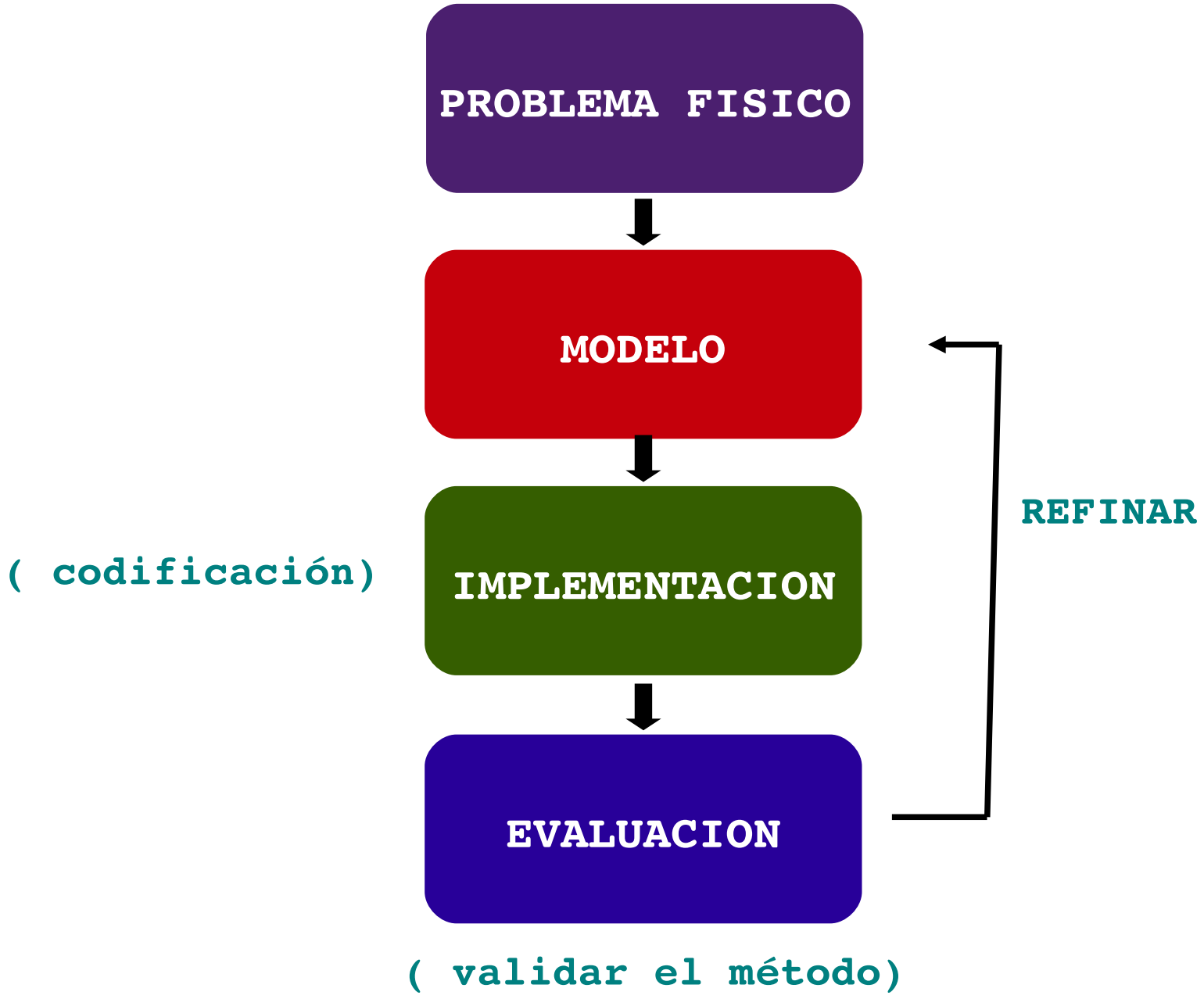
1-11 Marzo de 2017
FACET -UNT

OPTIMIZACION

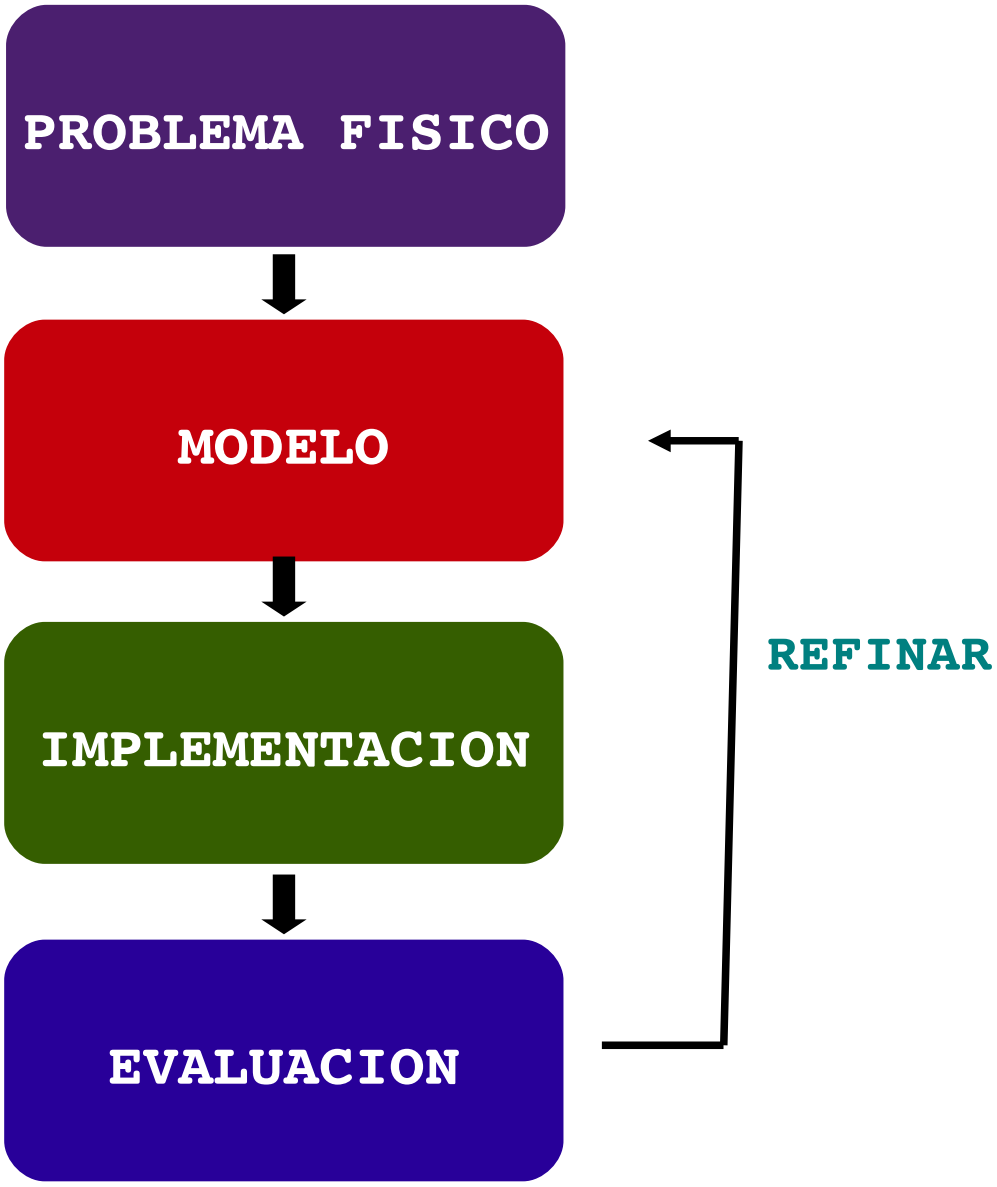
Graciela Molina

m.graciela.molina@gmail.com
gmolina@herrera.unt.edu.ar

Cómo es el proceso de escribir software científico?



Cómo es el proceso de escribir software científico?

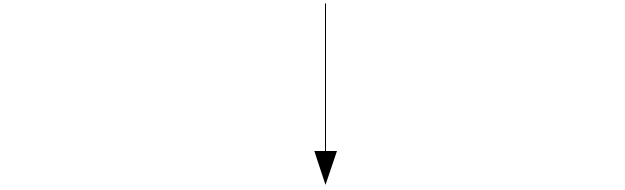


(codificación)

Un amigo de un amigo dice que ... nunca tengo que hacer XYZ porque el código se será más lento

(validar el método)

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



La primera versión que “anda”
(resultados correctos?!)

Pero cuando el problema crece ...

Recién notamos que no son soluciones óptimas

OPTIMIZAR !!!

QUE OPTIMIZAMOS ?

Tiempo de ejecución?

QUE OPTIMIZAMOS ?

Debemos pensar en lograr
un trade-off entre:

Tiempo de desarrollo
Debugging
Validación
portabilidad, etc.

Y claro el tiempo de
ejecución

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

COMO OPTIMIZAMOS ?

Seguramente alguien ya nos resolvió parte del problema !

No vamos a inventar la rueda

COMO OPTIMIZAMOS ?

USAR LIBRERIAS OPTIMIZADAS!!!

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/

Numpy

http://www.numpy.org/

Scipy

https://www.scipy.org/

Pandas

http://pandas.pydata.org/

QUÉ MÁS PODEMOS HACER ?

APRENDER SKILLS DE GOOGLEO!
(EVALUAR LO QUE EXISTE)

Seguramente conseguiremos sw de mejor calidad que si intentamos escribirlo nosotros mismos

Ojo! No siempre!

QUÉ MÁS PODEMOS HACER ?

Y SI TENGO QUE PROGRAMAR.
"ELEGIR" UN LENGUAJE DE PROGRAMACIÓN
(Tener presente qué existe, conocer sus fortalezas y debilidades)

HAY MÁS PARA OPTIMIZAR

Usar aproximaciones cuando sea posible.

Desarrollar **algoritmos más eficientes**

Utilizar **estructuras de datos apropiadas**

AÚN MÁS!

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

OPTIMIZACION

1. Respecto a los algoritmos
2. Respecto a las estructuras de datos
3. Respecto al hardware

OPTIMIZACION

1. Respecto a los algoritmos

2. Respecto a las estructuras de datos

3. Respecto al hardware

ALGORITMOS

¿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)

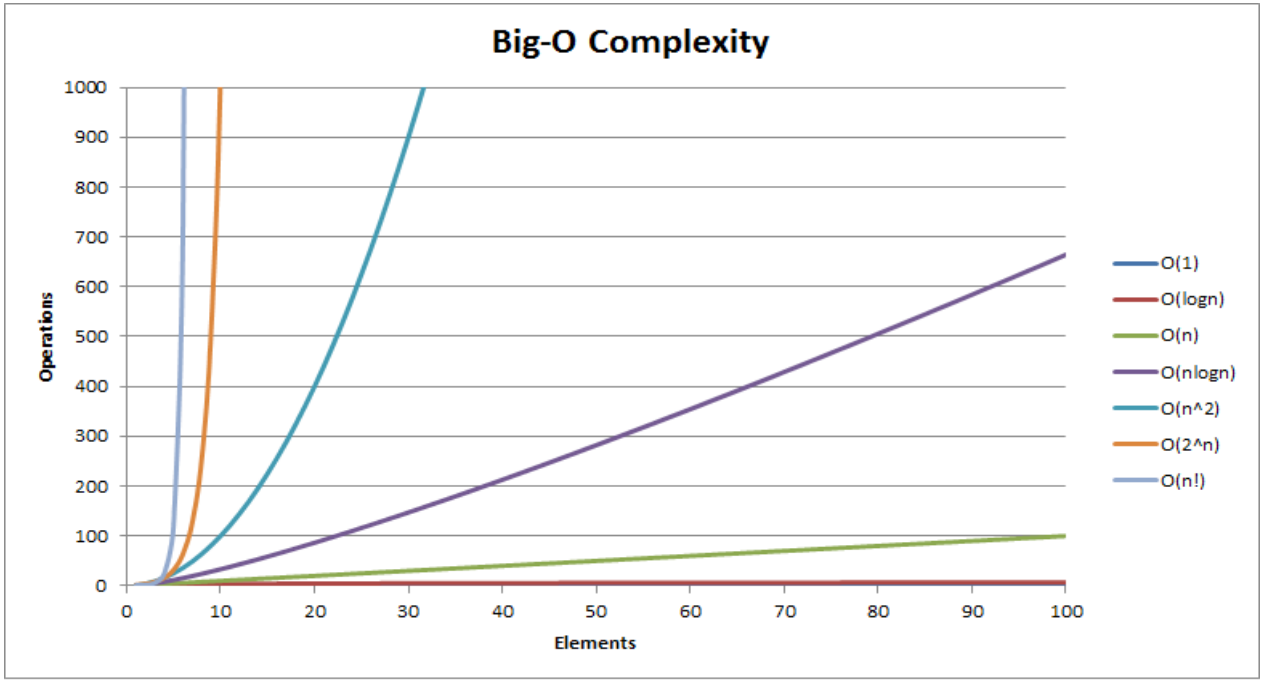
ALGORITMOS (evaluación teórica)

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

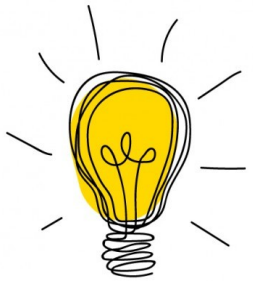


$O(1) < O(\log n) < O(n) < O(n^2) < O(n^3) < \ldots < O(c^n)$

ALGORITMOS

Ejemplo: Algoritmos de Ordenación

[9 0 0 4 3 4 1 2] → [0 0 1 2 3 4 4 9]



[9 0 0 8 3 4 1 2] → [**0** **9** 0 8 3 4 1 2]

9>0?

[0 **0** **9** 8 3 4 1 2]

[0 0 **8** **9** 3 4 1 2]

...

ALGORITMOS

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

Necesitamos ahora comparar los n-1 elementos restantes

Mejor caso: el arreglo ya esta ordenado.

Peor caso: el arreglo esta en el orden inverso.

$O(n^2)$

ALGORITMOS

¿ Como repensar el algoritmo?

Si mantenemos un contador de la cantidad de veces que se mueve:

Mejor caso: el arreglo ya esta ordenado y al finalizar el contador esta en cero y el algoritmo termina. ¿Qué ganamos?

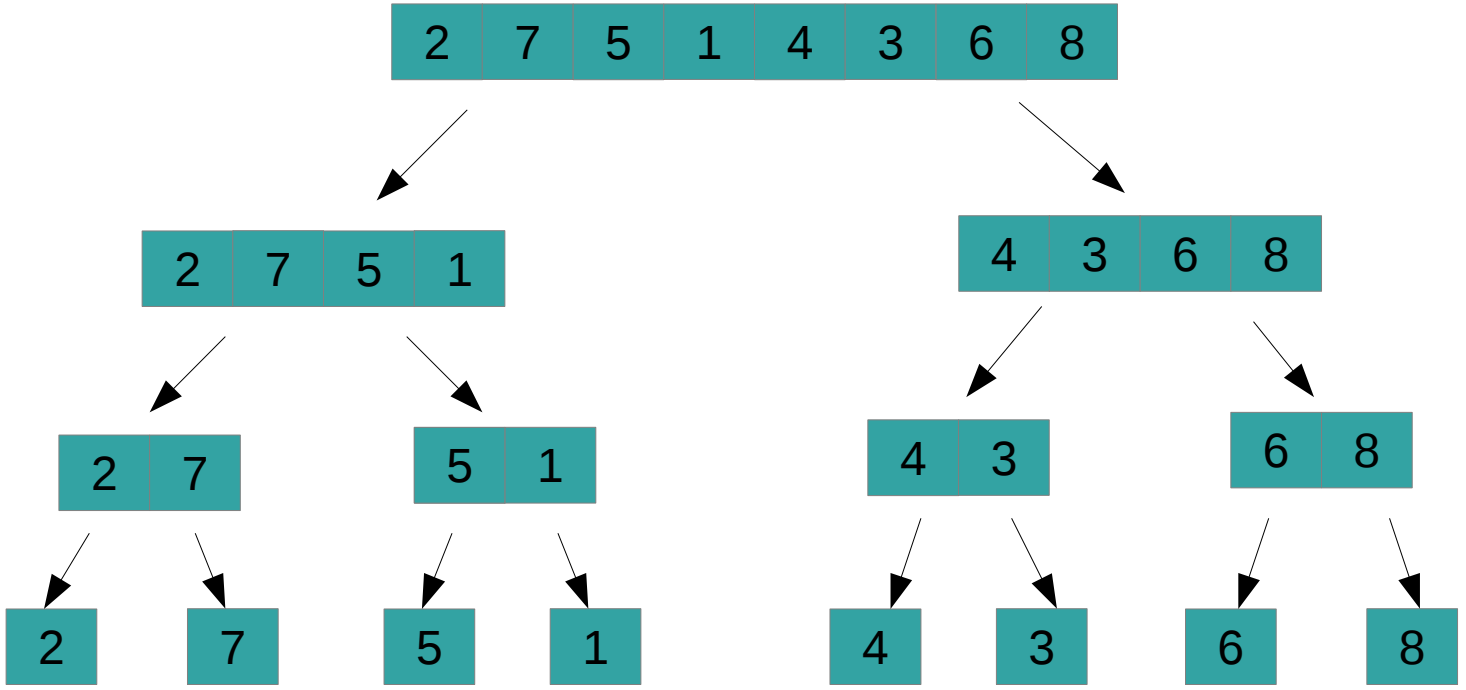
O(n)

Peor caso: No hay nada que hacer!

NO ALCANZA!!! A SEGUIR PENSANDO ...

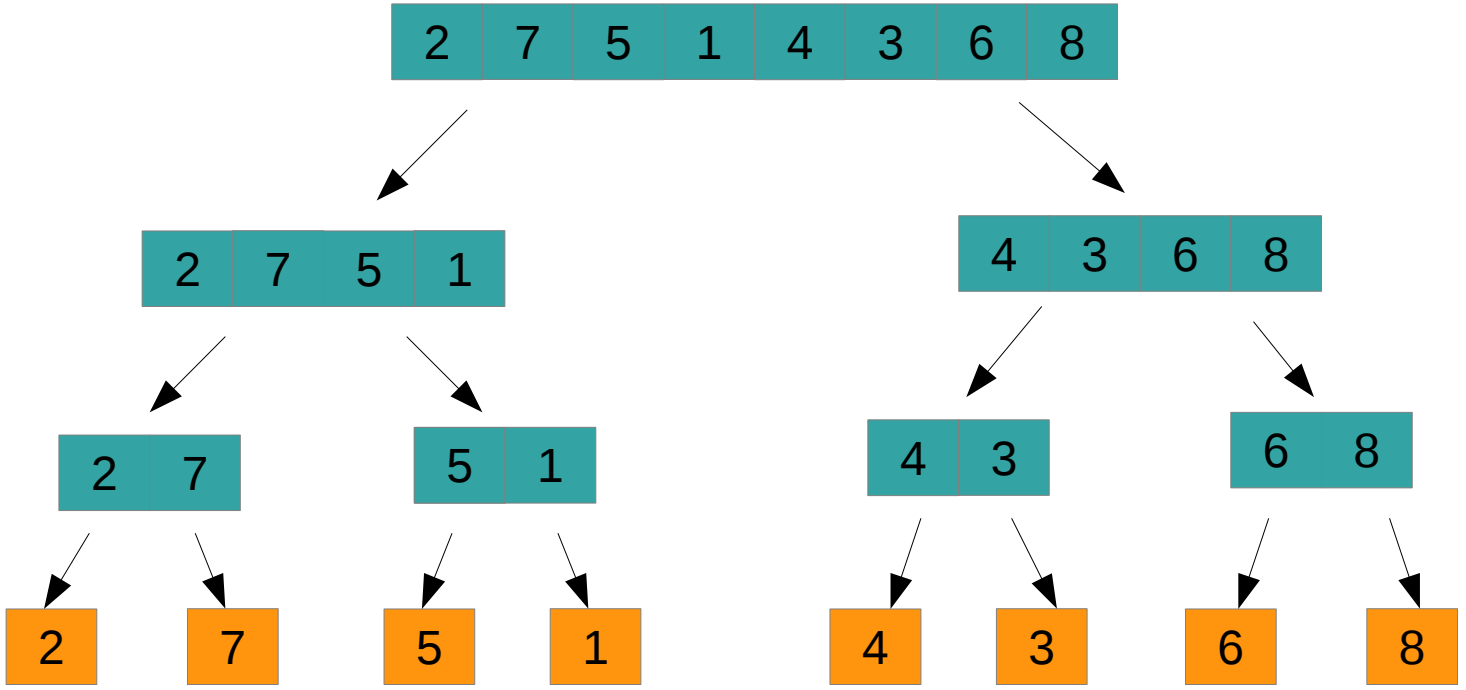
ALGORITMOS

EJEMPLO 2: Merge Sort



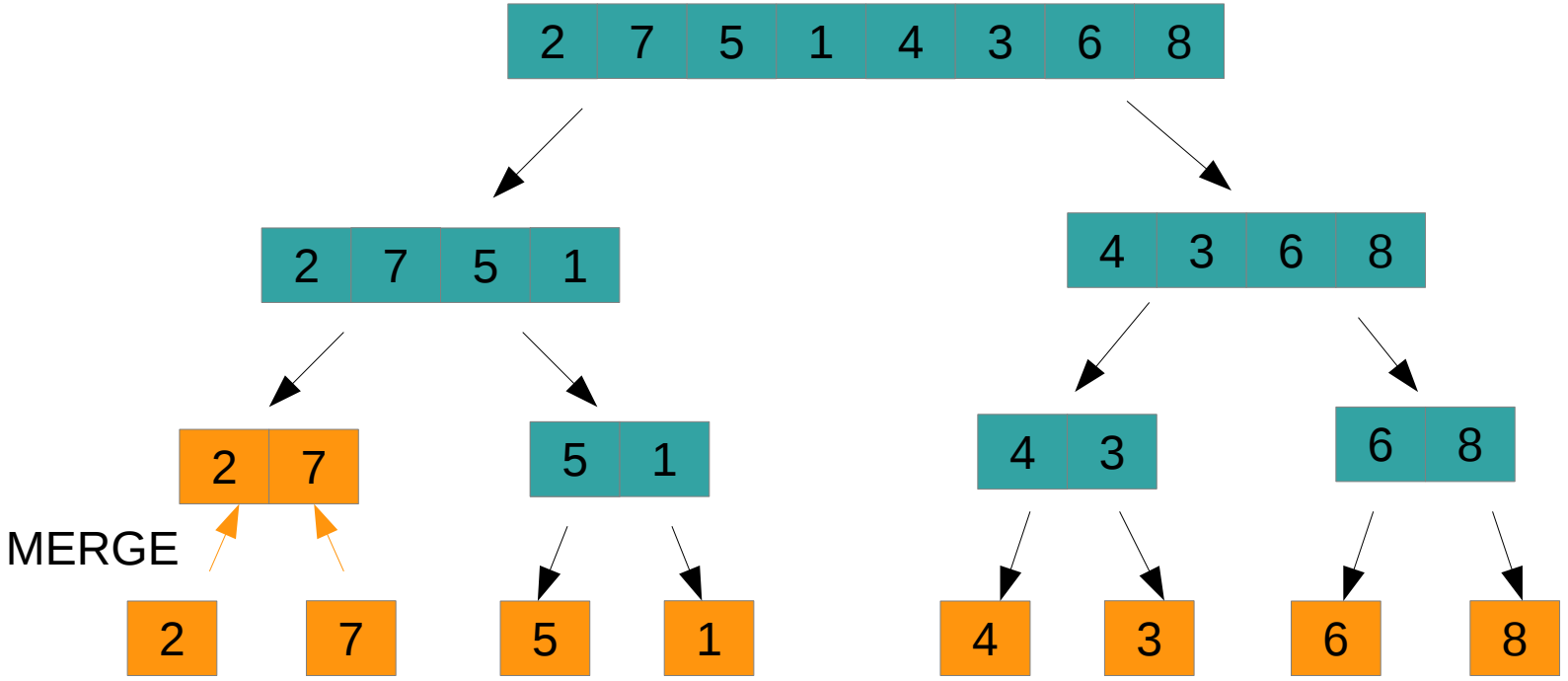
ALGORITMOS

EJEMPLO 2: Merge Sort



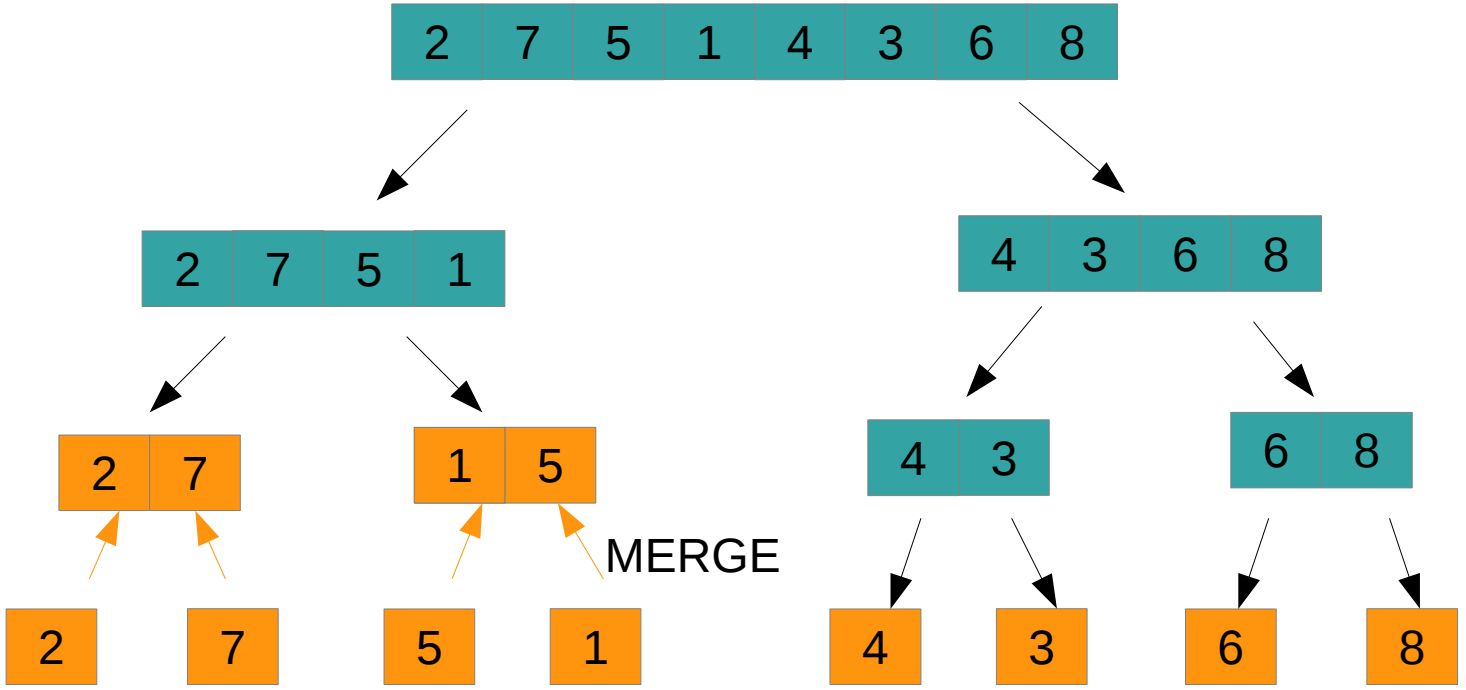
ALGORITMOS

EJEMPLO 2: Merge Sort



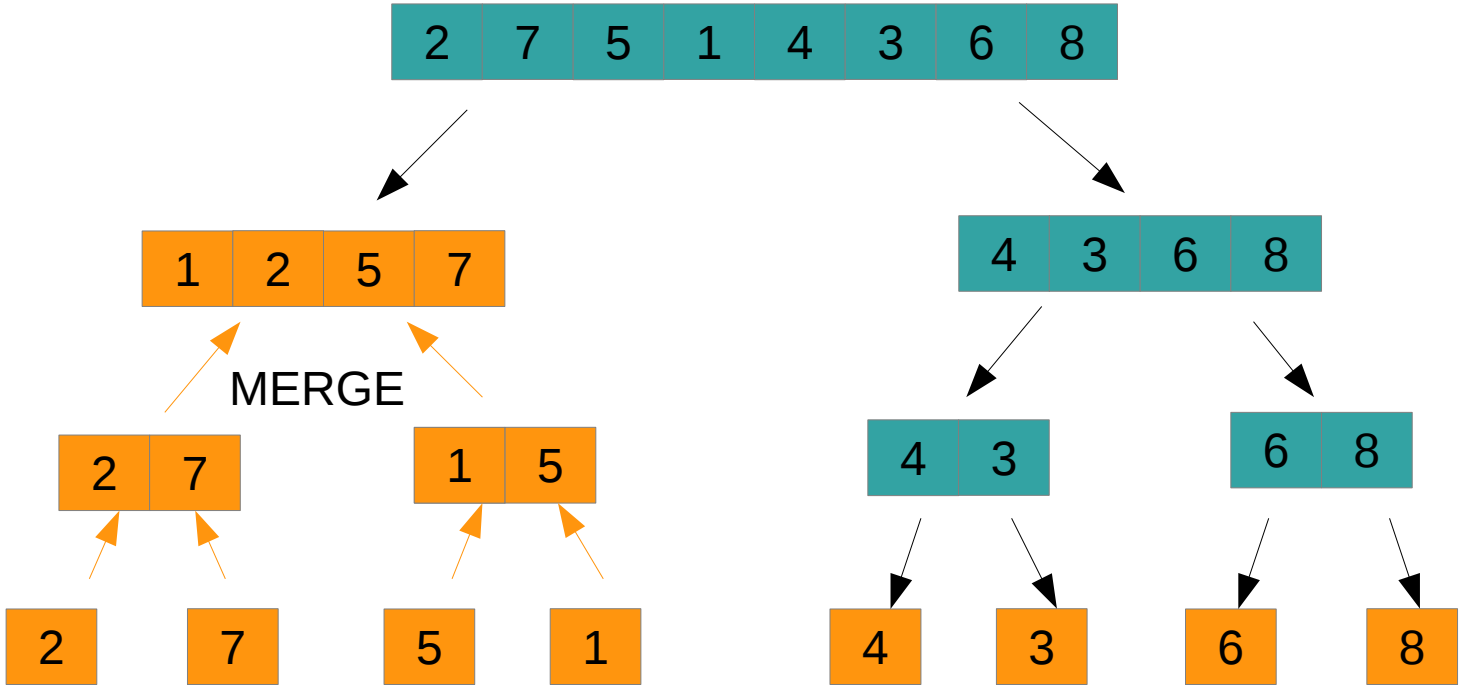
ALGORITMOS

EJEMPLO 2: Merge Sort



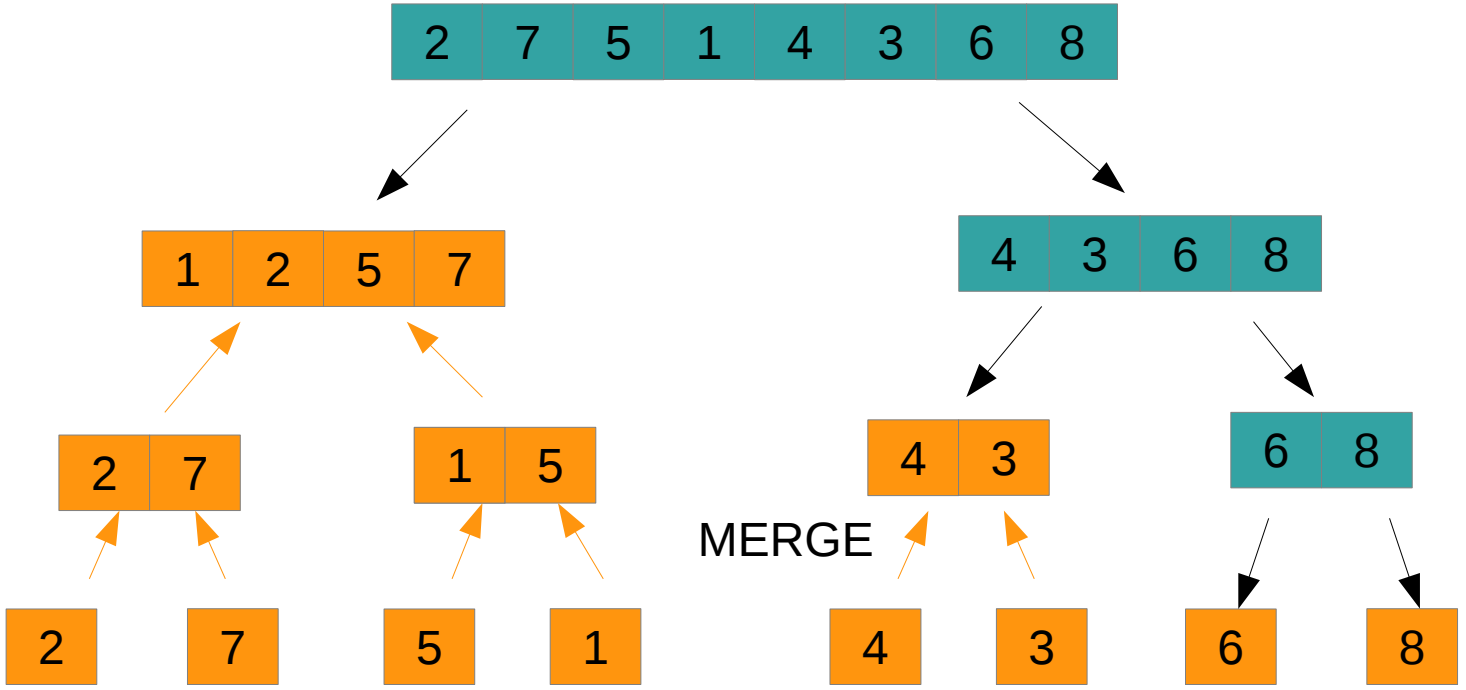
ALGORITMOS

EJEMPLO 2: Merge Sort



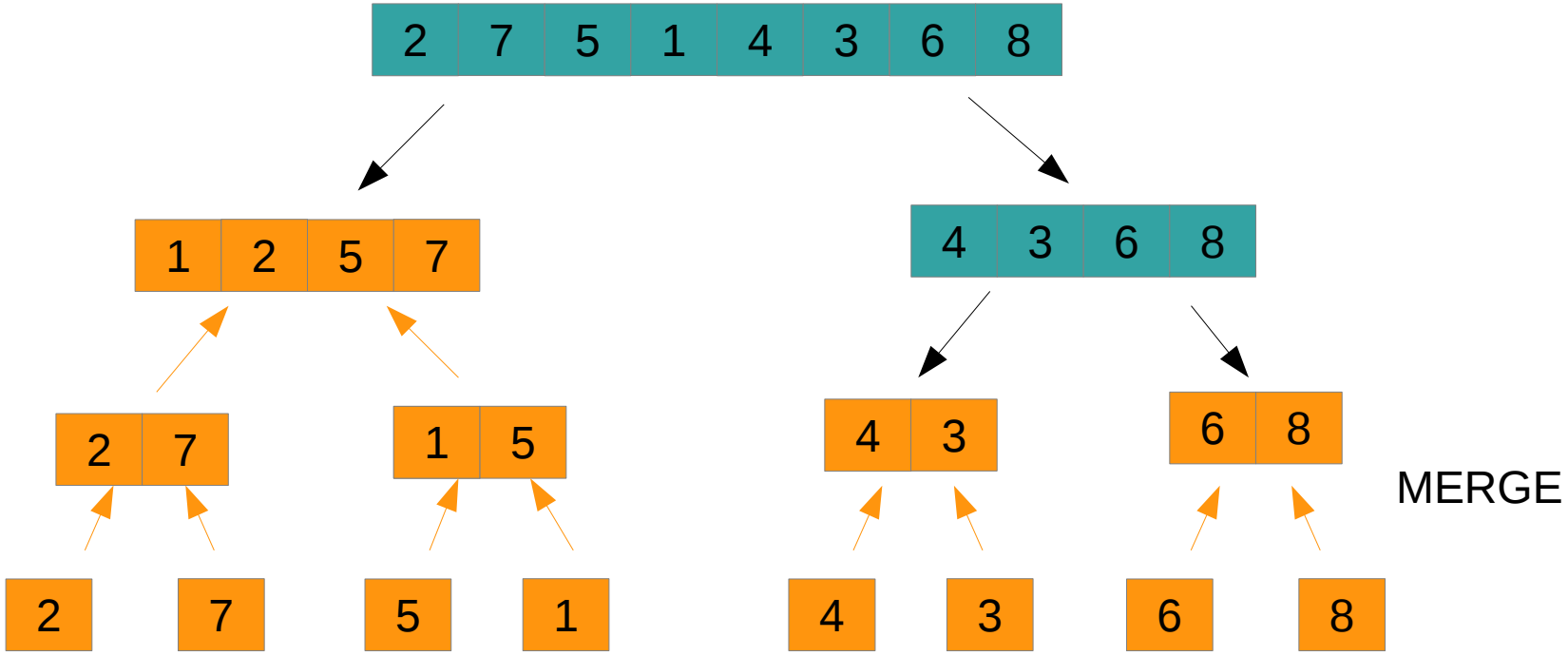
ALGORITMOS

EJEMPLO 2: Merge Sort



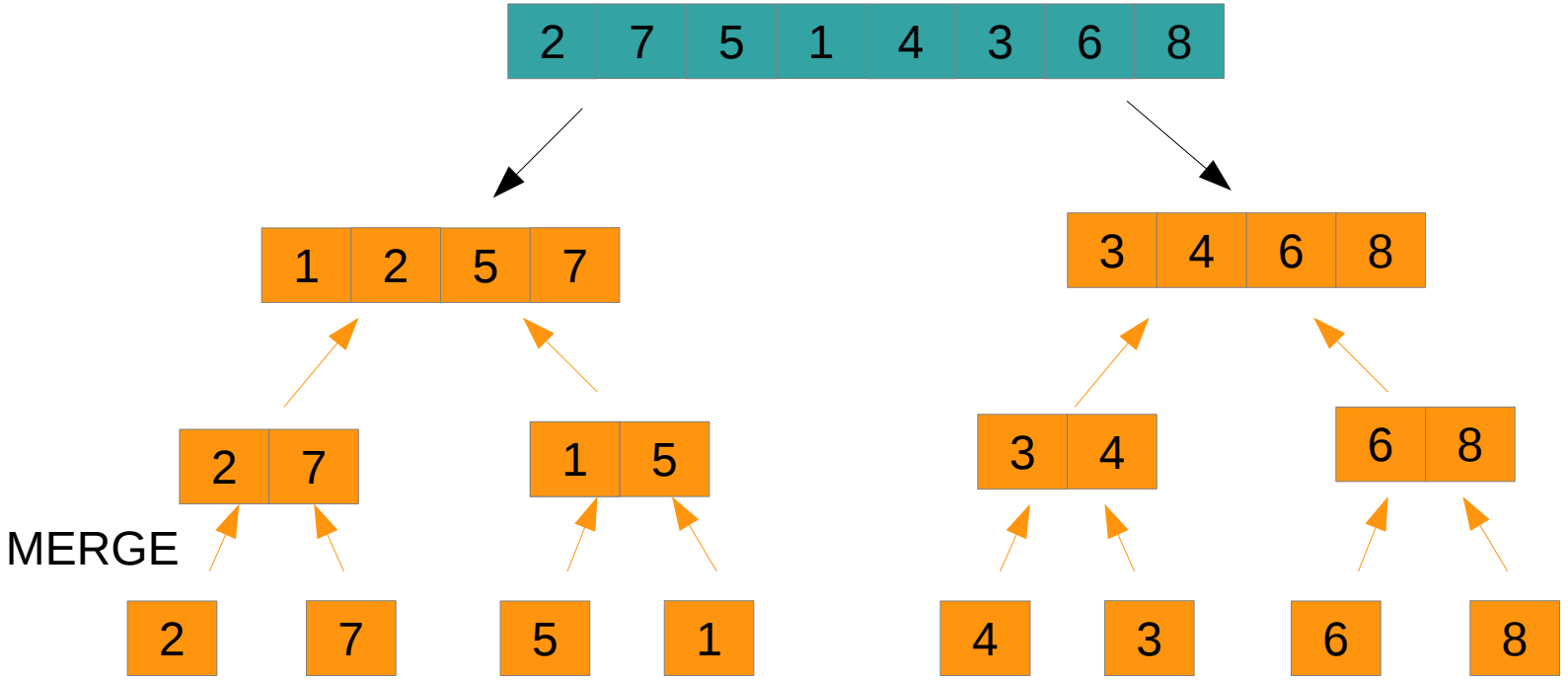
ALGORITMOS

EJEMPLO 2: Merge Sort



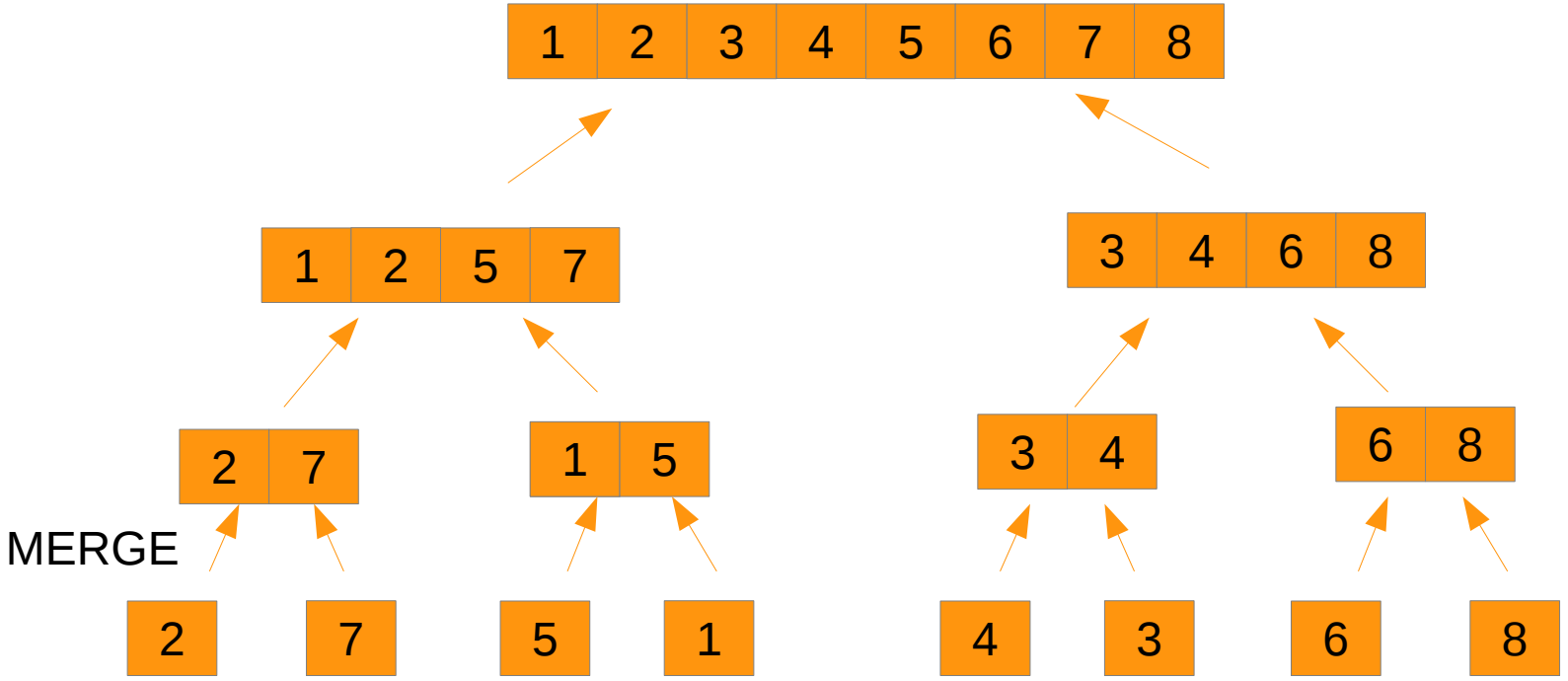
ALGORITMOS

EJEMPLO 2: Merge Sort



ALGORITMOS

EJEMPLO 2: Merge Sort



Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	O(n log(n))	O(n log(n))	O(n^2)	O(log(n))
Mergesort	O(n log(n))	O(n log(n))	O(n log(n))	O(n)
Timsort	O(n)	O(n log(n))	O(n log(n))	O(n)
Heapsort	O(n log(n))	O(n log(n))	O(n log(n))	O(1)
Bubble Sort	O(n)	O(n^2)	O(n^2)	O(1)
Insertion Sort	O(n)	O(n^2)	O(n^2)	O(1)
Selection Sort	O(n^2)	O(n^2)	O(n^2)	O(1)
Shell Sort	O(n)	O((nlog(n))^2)	O((nlog(n))^2)	O(1)
Bucket Sort	O(n+k)	O(n+k)	O(n^2)	O(n)
Radix Sort	O(nk)	O(nk)	O(nk)	O(n+k)

http://bigocheatsheet.com/

ALGORITMOS

¿ Qué tanto importa la elección de un algoritmo?

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

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

¿ Cual es realmente le beneficio?

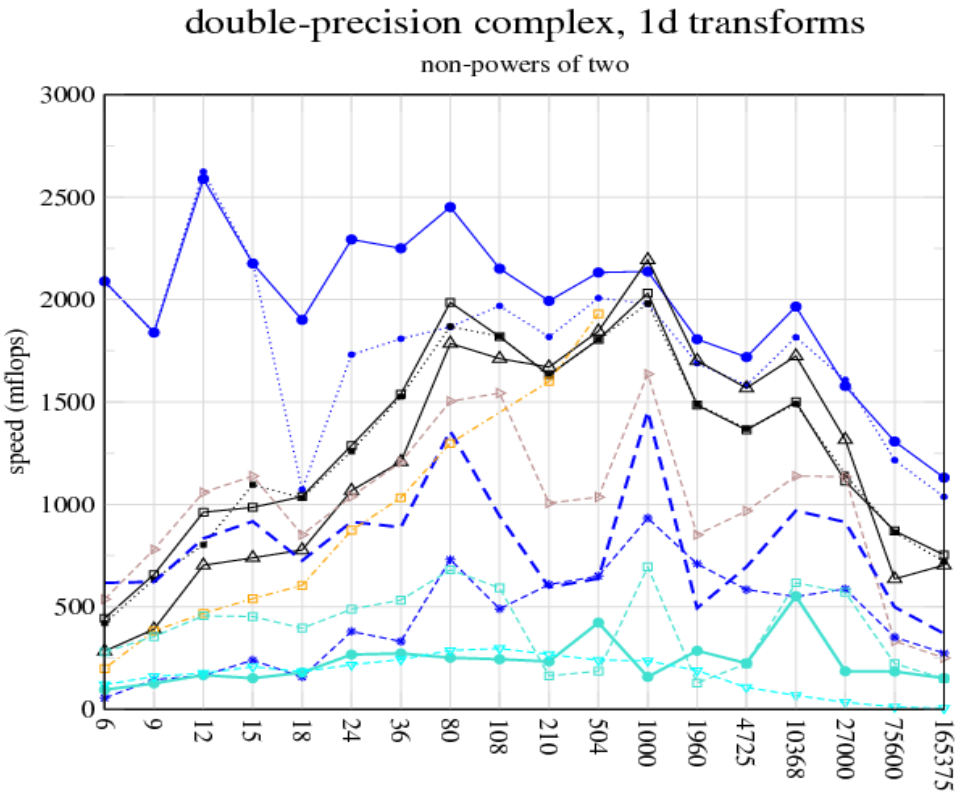
N	**1000**	**10^6**	**10^9**
N^2	10^6	10^{12}	10^{18}
$N\log_2 N$	10^4	$20x10^6$	$30x10^9$

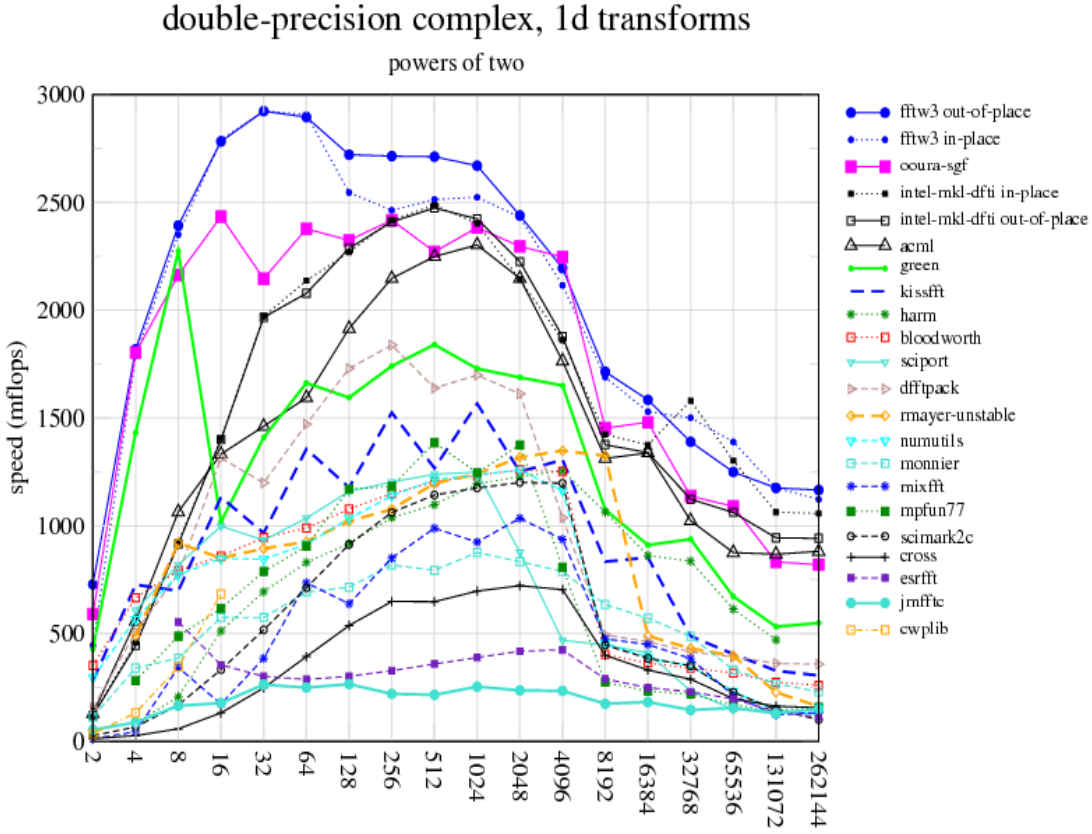
Supongamos que cada operación demora 1ns

10^{18} ns ~ 31.2 años
$30x10^9$ ns ~30 seg

ALGORITMOS

http://www.fftw.org/speed/





15 Sorting Algorithms in 6 Min.

http://youtube/kPRA0W1kECg

ALGORITMOS

Encontrar un mejor algoritmo

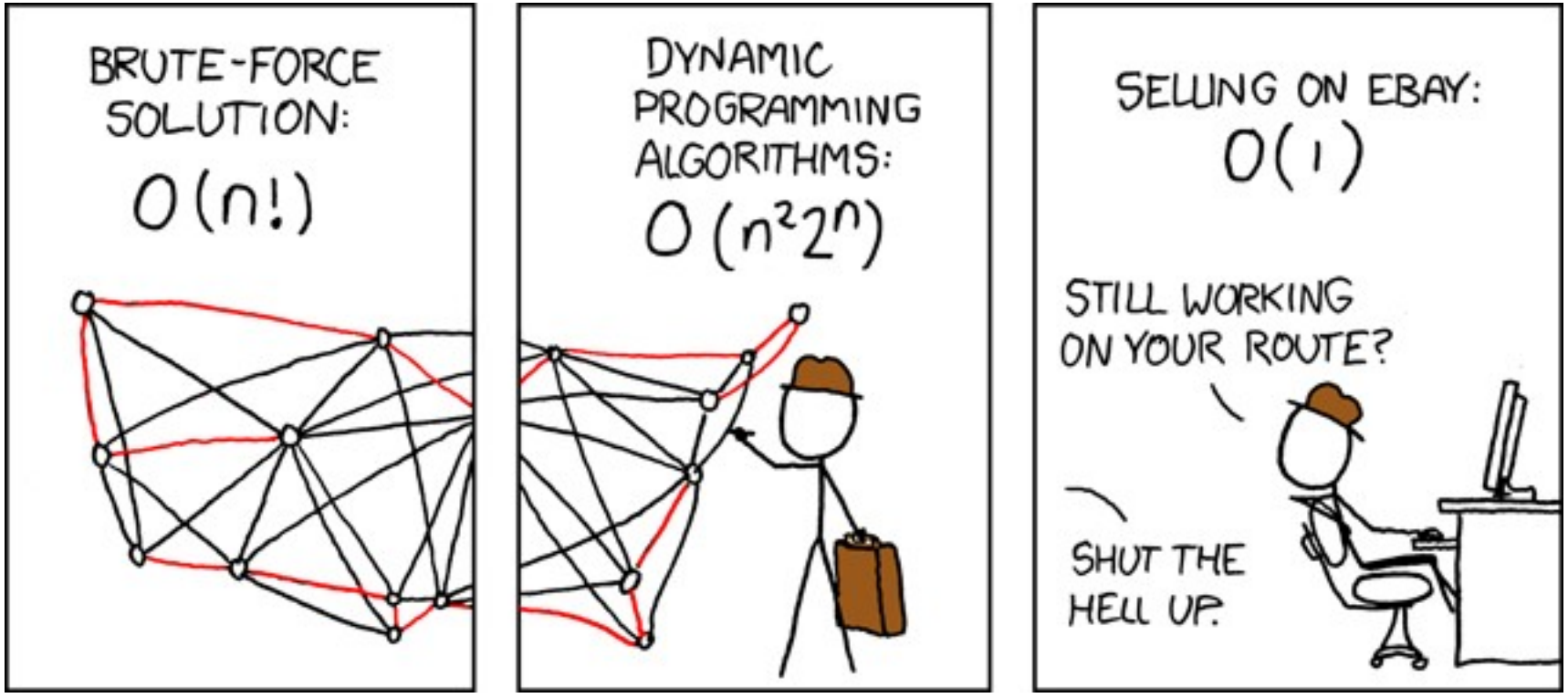
Es mejor que

Optimizar un algoritmo

!

ALGORITMOS

travel salesman problem



OPTIMIZACION

1. Respecto a los algoritmos

2. Respecto a las estructuras de datos

3. Respecto al hardware

ESTRUCTURAS DE DATOS

Almacenar información de forma organizada y estructurada y no como datos simples.

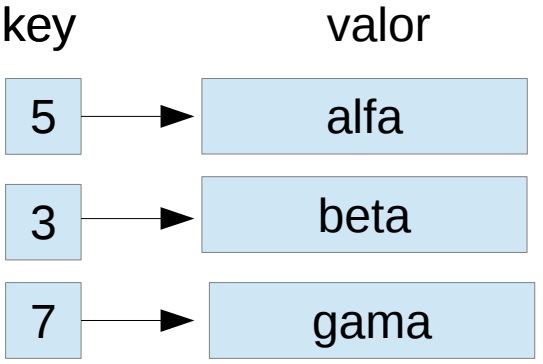
Se pueden ver como una **colección de datos que se caracterizan por su organización y las operaciones que se definen en ellos.**

Objetivo: tener un **fácil acceso y manejo de datos**

ESTRUCTURAS DE DATOS

Secuenciales:
Datos organizados consecutivamente.

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



ESTRUCTURAS DE DATOS

Contenedores secuenciales

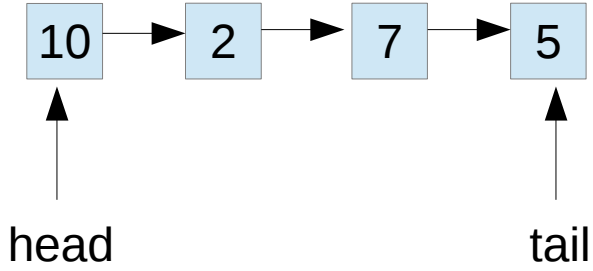
Arreglos

Insertar al final
O(1)

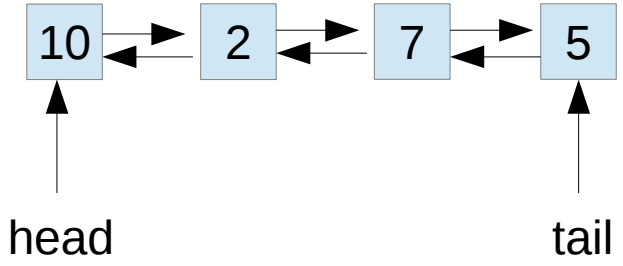
Insertar al principio
O(1)

Listas enlazadas

Single linked-list

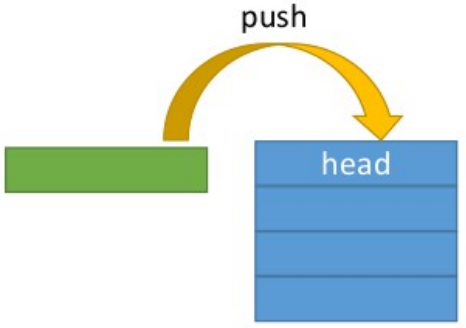


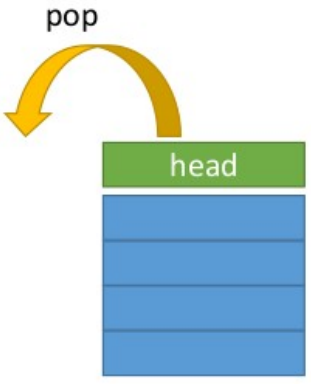
Double linked-list



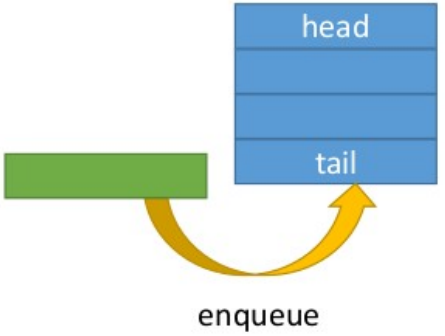
ESTRUCTURAS DE DATOS
Contenedores secuenciales

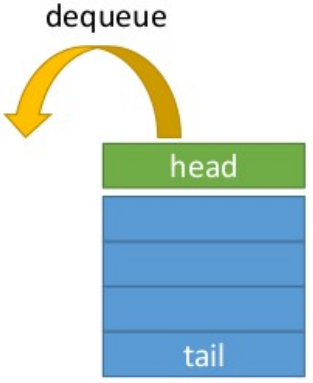
Pila (LIFO)





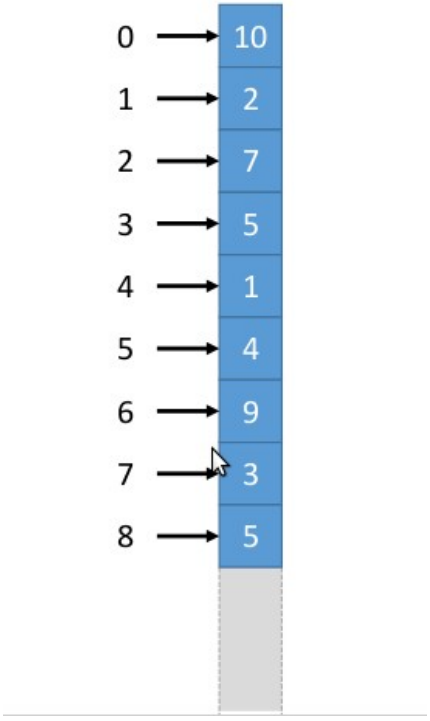
Cola (FIFO)



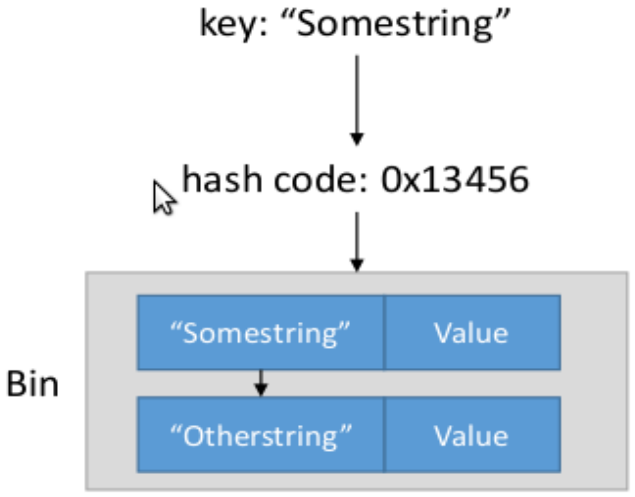


ESTRUCTURAS DE DATOS
Contenedores asociativos

Arreglos



Mapas no ordenados (hash maps)



+ diccionarios, mapas ordenados, conjuntos

OPTIMIZACION

1. Respecto a los algoritmos

2. Respecto a las estructuras de datos

3. Respecto al hardware

OPTIMIZACION

1. Respecto a los algoritmos

2. Respecto a las estructuras de datos

3. Respecto al hardware

OPTIMIZACION CON HARDWARE

Cómo hacer más rápido nuestros cálculos?

1. Escribir y leer
más rápido (I/O > veloc)

2. Dar vueltas más rápido
la manija (> veloc clock)

3. Mejorar la tecnología
(mejor CPU)

OPTIMIZACION CON HARDWARE

1. Escribir y leer más rápido (I/O > veloc)

Limita el acceso a memoria no-uniforme y multi-core

2. Dar vueltas más rápido la manija (> veloc clock)

Arquitecturas multi-cores. Paralelización

3. Mejorar la tecnología (mejor CPU)

Compiladores que permiten optimización para aprovechar la CPU (pipelining, unidades vectoriales y superescalares , etc)

OPTIMIZACION CON HARDWARE

El tipo de operaciones importa!!!

Operaciones económicas (más veloces),0.5x-1x

`+ - *`

Operaciones velocidad media,5x-10x

`/ % sqrt()`

Operaciones lentas, 20x-50x

`Func. trascendentales`

Operaciones costosas, muy lentas, >100x

`Pow(x,y) x,y reales`

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

OPTIMIZACION CON HARDWARE

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

OPTIMIZACION CON HARDWARE

Copy Propagation

Antes

```
x=y
z=1+x
```

← Hay dependencia

Después

```
x=y
z=1+y
```

OPTIMIZACION CON HARDWARE

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

OPTIMIZACION CON HARDWARE

Strength Reduction

Antes

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

Pow, / operaciones más costosas

Después

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

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.

OPTIMIZACION CON HARDWARE

Eliminación subexpresiones comunes

Antes

```
d=c*(a/b);
e=(a/b)*2.0;
```

Eliminar el Cálculo doble de la /

Después

```
x=a/b;
d=c*x;
e=x*2.0;
```

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)

OPTIMIZACION CON HARDWARE

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

OPTIMIZACION CON HARDWARE

Loop unrolling

Antes

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

X el compilador

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
```

OPTIMIZACION CON HARDWARE

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)

Loop fisión/fusión

Separar

```
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
```

Fusión →

← Fisión

Unir

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

OPTIMIZACION CON HARDWARE

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

En resumen:

1. Hay algún algoritmo más eficiente para resolver el mismo problema?

2. Cuello de botella: estructuras de datos y acceso a memoria eficiente.

3. Usar **bibliotecas optimizadas!**

4. Usar **opciones de optimización** del compilador.

5. Aprovechar las características del CPU (vectorial, paralelismo).

6. **Re-pensar el código.** A veces pequeños cambios mejoran sustancialmente la performance. No quedarse con la primera versión que anda!

1-11 Marzo de 2017
FACET -UNT

OPTIMIZACION

Graciela Molina

m.graciela.molina@gmail.com
gmolina@herrera.unt.edu.ar