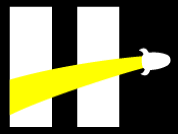
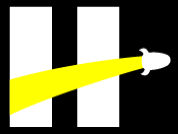

HENRY

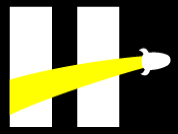


Algoritmos



Algoritmos

Un algoritmo es un conjunto prescrito de instrucciones o reglas bien definidas, ordenadas y finitas que permite realizar una actividad mediante pasos sucesivos que no generen dudas a quien deba realizar dicha actividad. O sea, una serie de pasos a seguir para completar una tarea.

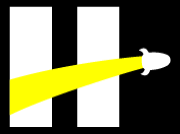


Algoritmos

1. **Resuelve un problema:** Este es el objetivo principal del algoritmo, fue diseñado para eso. Si no cumple el objetivo, no sirve para nada :S.

2. **Debe ser comprensible:** El mejor algoritmo del mundo no te va a servir si es demasiado complicado de implementar.

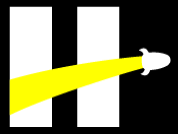
3. **Hacerlo eficientemente:** No sólo queremos tener la respuesta perfecta (o la más cercana), si no que también queremos que lo haga usando la menor cantidad de recursos posibles.



Algoritmos

¿Cómo medimos la eficienta del un algoritmo?

- Tiempo
- Espacio
- Otros recursos:
  - Red
  - Gráficos
  - Hardware (Impresoras, Cpus, Sensores, etc...)



Algoritmos

Ejemplo juego Adivinar un número:

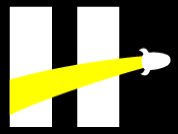
$$N = 1/2^x$$

$$log^2(2^x) = log^2 N$$

$$x * log^2(2) = log^2 N$$

$$x * 1 = log^2 N$$

$$x = log^2 N$$

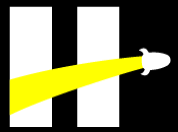


Algoritmos

Ahora... ¿Por qué nos importa medir la complejidad de los algoritmos? Básicamente nos va a servir a:

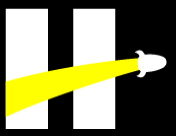
1. Predecir el comportamiento: hay casos donde algo puede tardar tanto que no tenemos el lujo del prueba y error, tenemos que conocer de antemano si un algoritmo va a terminar en un tiempo adecuado para el problema.

2. Compararlos: Según el problema vamos a tener que decidir cuál es el mejor algoritmo para usar, tampoco podemos ponernos a probar uno por uno.

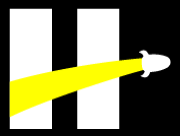


Cota superior asintótica (Big O Notation / Notación O grande)

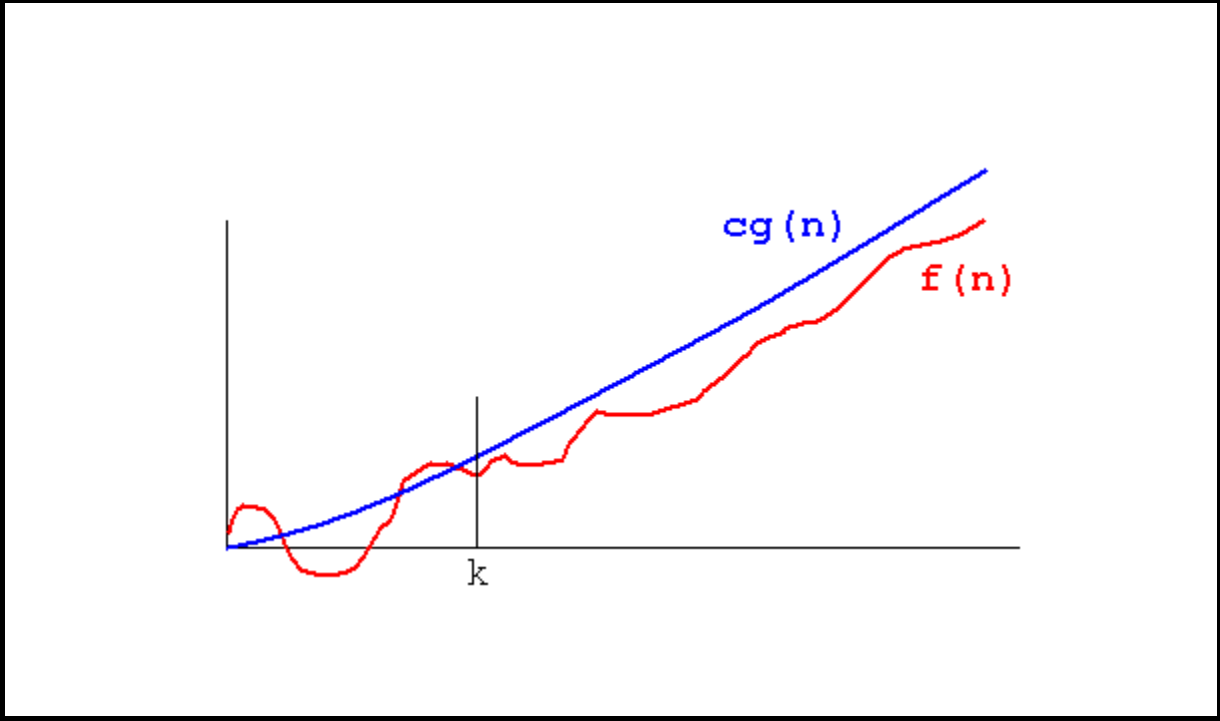
$$O(g(x)) = \left\{ \begin{array}{c} f(x) : \text{existen } x_0, c > 0 \text{ tales que} \\ \forall x \geq x_0 > 0 : 0 \leq |f(x)| \leq c|g(x)| \end{array} \right\}$$

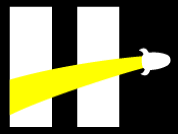


Cota superior asintótica (Big O Notation / Notación O grande)



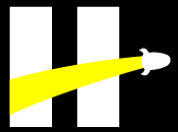
Cota superior asintótica (Big O Notation / Notación O grande)





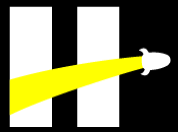
Cota superior asintótica (Big O Notation / Notación O grande)

N	N^2	N^2+N	%N
10	100	110	10%
100	10,000	10,100	1%
1,000	1,000,000	1,001,000	0.1%
10,000	100,000,000	100,010,000	0.01%
100,000	10,000,000,000	10,000,100,000	0.001%



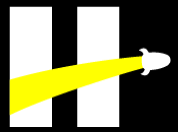
Ejemplos

```
1 var max = array[0];
2 for( var i = 0, i <= array.lenght; i++){
3         if( array[i] > max)     {
4                 max = array[i];
5         }
6 }
7 console.log(max);
8 }; // O ( N )
```



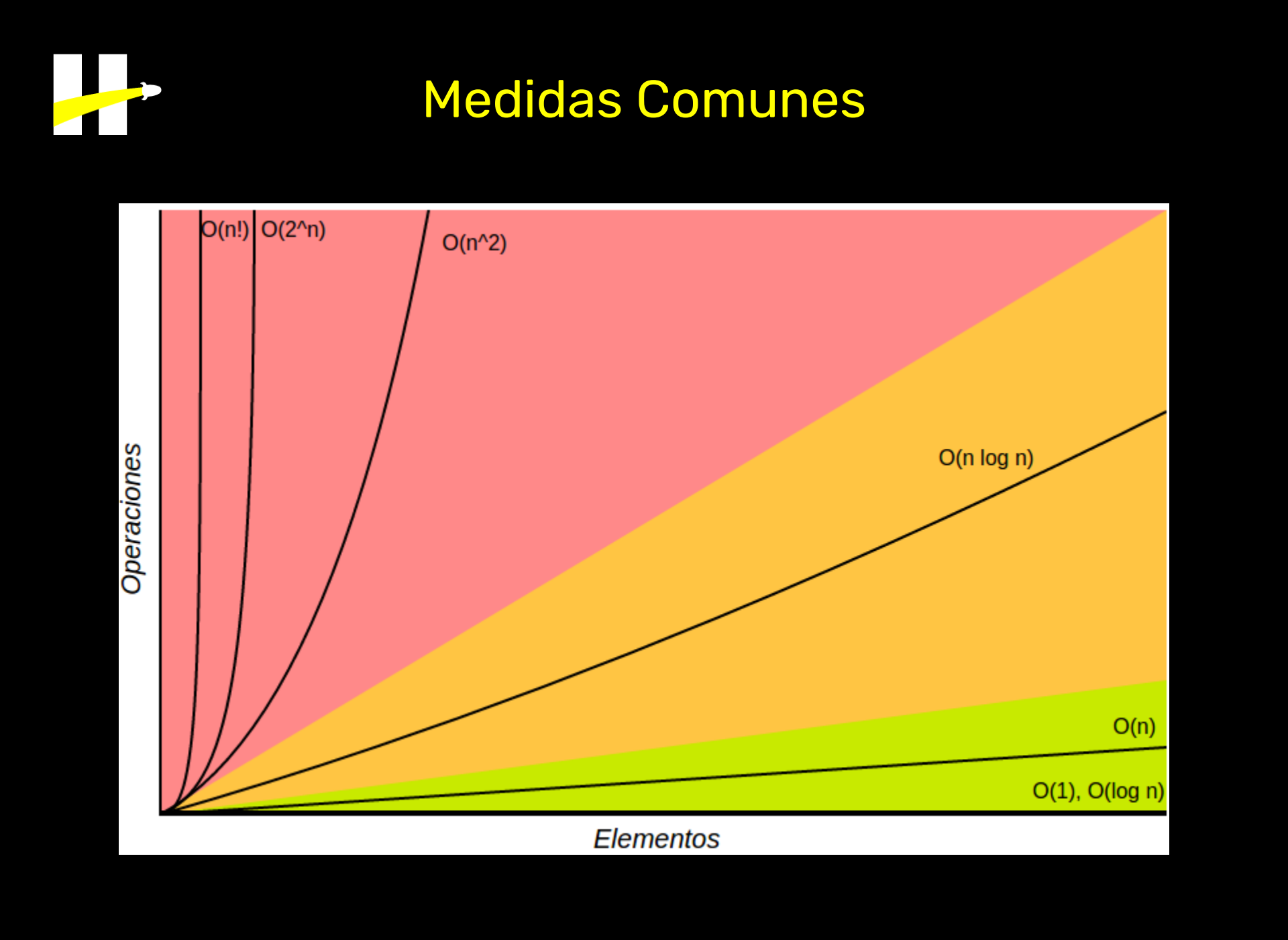
Ejemplos

```
1 for( var i = 0, i <= array.lenght; i++){
2   for( var j = 0, j <= array.lenght; j++){
3         if(array[i] === array[j]){
4           return true;
5         }
6    }
7 };
8 // O( N x N) = O (n²)
```

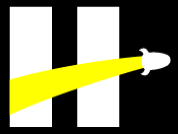


Ejemplos

```
1  function sumArray(array, n) {
2    var fin = array.length - 1;
3    var ini = 0;
4    while (ini < fin) {
5      var suma = array[ini] + array[fin];
6      if( suma === n) return true;
7      if( suma > n) fin = fin - 1;
8      if( suma < n) ini = ini + 1;
9    }
10   return false;
11 };
```

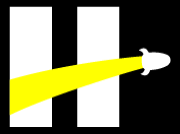


Medidas Comunes



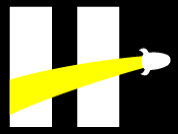
Medidas Comunes

Runtime	F(1,000)	Time
O(Log N)	10	0.00001 sec
$O(\sqrt{N})$	32	0.00003 sec
N	1,000	0.001 sec
N^2	1,000,000	1 sec
2^N	1.07×10^{301}	3.40×10^{287} years
N!	4.02×10^{2567}	1.28×10^{2554} years

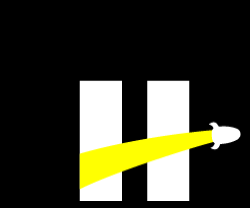


Qué cantidad de datos podría cada algoritmo en un segundo?:

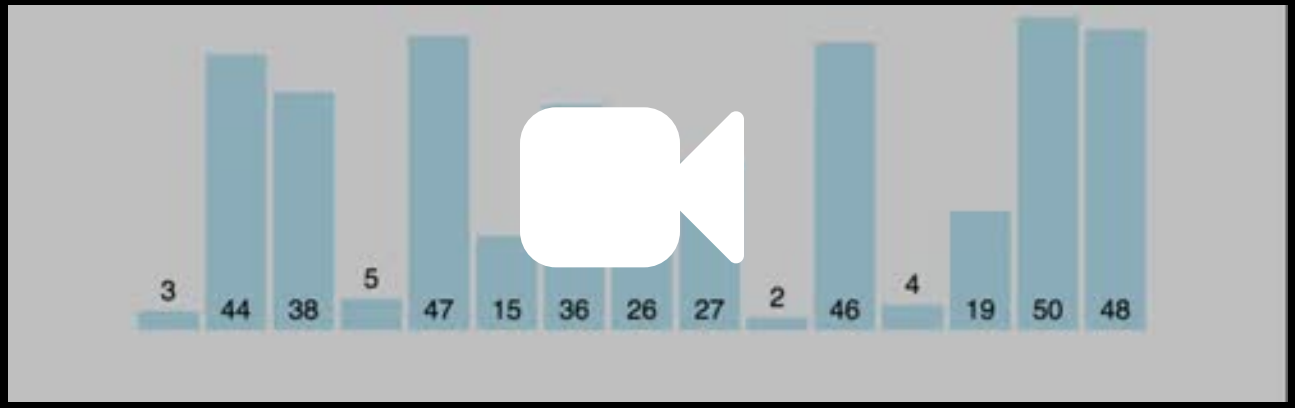
Runtime	N
O(Log N)	$> 1\times10^{300,000}$
$O(\sqrt{N})$	1 trillion
N	1 million
N^2	1 thousand
2^N	20
N!	10

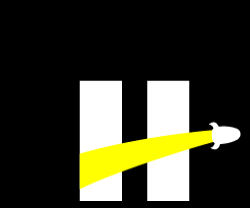


Algoritmos de Ordenamiento I

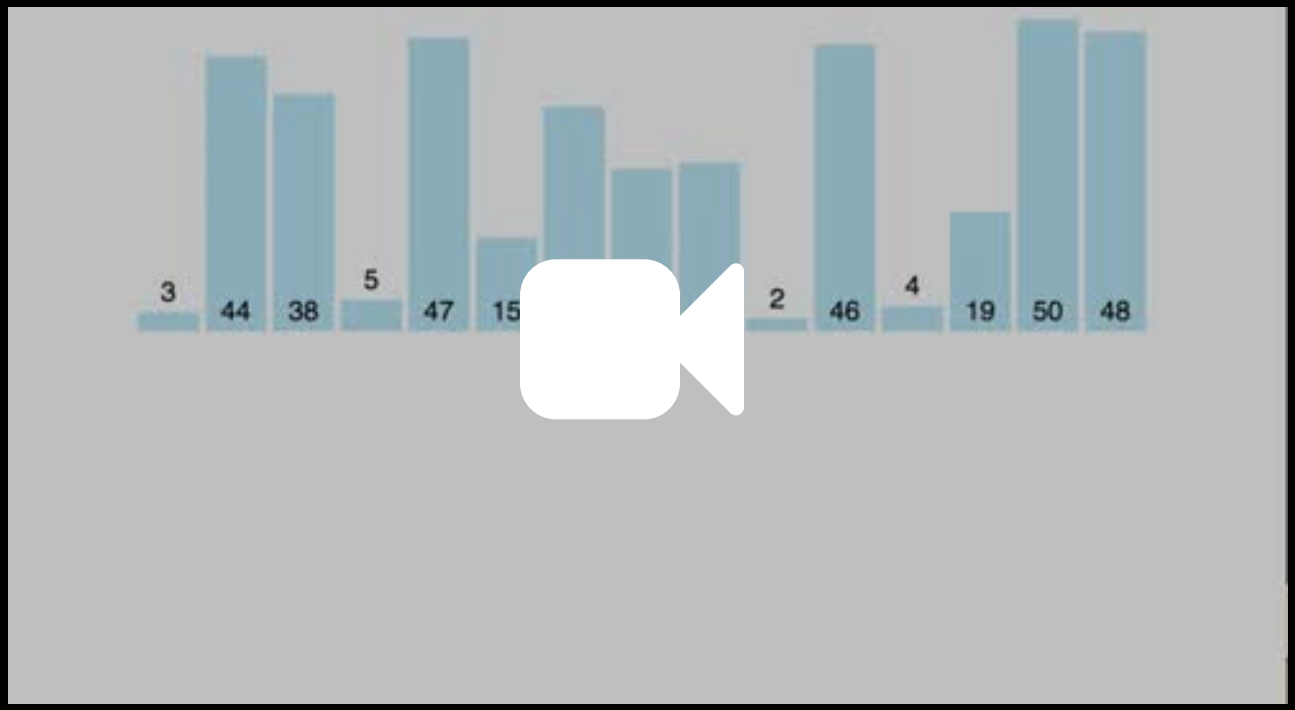


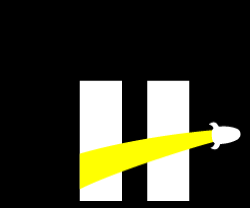
Bubble Sort



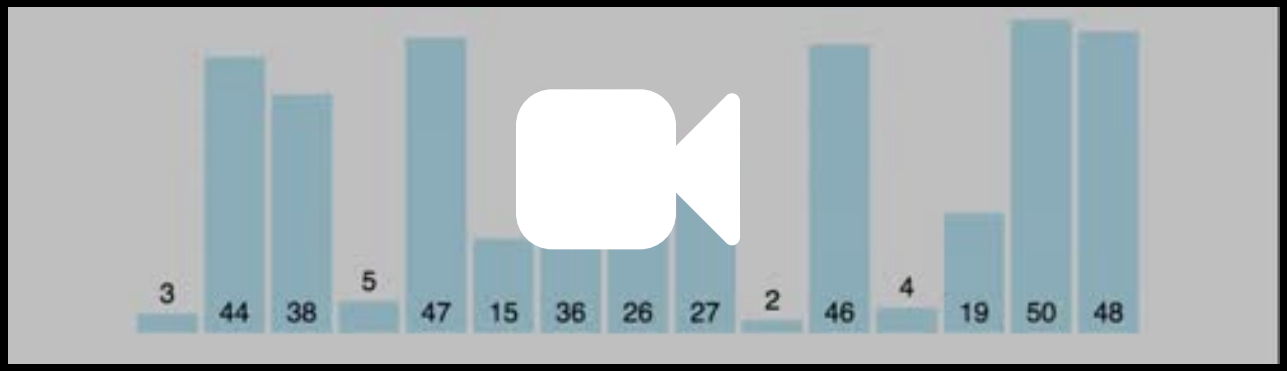


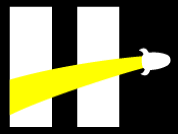
Insertion Sort





Selection Sort

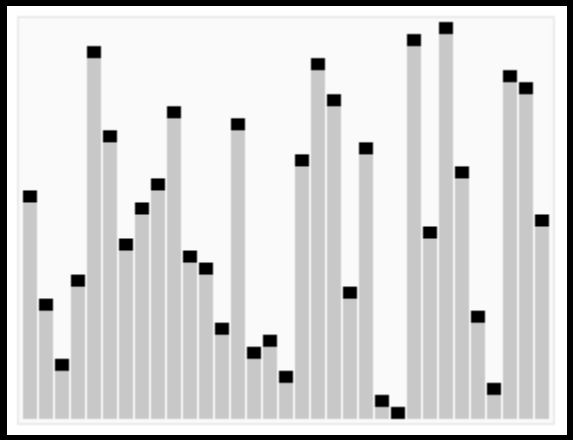


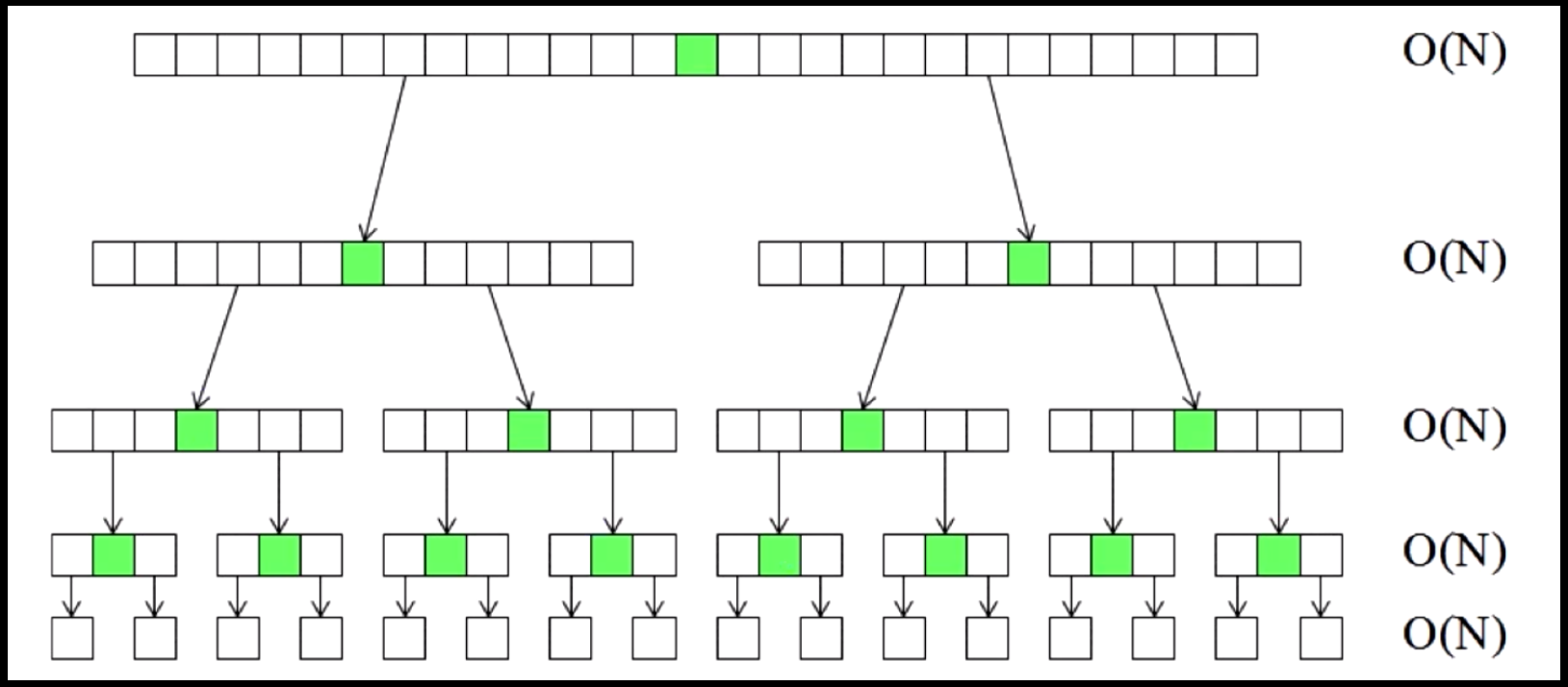


Algoritmos de Ordenamiento II

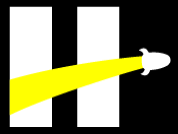
Quick Sort

- Elegir un elemento de la lista de elementos a ordenar, al que llamaremos pivote.
- Mover los demás elementos de la lista a cada lado del pivote, de manera que a un lado queden todos los menores que él, y al otro los mayores. Los elementos iguales al pivote pueden ser colocados tanto a su derecha como a su izquierda, dependiendo de la implementación deseada. En este momento, el pivote ocupa exactamente el lugar que le corresponderá en la lista ordenada.
- La lista queda separada en dos sublistas, una formada por los elementos a la izquierda del pivote, y otra por los elementos a su derecha.
- Repetir este proceso de forma recursiva para cada sublista mientras éstas contengan más de un elemento. Una vez terminado este proceso todos los elementos estarán ordenados.

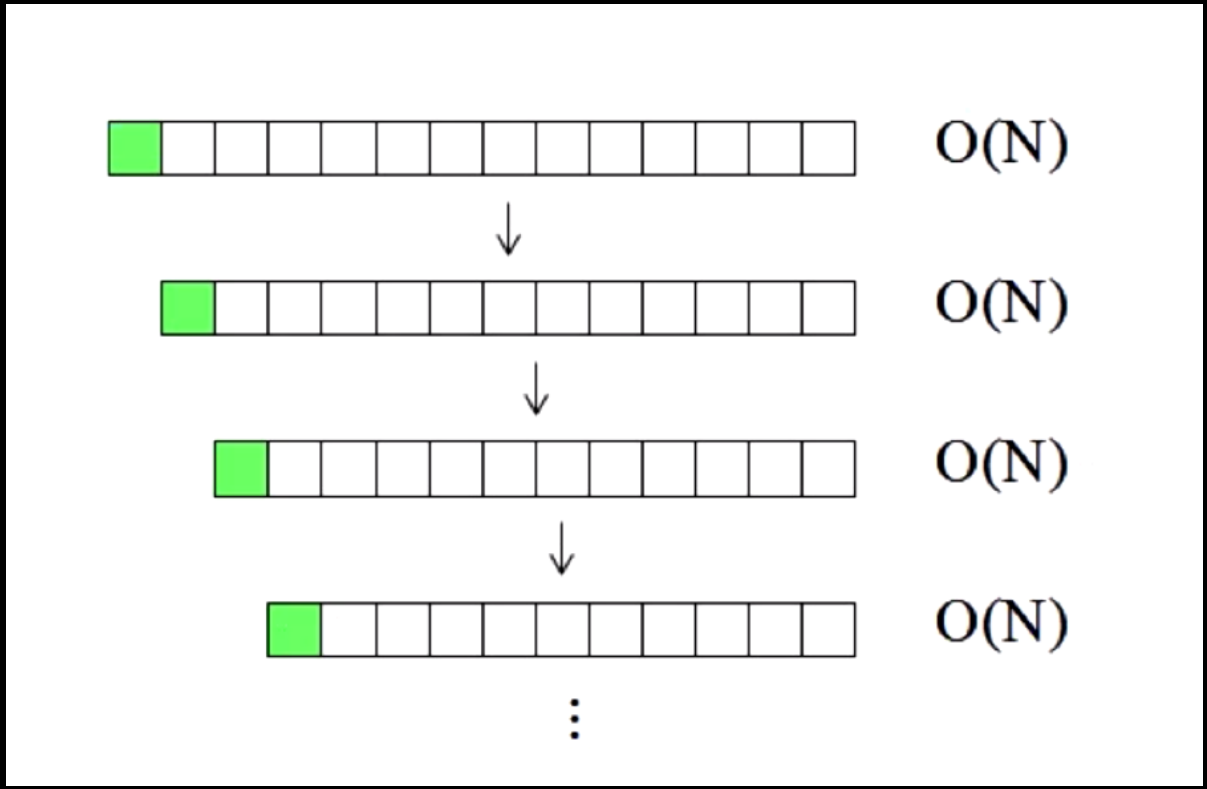


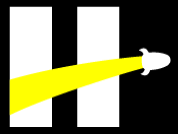


Quick Sort



Quick Sort





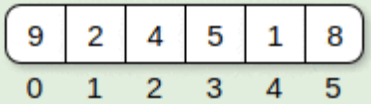
Merge Sort

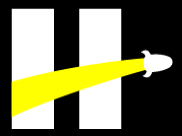
1 - Divide el conjunto en dos grupos iguales.

2 - Ordena recursivamente los dos grupos

3 - Junta (o mergea) los grupos ordenados.

Merge Sort





Merge Sort

Complejidad:

- O(N) steps per level
- O(Log N) Levels
- Total: O(N Log N)