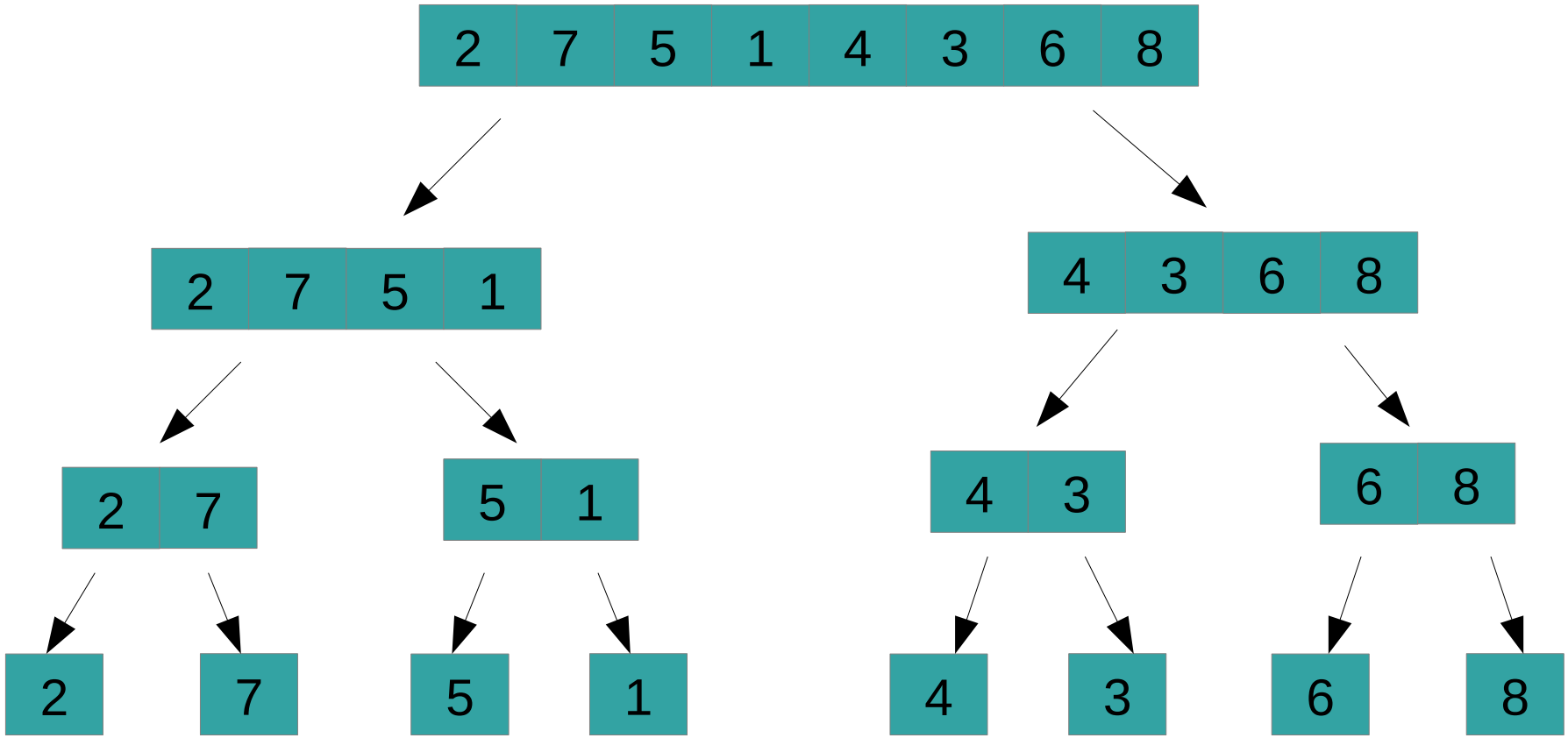
Analicemos la complejidad algoritmica:

Mejor caso: el arreglo ya esta ordenado
Peor caso: el arreglo esta en el orden inverso
En cualquier caso el algoritmo requiere **n** comparaciones x **n** elementos que tiene el arreglo $\rightarrow O(n^2)$

Se puede mejorar este algoritmo?

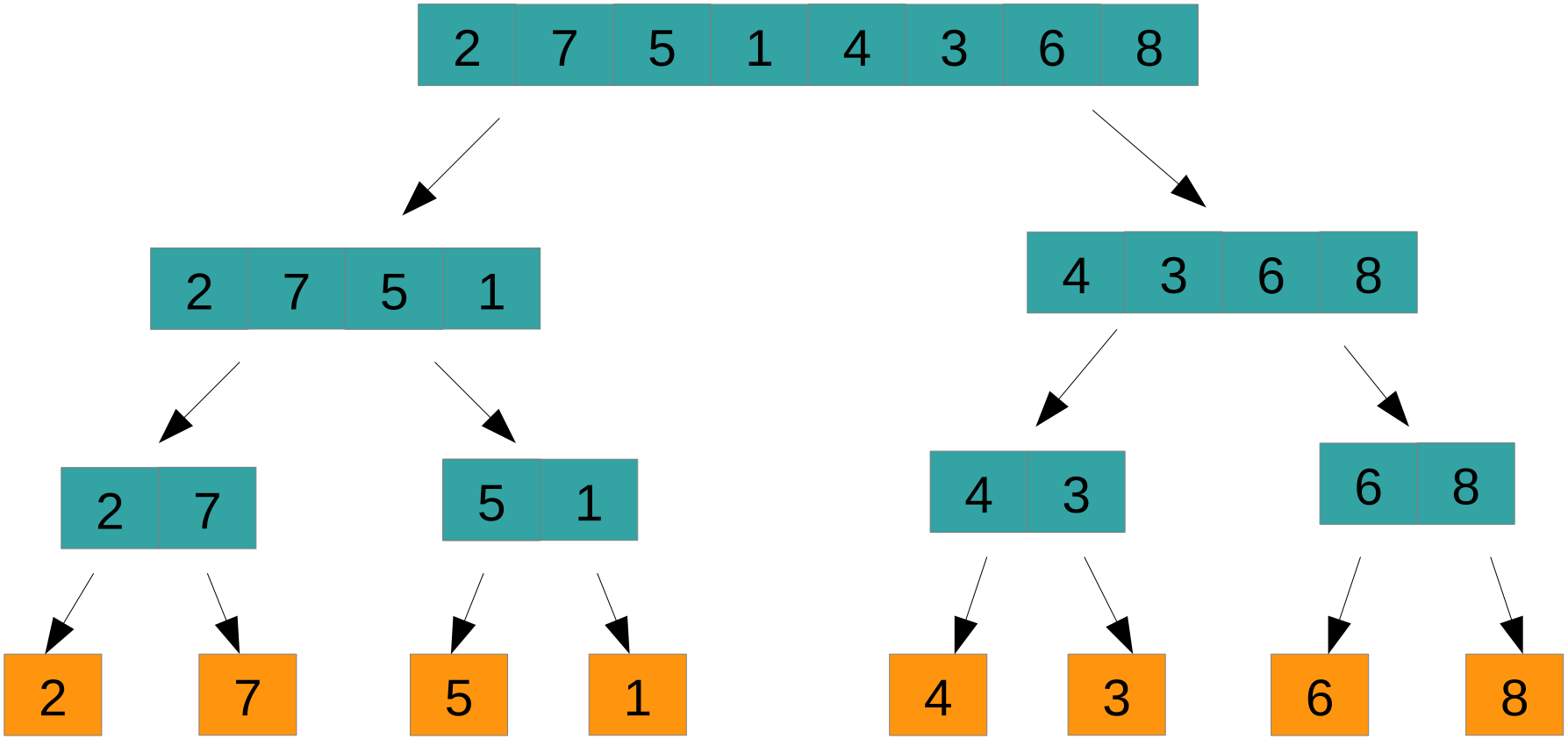
# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



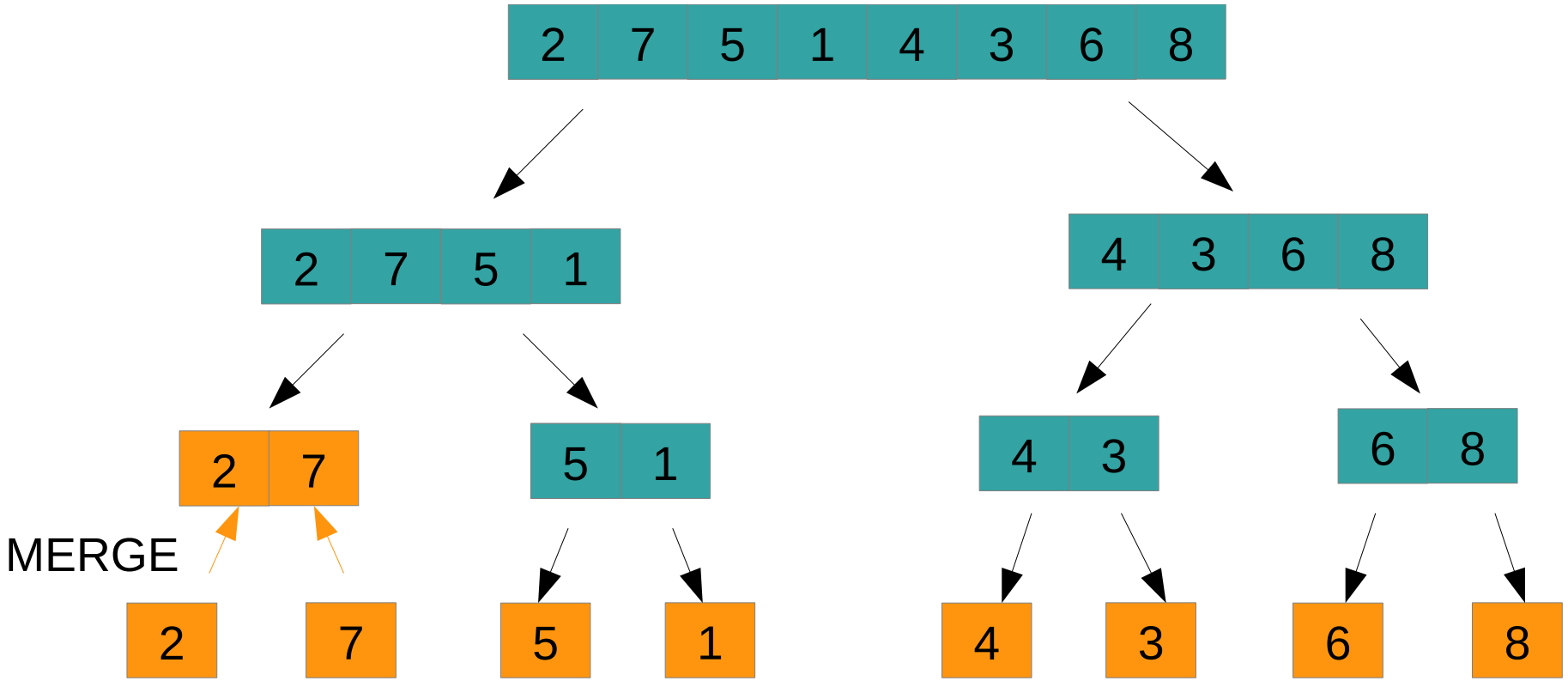
# Ejemplo: Algoritmos de ordenación

## Algoritmo 2: Merge sort



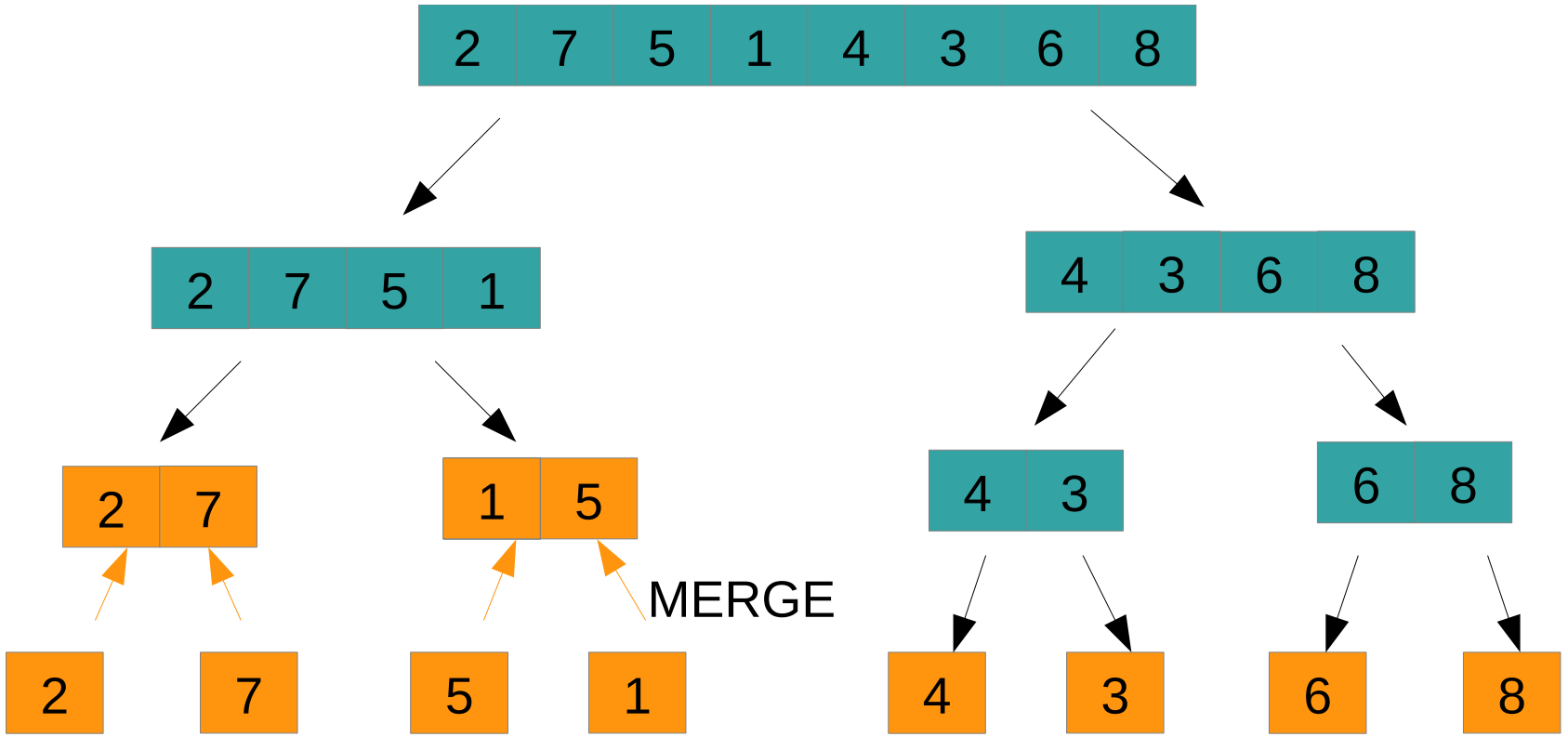
# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



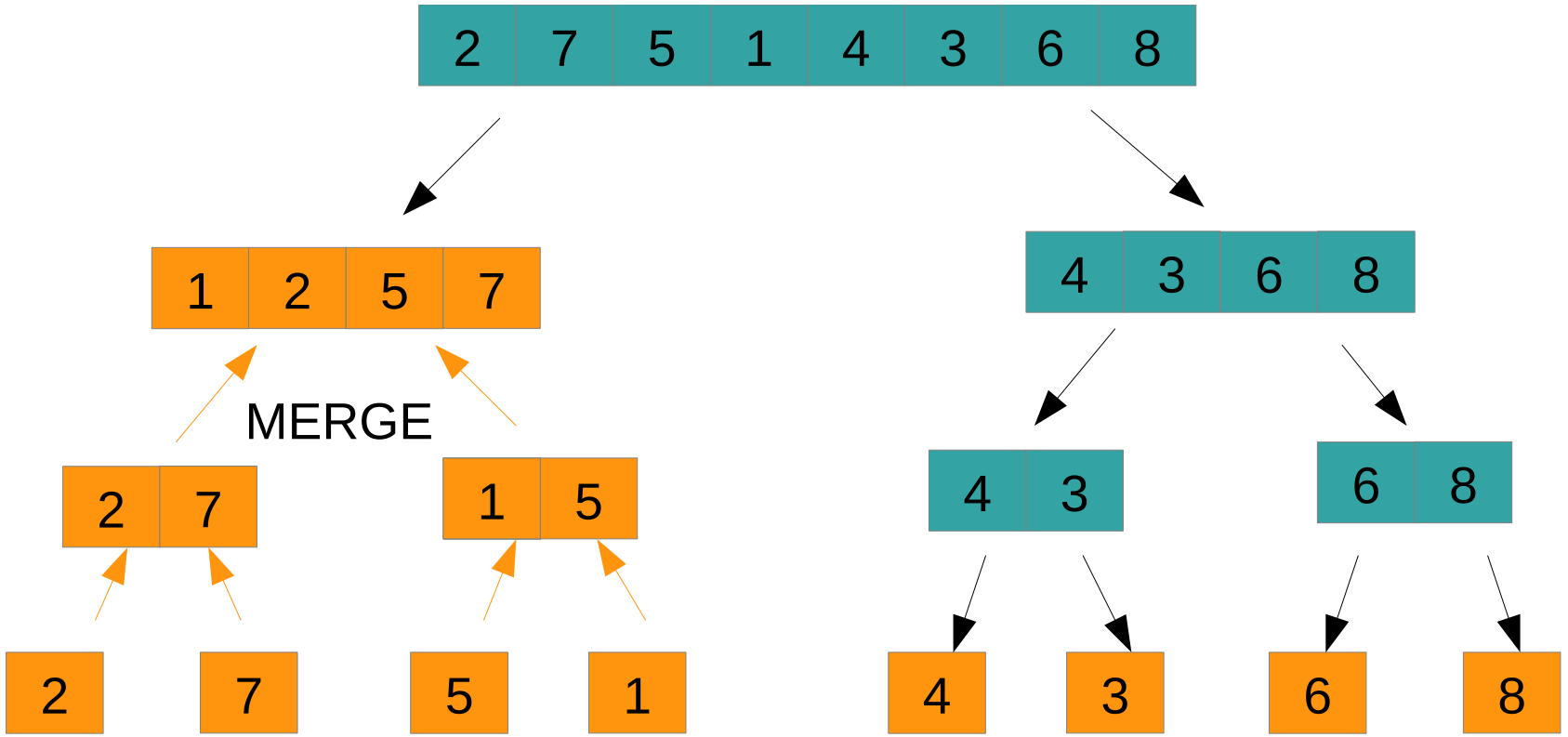
# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



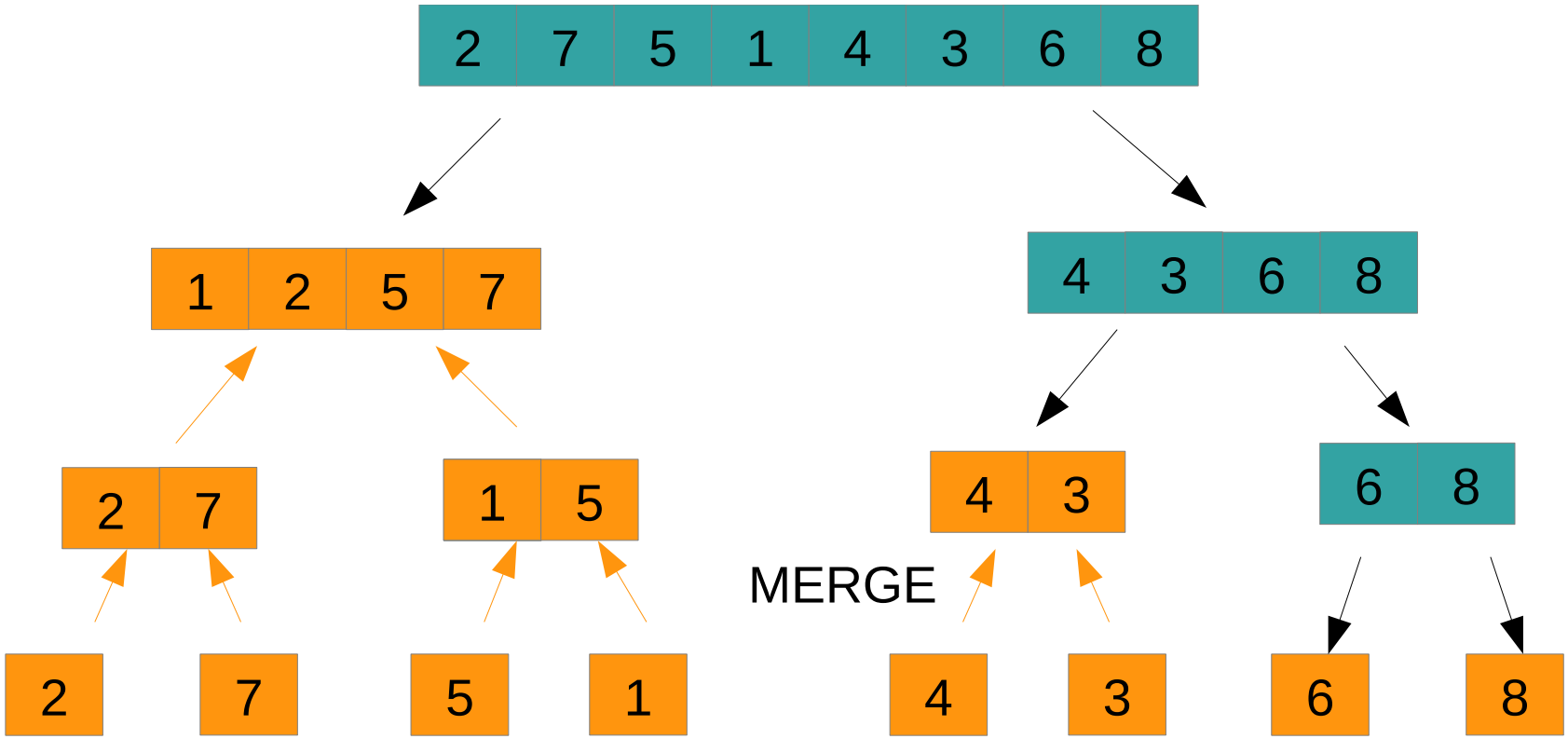
# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



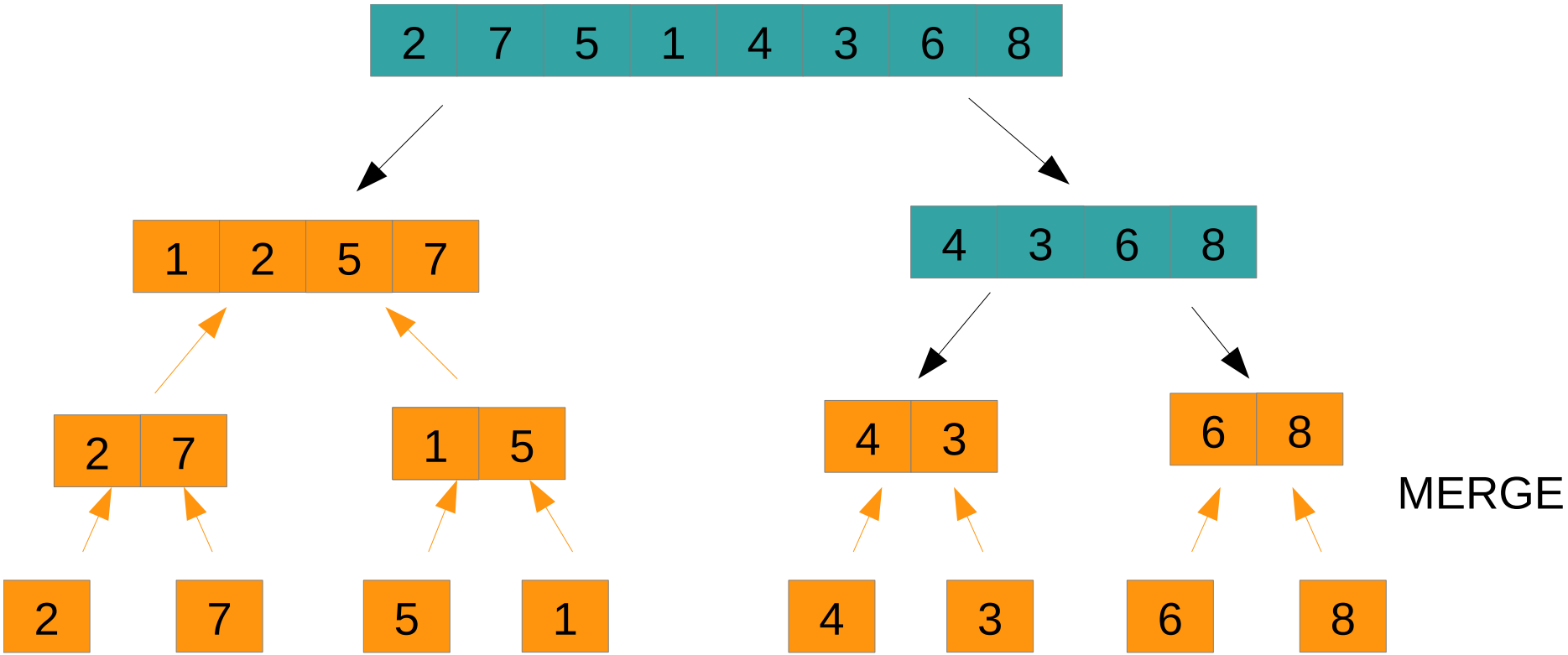
# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



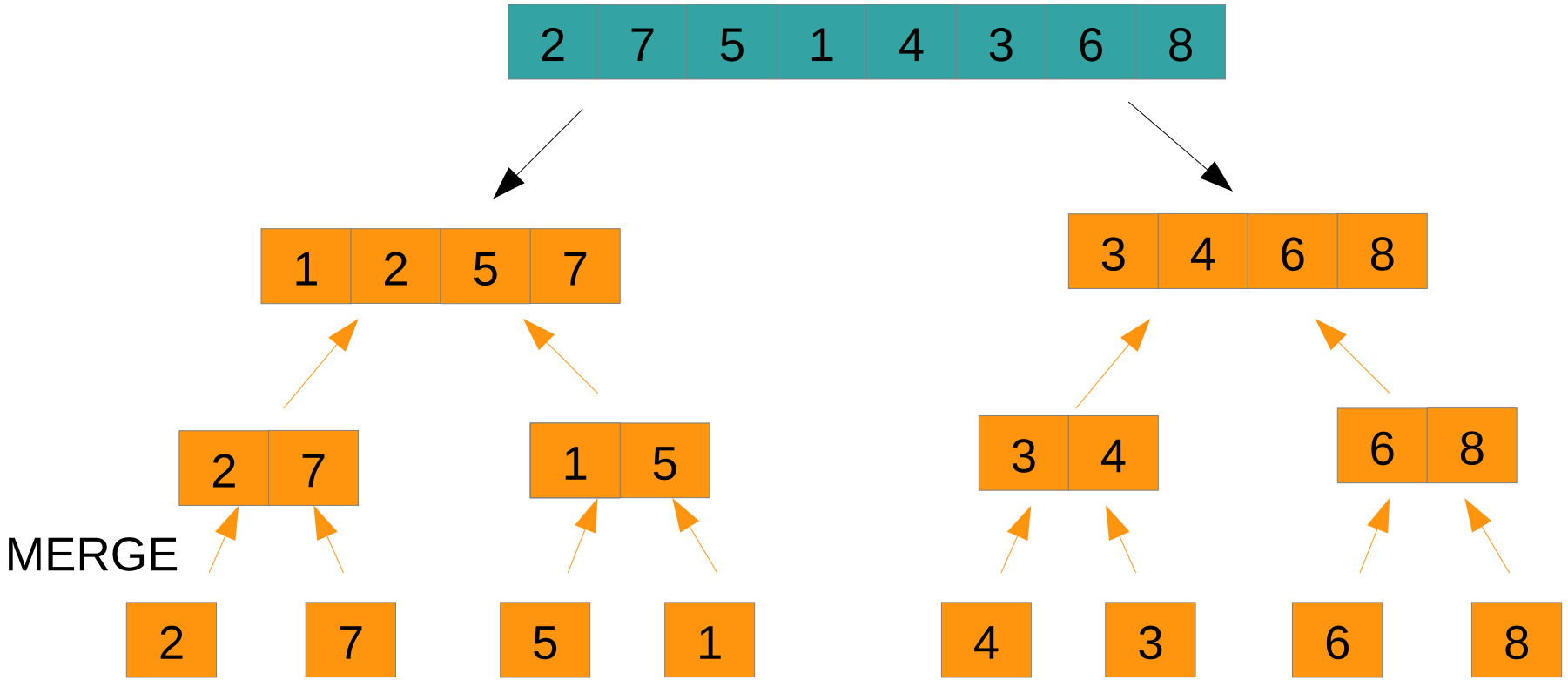
# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



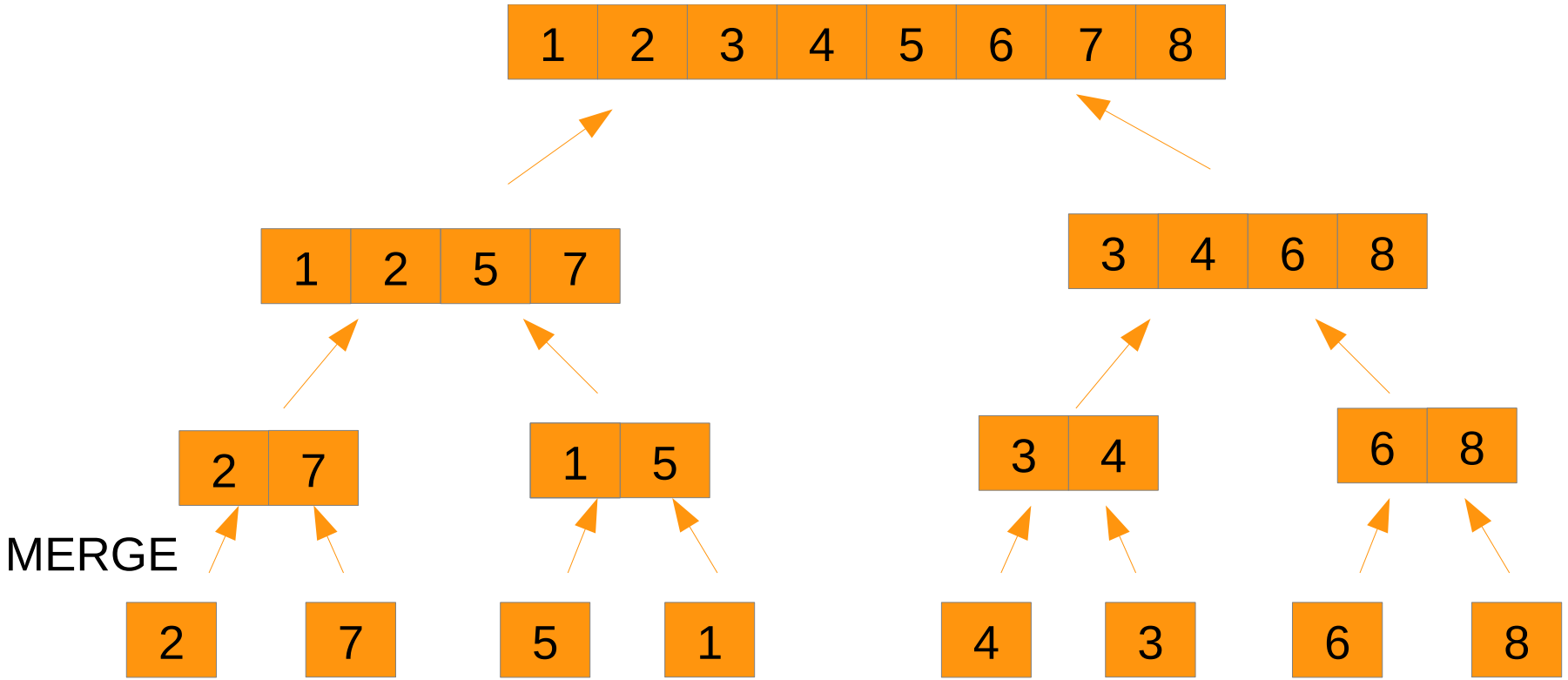
# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



# Ejemplo: Algoritmos de ordenación

Algoritmo 2: Merge sort



# Ejemplo: Algoritmos de ordenación

## Array Sorting Algorithms

| Algorithm | Time Complexity | | | Space Complexity |
|---|---|---|---|---|
| | Best | Average | Worst | Worst |
| Quicksort | Ω(n log(n)) | θ(n log(n)) | O(n^2) | O(log(n)) |
| Mergesort | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) | O(n) |
| Timsort | Ω(n) | θ(n log(n)) | O(n log(n)) | O(n) |
| Heapsort | Ω(n log(n)) | θ(n log(n)) | O(n log(n)) | O(1) |
| Bubble Sort | Ω(n) | θ(n^2) | O(n^2) | O(1) |
| Insertion Sort | Ω(n) | θ(n^2) | O(n^2) | O(1) |
| Selection Sort | Ω(n^2) | θ(n^2) | O(n^2) | O(1) |
| Tree Sort | Ω(n log(n)) | θ(n log(n)) | O(n^2) | O(n) |
| Shell Sort | Ω(n log(n)) | θ(n(log(n))^2) | O(n(log(n))^2) | O(1) |
| Bucket Sort | Ω(n+k) | θ(n+k) | O(n^2) | O(n) |
| Radix Sort | Ω(nk) | θ(nk) | O(nk) | O(n+k) |
| Counting Sort | Ω(n+k) | θ(n+k) | O(n+k) | O(k) |
| Cubesort | Ω(n) | θ(n log(n)) | O(n log(n)) | O(n) |

WTPC Workshop en técnicas de programación científica

# Ejemplo: Algoritmos de ordenación

5 Sorting Algorithms en 6 minutos
http://youtube/kPRA0W1kECg

# Cuanto importa la elección de un buen algoritmo

Algoritmo de transformada de Fourier

Costo TDF $O(N^2)$ $\rightarrow$Costo FFT $O(N \log_2 N)$

*J. W. Cooley,J. W. Tukey (1965)*

¿ Cual es realmente le beneficio?

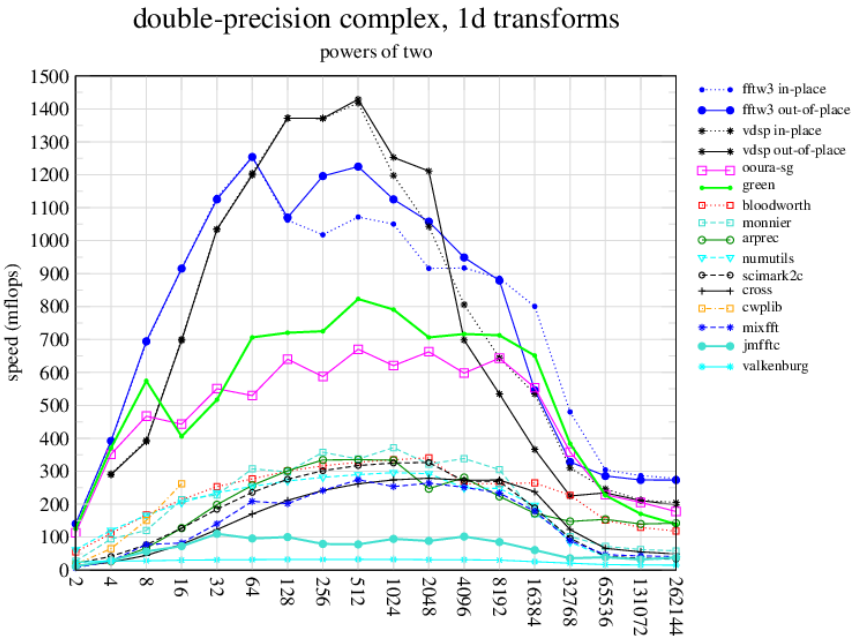
| **N** | 1000 | $10^6$ | $10^9$ |
|---|---|---|---|
| $O(N^2)$ | $10^6$ | $10^{12}$ | $10^{18}$ |
| $O(N \log_2 N)$ | $10^4$ | $20 \times 10^6$ | $30 \times 10^9$ |

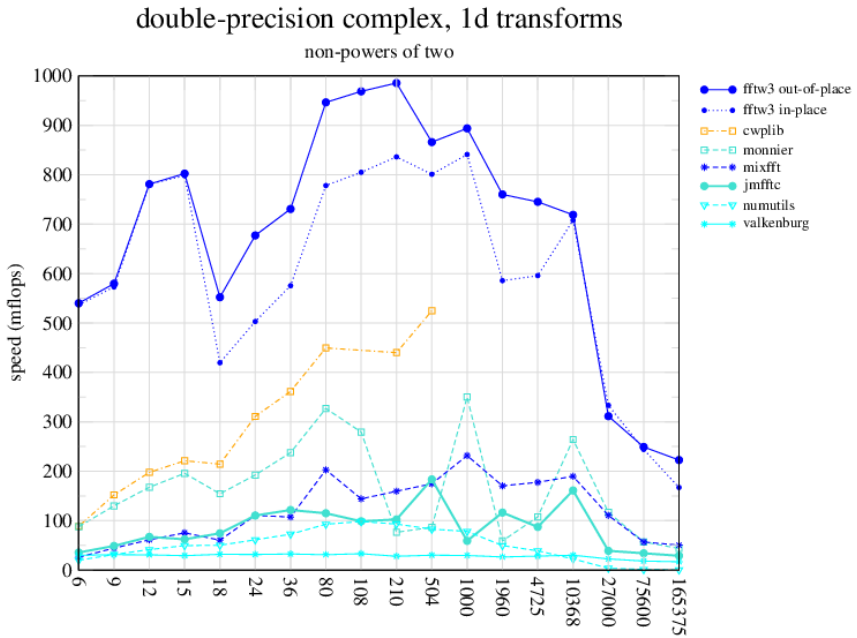
Supongamos que cada operación demora 1 *ns*

$10^{18}$ ns 31.2 años

$30 \times 10^9$ ns 30 seg

# Cuanto importa la elección de un buen algoritmo

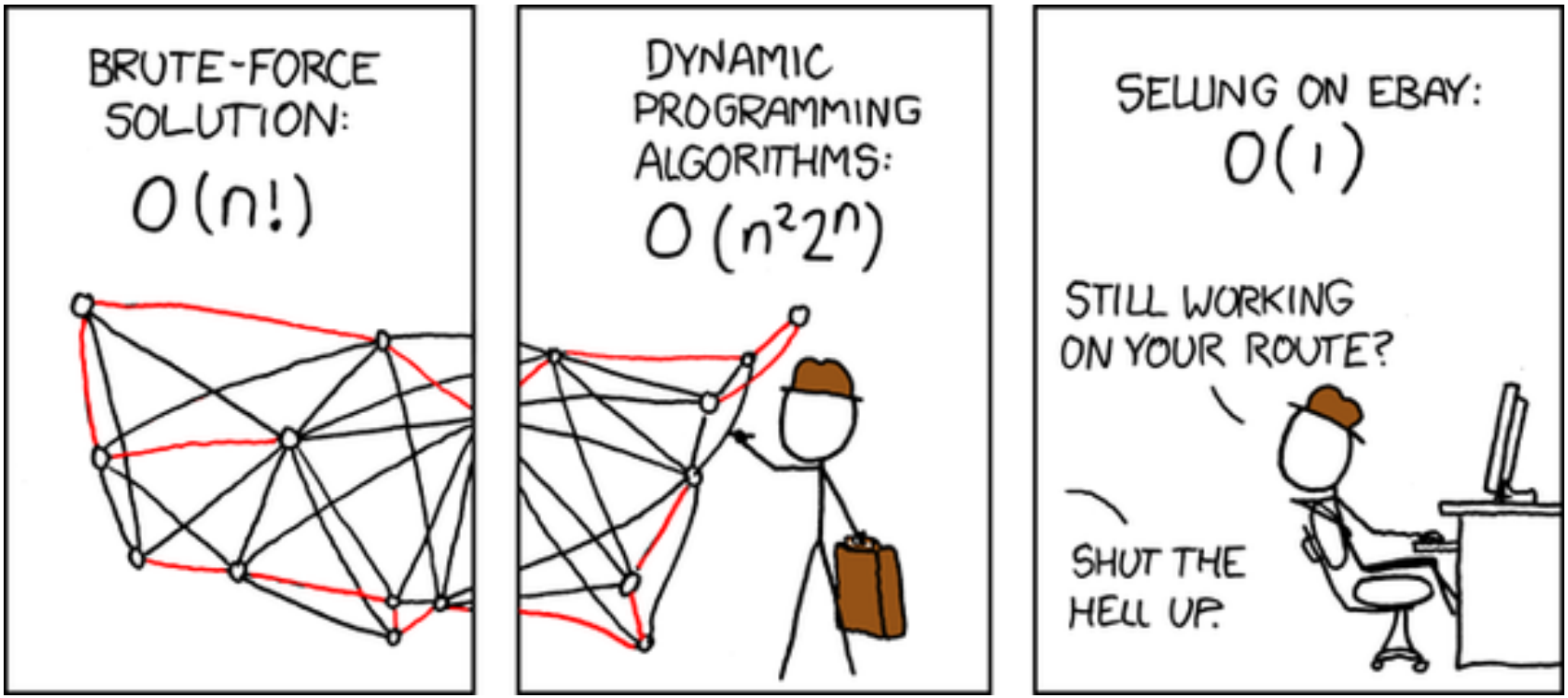



http://www.fftw.org/speed/G4-1.06GHz-macosx/

# Cuanto importa la elección de un buen algoritmo

Encontrar un mejor algoritmo es mejor que optimizar un algoritmo!



# Optimización respecto a las estructuras de datos

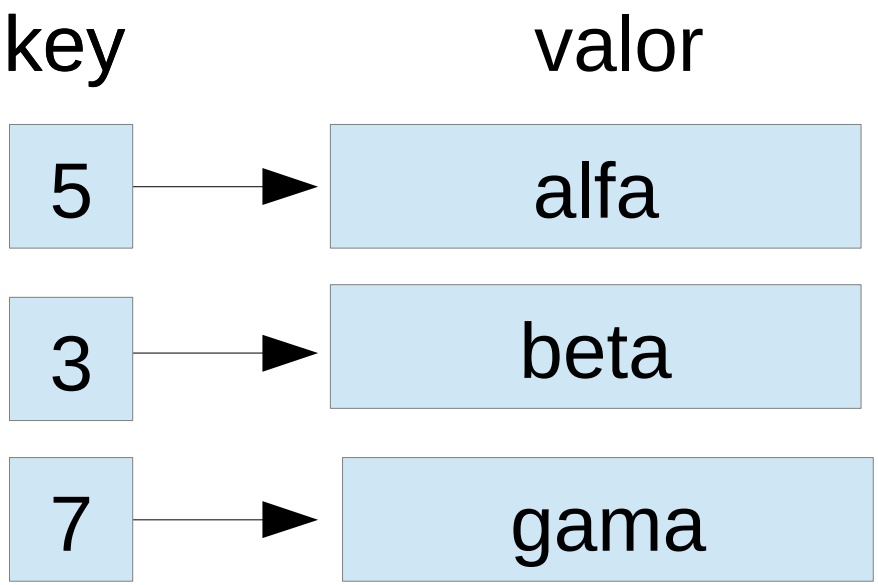
- Almacenar información de forma organizada y estructurada y no como datos simples.
- Se pueden ver como una colección de datos que se caracterizan por su organización y las operaciones que se definen en ellos
- Objetivo: tener un fácil acceso y manejo de datos

# Estructuras de datos

- Secuenciales: Datos organizados consecutivamente.

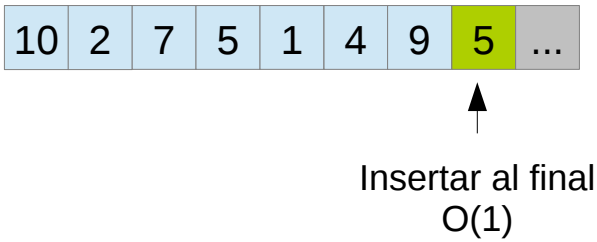
| 10 | 2 | 7 | 5 | 1 | 4 | 9 |
|---|---|---|---|---|---|---|

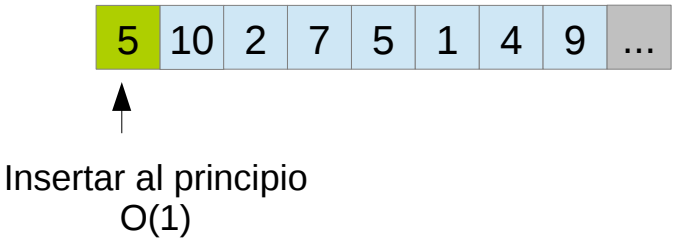
- Asociativas: Los datos no tienen por qué situarse de forma contigua sino que se localizan mediante una clave.



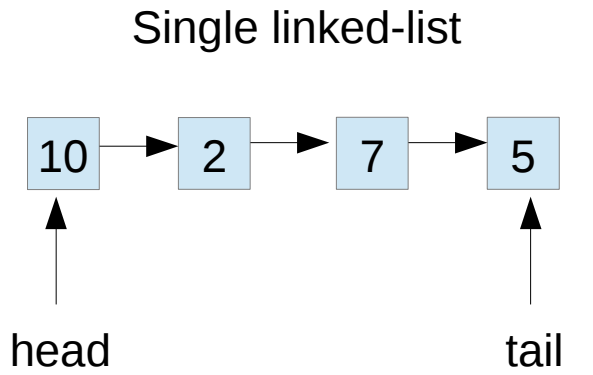
# Contendores secuenciales

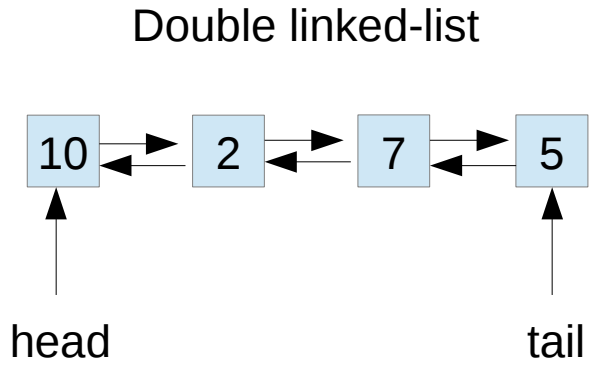
## Arreglos



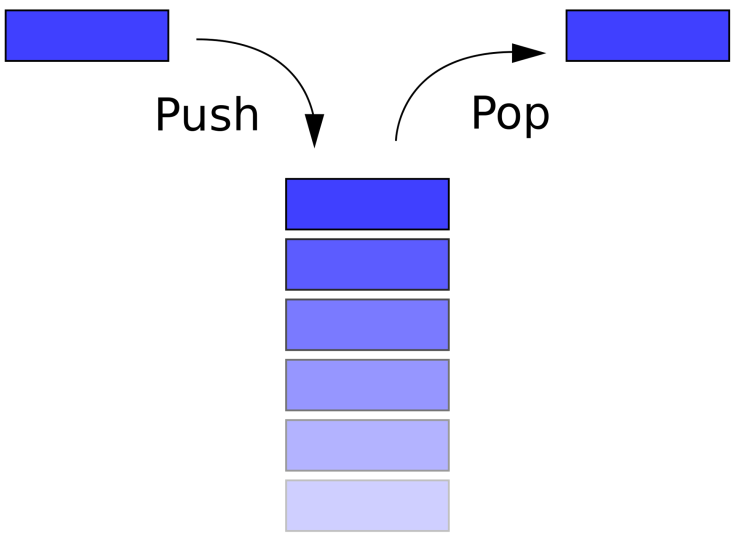

## Listas enlazadas



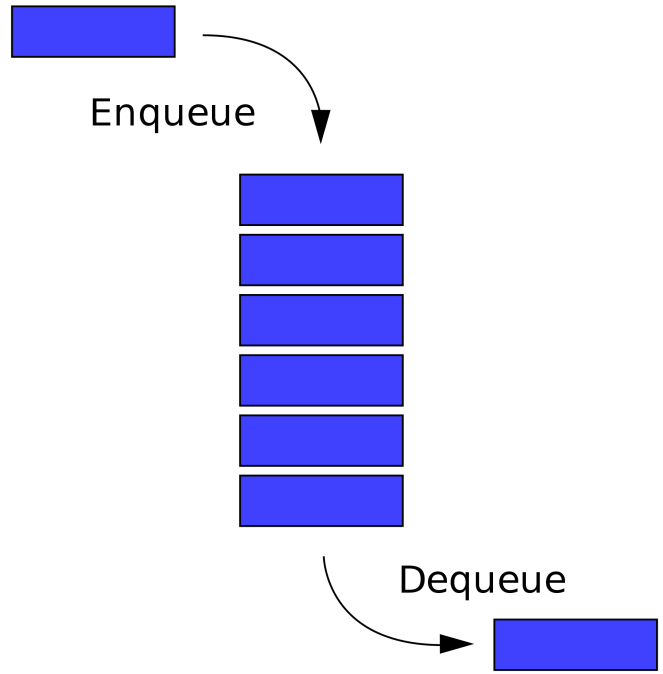



# Contendores secuenciales

Cola o LIFO (last in first out)

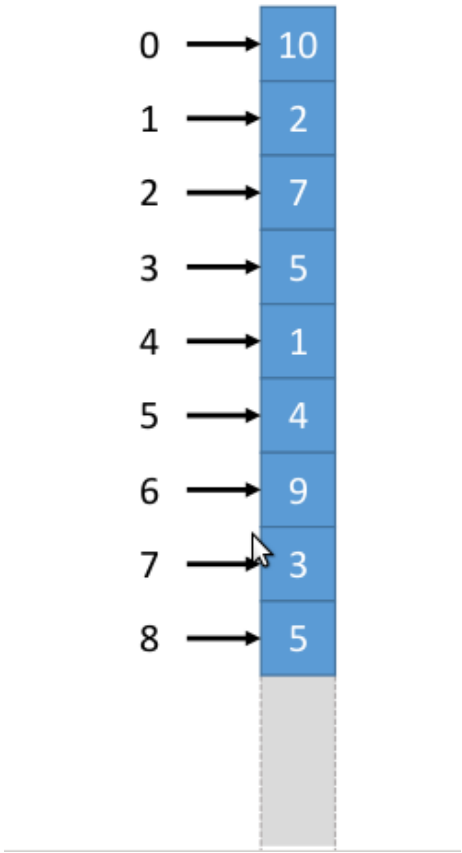



Fila o FIFO (first in first out)

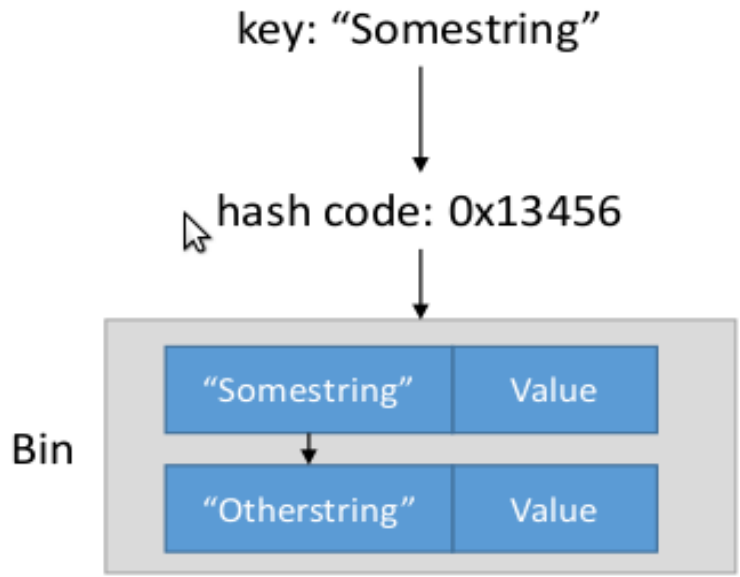


# Contendores asociativos

## Arreglos




## Mapas no ordenados (hash maps)




+ diccionarios, mapas ordenados, conjuntos

# Optimización respecto a las estucturas de datos

**Qué tanto impacta la elección de las estructuras de datos en nuestro programa?**

EJEMPLO
Supongamos que queremos programar un jueguito (Follow me):

- Se genera una secuencia aleatorea de números que se van mostrando por pantalla.
- El jugador debe reproducir la secuencia
- Cada coincidencia en el orden de aparición es un punto

# Optimización respecto a las estucturas de datos

El algoritmo debe realizar las siguientes tareas:

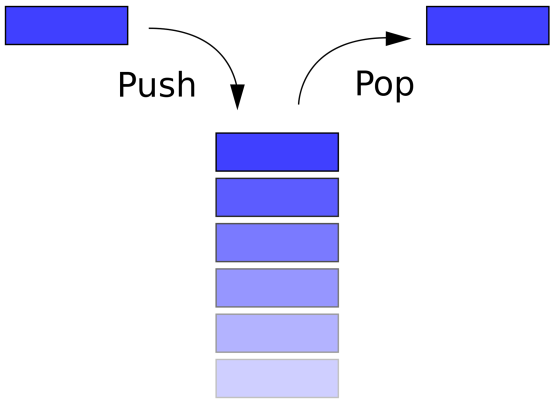
- generar, almacenar y mostrar por pantalla una secuencia
- capturar y almacenar la secuenca ingresada por el jugador
- Comparar las dos secuencias y establecer el puntaje

WTPC workshop en técnicas de programación científica

# Optimización respecto a las estucturas de datos

PRIMERA IDEA: Usar una estructura pila (LIFO)

Veamos una implementación con lista enlazada en C




```
1  #include <stdio.h>
2
3  typedef int item;
4  const item indefinido=9999;
5
6  struct nodo{
7        item dato;
8              nodo *sig;
9         };
10
11 typedef nodo* Pila;
12
13 Pila pilaVacia();
14 int esPilaVacia(Pila l);
15 Pila push(Pila l, char x);
16 Pila pop(Pila l);
17 item top(Pila l);
18 void escribir(Pila l);
```

# Optimización respecto a las estucturas de datos

Cuál es el costo de cada operación?

```
1  int esPilaVacia(Pila l){
2    return l==NULL;}
3
4  Pila push(Pila l, item x){
5    nodo *aux;
6    aux=new nodo;
7    aux->dato=x;
8    aux->sig=l;
9          l=aux;
10         return l;}
11
12 item top(Pila l){
13   if(!esPilaVacia(l))
14     return l->dato;
15   else
16     return indefinido;}
```

```
1
2  Pila pop(Pila l){
3    if(!esPilaVacia(l)){
4      nodo * aux=l;
5      l=aux->sig;
6      delete aux;
7    }
8    return l;}
9
10 void escribir(Pila l){
11   printf("\n PILA ->");
12   while(!esPilaVacia(l)){
13     printf(" %d ->",l->dato);
14                   l=l->sig;}
15   printf(" FIN\n");
16 }
```

# Optimización respecto a las estucturas de datos

Volviendo al problema:ALGORITMO (pseudocódigo)

```
WHILE not EndOfSequence
    item=aleat(N)
    p1=push(p,item)
    display(item)
END
WHILE not EndOfSequence
    item=GetItem()
    p2=push(p2,item)
    display(item)
END
```

Ahora tendríamos que comparar p1 y p2

Cuál sería el costo?

# Optimización respecto a las estucturas de datos

PRIMERA IDEA: Usar una estructura pila (LIFO)

Caso1:
pc: [1,**2**]
player:[] (pila vacia)
player: player +0 ptos
Caso2:
pc: [1,**2**]
player: [1,**2**]
Listas equivalentes

Caso 3:
pc: [1,2,**2**]
player: [1,2,**3**]
Caso 4:
pc: [1,2,3,**4**]
player: [1,2,**3**]
Player +3 ptos

**Se puede mejorar?**
**Era esta la estructura de datos mas acertada para el problema?**

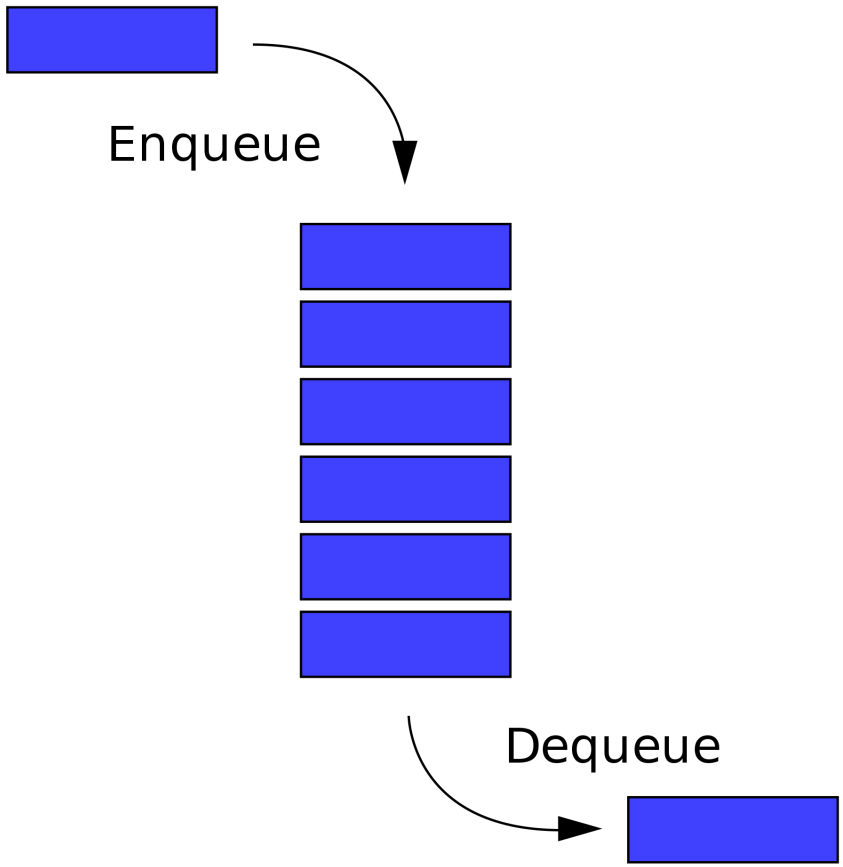
WTPC workshop en técnicas de programación científica

# Optimización respecto a las estucturas de datos

PENSEMOS OTRA SOLUCION: Usar una estructura lista (FIFO)

**Como se realizaria entonces la comparación?**

Recordemos



# Optimización respecto a las estucturas de datos

Entonces

Caso1:
pc: [**1**,2]
player:[] (fila vacia)
player: player +0 ptos
Caso2:
pc: [**1**,2]
player: [**1**,2]
Listas equivalentes

Caso 3:
pc: [**1**,2,2]
player: [**1**,2,3]
Caso 4:
pc: [**1**,2,3,4]
player: [**1**,2,3]
Player +3 ptos

**Cuanto mejora?**

WTPC workshop en técnicas de programación científica

# Optimización respecto al hardware

Cómo hacer más rápido nuestros cálculos?

# Optimización respecto al hardware

Cómo hacer más rápido nuestros cálculos?

- Escribir y leer más rápido (I/O ¿ veloc)
- Dar vueltas más rápido la manija (¿ veloc clock)
- Mejorar la tecnología (mejor CPU)
- Limitar el acceso a memoria no-uniforme y multi-core
- Arquitecturas multi-cores. Paralelización
- Compiladores que permiten optimización para aprovechar la CPU (pipelining, unidades vectoriales y superescalares , etc)

WTPC workshop en técnicas de programación científica

# Optimización respecto al hardware

El tipo de operaciones importa!!!

- + - *
- / % sqrt()
- Func. trascendentales
- Pow(x,y) x,y reales

- Op. más veloces,0.5x-1x
- Op. velocidad media,5x-10x
- Op. lentas, 20x-50x
- Op. costosas, muy lentas,+100x

Pipelining →operaciones veloces.
Usar BLAS-LAPACK (álgebra lineal, son + y *)

WTPC workshop en técnicas de programación científica

# Técnicas de optimización (compilador)

Opt. Escalar

- Copy propagation
- Const folding
- Strength reduction
- Eliminación subexpresiones comunes
- Renombrado vbles

Opt. Lazos

- Loop invariant
- Loop unrolling
- Intercambio orden loop
- Fusion/fision loop

Inlining

- Reemplaza una porción de código por otro equivalente + veloz)

WTPC workshop en técnicas de programación científica

# Copy propagation

Antes:

**x**=y

z=1+**x**

↓

Después:

x=y

z=1+Y

El compilador elimina la dependencia (en caso de ser posible)

# Const folding

Antes
a=100;
b=200;
sum=a+b;
↓
Después:
sum=300;

El compilador pre-calcula el resultado una única vez en tpo de compilación. Elimina código redundante.

# Strength Reduction

Antes

```
x=pow(y,2.0);
a=b/2.0;
```

↓

Después:

```
x=y*y;
a=b*0.5;
```

Pow, / son las operaciones más costosas

Si el compilador puede saber que se están realizando la potencia de un num entero pequeño, o el denominador de una división es una constante; entonces puede reemplazar estas operaciones por productos que es menos costoso

WTPC workshop en técnicas de programación científica

# Eliminación subexpresiones comunes

Antes

d=c*(a/b);

e=(a/b)*2.0;

↓

Después:

x=a/b;

d=c*x;

e=x*2.0;

La idea es eliminar el cálculo doble de la /

Solo si la subexpresion es un cálculo costoso o si resulta en la reducción de registros a utilizar

# Renombrado de variables

Antes

**x**=y*z;

q=r+**x***2;

**x**=a+b;

↓

Después:

x0=y*z;

q=r+x0*2;

x=a+b;

El código original tiene una dependencia de salida, el segundo no (x sigue teniendo el mismo resultado final)

# Loop invariante

Antes

```
DO i=1,n
  a(i)=b(i)+c*d
  e=g(n)
END DO
```

↓

Después:

```
tempx=c*d;
Do i=1,n
  a(i)=b(i)+temp
END DO
e=g(n)
```

Loop invariant= código que no cambia dentro de un lazo.

No es necesario que se ejecute iterativamente

WTPC workshop en técnicas de programación científica

# Loop unrolling

Antes

```
DO i=1,n
 a(i)=a(i)+b(i)
END DO
```

↓

Después:

```
Do i=1,n,4
 a(i)=a(i)+b(i)
 a(i+1)=a(i+1)+b(i+1)
 a(i+2)=a(i+2)+b(i+2)
 a(i+3)=a(i+3)+b(i+3)
ENDO
```

X el compilador

# Intercambio orden loop

Antes

```
DO i=1,ni
  DO j=1,nj
    a(i,j)=b(i,j)
  END DO
END DO
```

↓

Después:

```
 DO j=1,nj
  DO i=1,ni
    a(i,j)=b(i,j)
  END DO
END DO
```

Depende : en Fortran (después), en C (antes)

WTPC workshop en técnicas de programación científica

# Loop fisión/fusión

Fisionar

```
DO i=1,n
  a(i)=b(i)+1
END DO
DO i=1,n
  c(i)=a(i)/2
END DO
DO i=1,n
  d(i)=1/c(i)
END DO
```

Fusionar

```
DO i=1,n
 a(i)=b(i)+1
 c(i)=a(i)/2
 d(i)=1/c(i)
END DO
```

WTPC workshop en técnicas de programación científica

# Innlining

Antes

```
DO i=1,n
  a(i)=f(i)
END DO
...

REAL FUNCTION f(x)
  f=x*3
END FUNCTION f
```

↓

Después:

```
DO i=1,n
  a(i)=i*3
END DO
```

Elimina overhead del llamado a la función.

En gral el compilador tiene esta funcionalidad para altos niveles de optimización

# Optimizaciones del compilador

https://gcc.gnu.org/onlinedocs/gcc-4.9.1/gcc/Optimize-Options.html

`-O2` turns on all optimization flags specified by `-O`. It also turns on the following optimization flags:

```
-fthread-jumps
-falign-functions  -falign-jumps
-falign-loops  -falign-labels
-fcaller-saves
-fcrossjumping
-fcse-follow-jumps  -fcse-skip-blocks
-fdelete-null-pointer-checks
-fdevirtualize -fdevirtualize-speculatively
-fexpensive-optimizations
-fgcse  -fgcse-lm
-fhoist-adjacent-loads
-finline-small-functions
-findirect-inlining
-fipa-sra
-fisolate-erroneous-paths-dereference
-foptimize-sibling-calls
-fpartial-inlining
-fpeephole2
-freorder-blocks  -freorder-functions
-frerun-cse-after-loop
-fsched-interblock  -fsched-spec
-fschedule-insns  -fschedule-insns2
-fstrict-aliasing -fstrict-overflow
-ftree-switch-conversion -ftree-tail-merge
-ftree-pre
-ftree-vrp
```

# Optimización

M. Graciela Molina

m.graciela.molina@gmail.com

WTPC workshop en técnicas de programación científica