Invarianza de Ciclo.

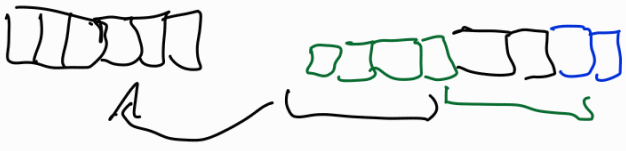
↳ Inducción

algo

Pruebo que $i+1$ se cumple basado en que i se cumple

Pruebo el estado final

Complejidad Amortizada



- Cuando la complejidad cambia segun el caso.

Arreglos Dinamicos.

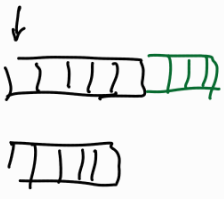
Incrementos en pot. de 2

Clase
{ int arr[]
int tamaño=0

}

2^0
2^1
2^2
2^3
$\vdots$

← ArrayList<> → int[]
← Vector<> → int arr[]
← List



#Insertados | OC

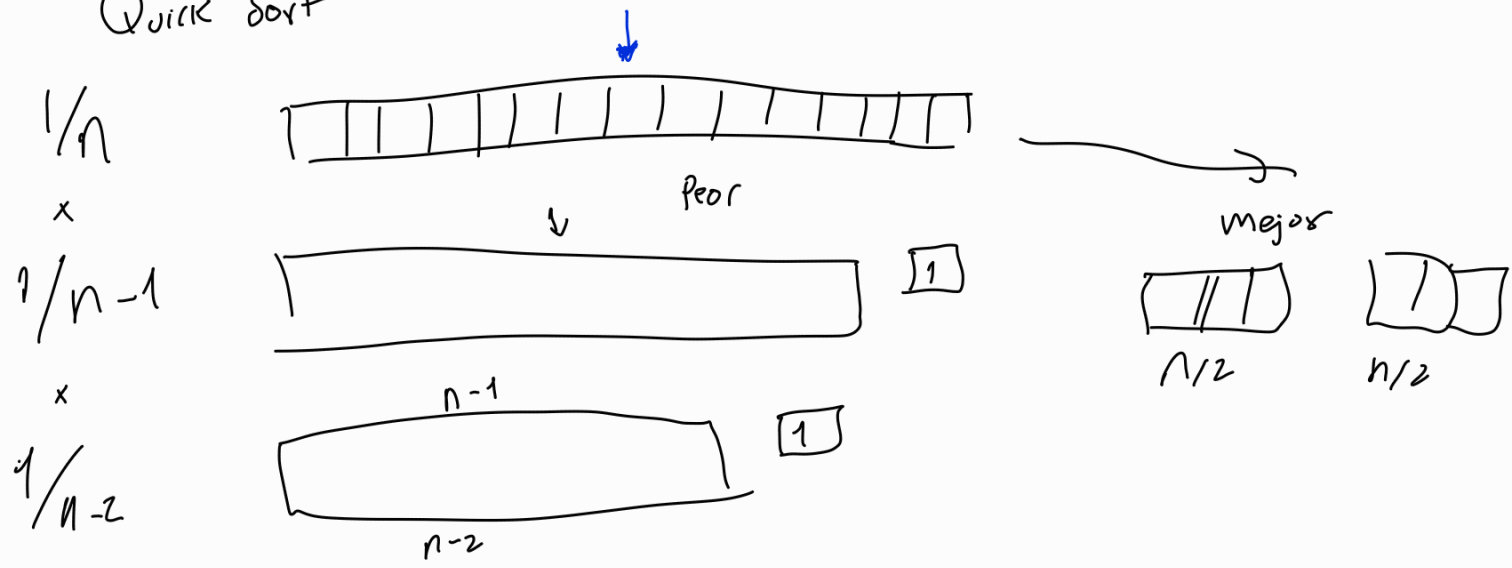
0 → 1
1 → 2
2 → 1
3 → 4
4 → 1
5 → 8
6 → 1
7 → 1
8 → 1
9 → 16

$\sum f(i)$ → $f(i)$ es la complejidad en la iteracion i.

$$\sum_{i=0}^{\log n} 2^i$$

- Cuando se hacen algoritmos cuyos estados dependan de una variable aleatoria.

Quick sort



$\sim$
$\sim$

$\frac{1}{n^2} \approx$ $n=1000$ $\frac{1}{1000000}$

$n-3$

$n-4$

1

- Peor de los casos

$\sum_{i=0}^{n} i = \sim O(n^2)$

↑ no se cumple siempre.

Parece "seguro" Ordenar

↓

El mejor de los casos.

$\sim$ Merge sort

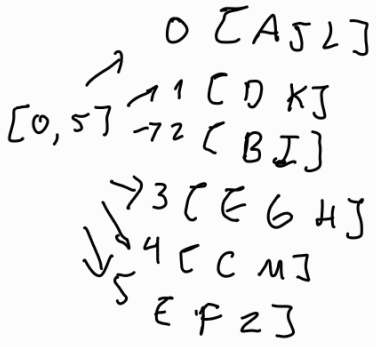
$O(n \log n)$

In-place.

↳ Su resolución no requiere creación de memoria extra y todo se puede resolver usando la memoria existente.

Ordenamiento lineal (max (|d| , n))

Counting Sort. → contar Frecuencias.



10000000

$10^7 \cdot \log_2 10^7$ 10^7

A J L D K B I E G H C M F Z

- Cuando el rango de elementos dentro del arreglo a ordenar es considerablemente menor que el tamaño del arreglo.
- Ordenar personas por estrato economico.

Radix sort → Tarea investigar e implementar

10^5 10^4 10^3 Centenas Decenas Unidades O(n)

1	3	7
2	1	1
9	5	2

0 2 1 1
2 9 5 2 1 3 7
1 3 7

0 2 1 1
1 1 3 7 → 2 1 1
9 5 2 9 5 2

1 3 7
2 1 1
9 5 2