

Algoritmos de Programación en Python

EXPLORANDO EL USO DE LOS
ALGORITMOS EN EL
LENGUAJE PYTHON

Contenido



Recursividad



Memorización



Divide y vencerás



Ordenamiento



Búsqueda

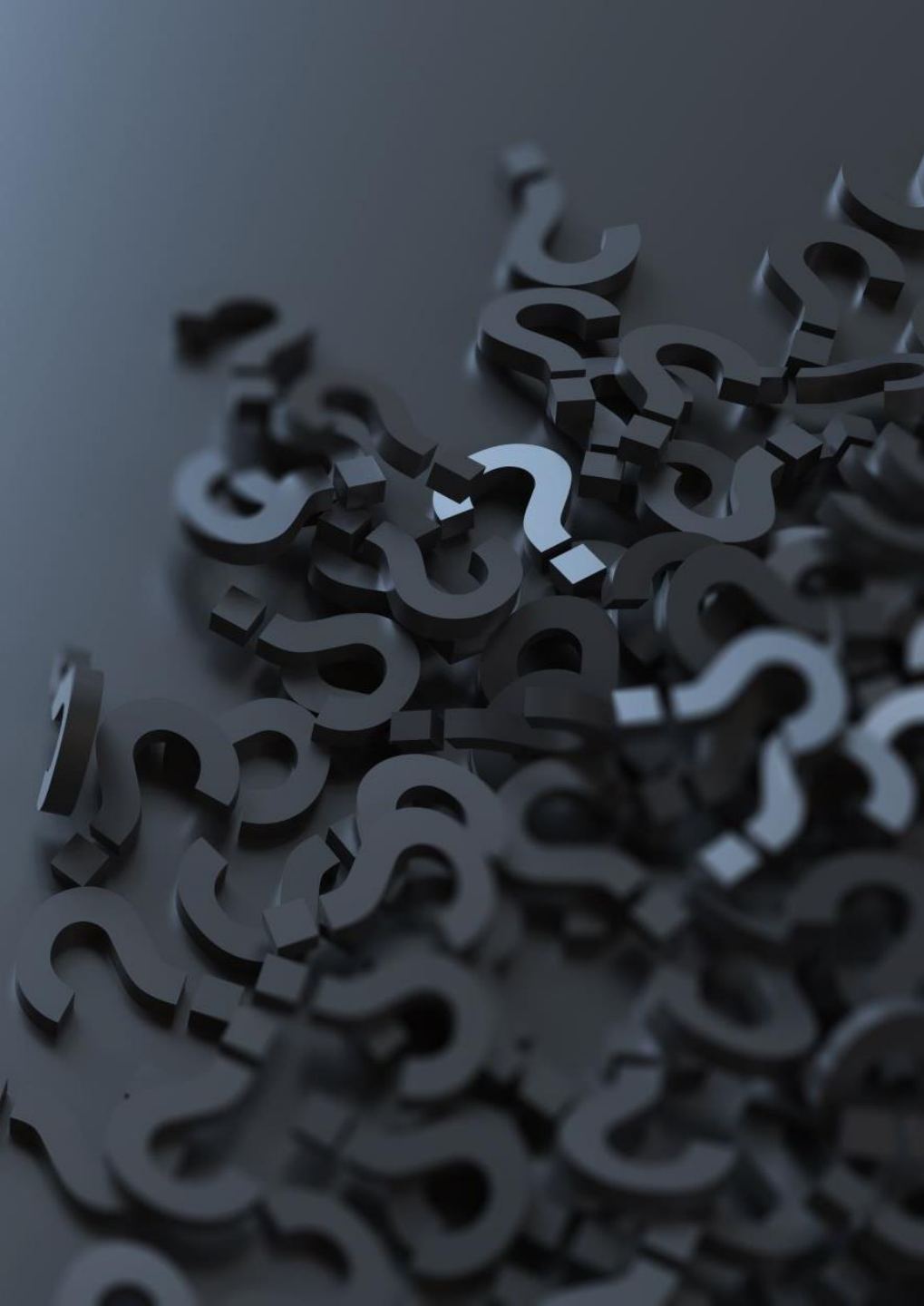


Paralelismo



Presentación

En esta presentación, se presentará una introducción a los algoritmos de programación y su aplicación en el lenguaje de programación Python. Se cubrirán los conceptos básicos de los algoritmos y cómo se pueden implementar en Python para resolver problemas complejos.



Introducción

- Algoritmo es el término que usamos para referirnos a un método para hacer algo.
- Los científicos de computadoras estudian nuevos métodos (algoritmos) para realizar algo.
- Uno de los retos más comunes es el ordenamiento de datos, por lo que existen múltiples algoritmos de ordenamiento, los cuales pudieran desempeñarse de mejor o peor manera dependiendo de la situación.

Introducción a los algoritmos de programación

¿Qué son los algoritmos de programación?

Los algoritmos de programación son secuencias lógicas de instrucciones que le dicen a una computadora cómo realizar una tarea. Estos algoritmos pueden variar en complejidad y tamaño según la tarea que se quiera realizar.

¿Cómo funcionan los algoritmos de programación?

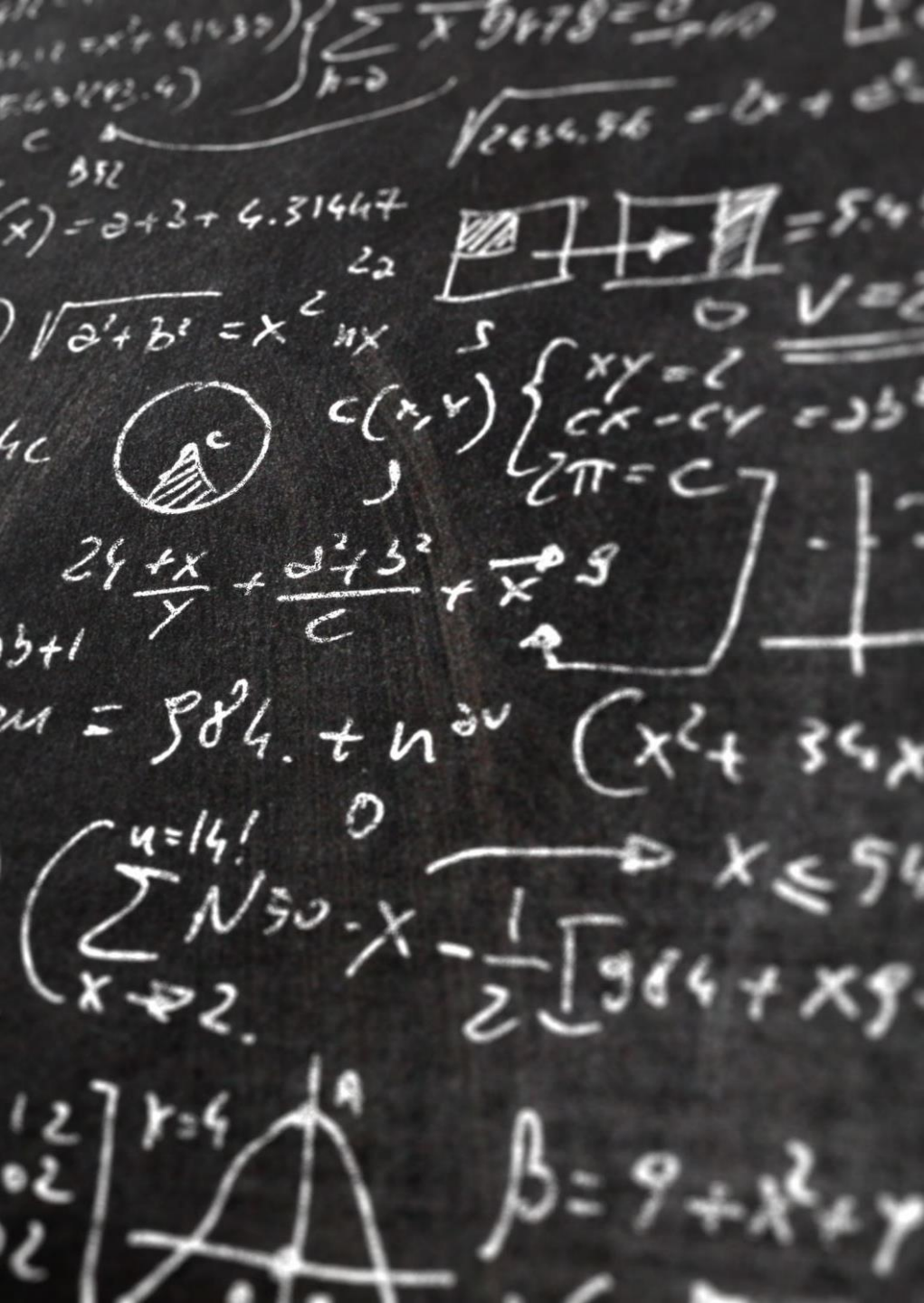
Los algoritmos de programación funcionan mediante la entrada de datos, procesamiento y salida de los resultados. El proceso de entrada, procesamiento y salida se repite a medida que se ejecuta el algoritmo hasta que se alcanza la solución deseada.

Ordenar una lista

La tarea de ordenar una lista de elementos es común en la programación y se puede realizar mediante el uso de algoritmos como el de burbuja o el de selección.

Encontrar el valor máximo

Encontrar el valor máximo en una lista de elementos también es una tarea común en la programación y se puede realizar mediante el uso de algoritmos como el de búsqueda lineal o el de búsqueda binaria.



Algoritmos

Existen diferentes tipos de algoritmos de programación y técnicas de resolución de problemas:

- Cuantitativos – Su resultado es un valor específico que proviene de cálculos.
- Cualitativos – Resultan de la revisión de condiciones

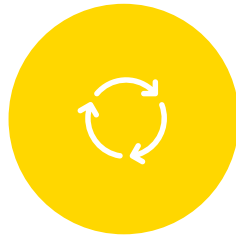
Recursividad



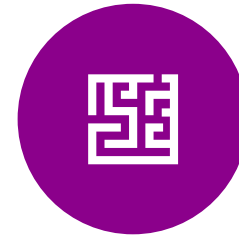
ES UNA TÉCNICA DE
SOLUCIÓN DE PROBLEMAS.



UNA FUNCIÓN RECURSIVA
ES UNA FUNCIÓN QUE SE
LLAMA A SÍ MISMA.



TODO PROBLEMA QUE
PUEDA SER RESUELTO DE
FORMA RECURSIVA, PUEDE
SER RESUELTO USANDO
CICLOS.



RESOLVER UN PROBLEMA
MEDIANTE RECURSIÓN
SIGNIFICA QUE LA
SOLUCIÓN DEPENDE DE
LAS SOLUCIONES DE
PEQUEÑAS INSTANCIAS
DEL MISMO PROBLEMA.




SE BASA EN LA
RECURRENCIA.

Construir un programa recurrente

Cada llamada recurrente se debe definir sobre un problema de menor complejidad (algo más fácil de resolver).



Ha de existir al menos un caso base para evitar que la recurrencia sea infinita.



Ejemplo de recursividad: Factorial

función factorial:

input: entero n de forma que $n \geq 0$

output: $[n \times (n-1) \times (n-2) \times \dots \times 1]$

1. if n es 0, **return** 1

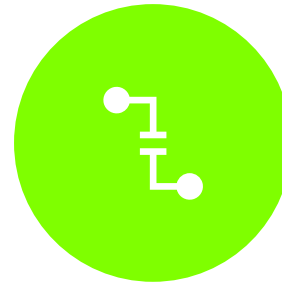
2. else, **return** $[n \times \text{factorial}(n-1)]$

end factorial

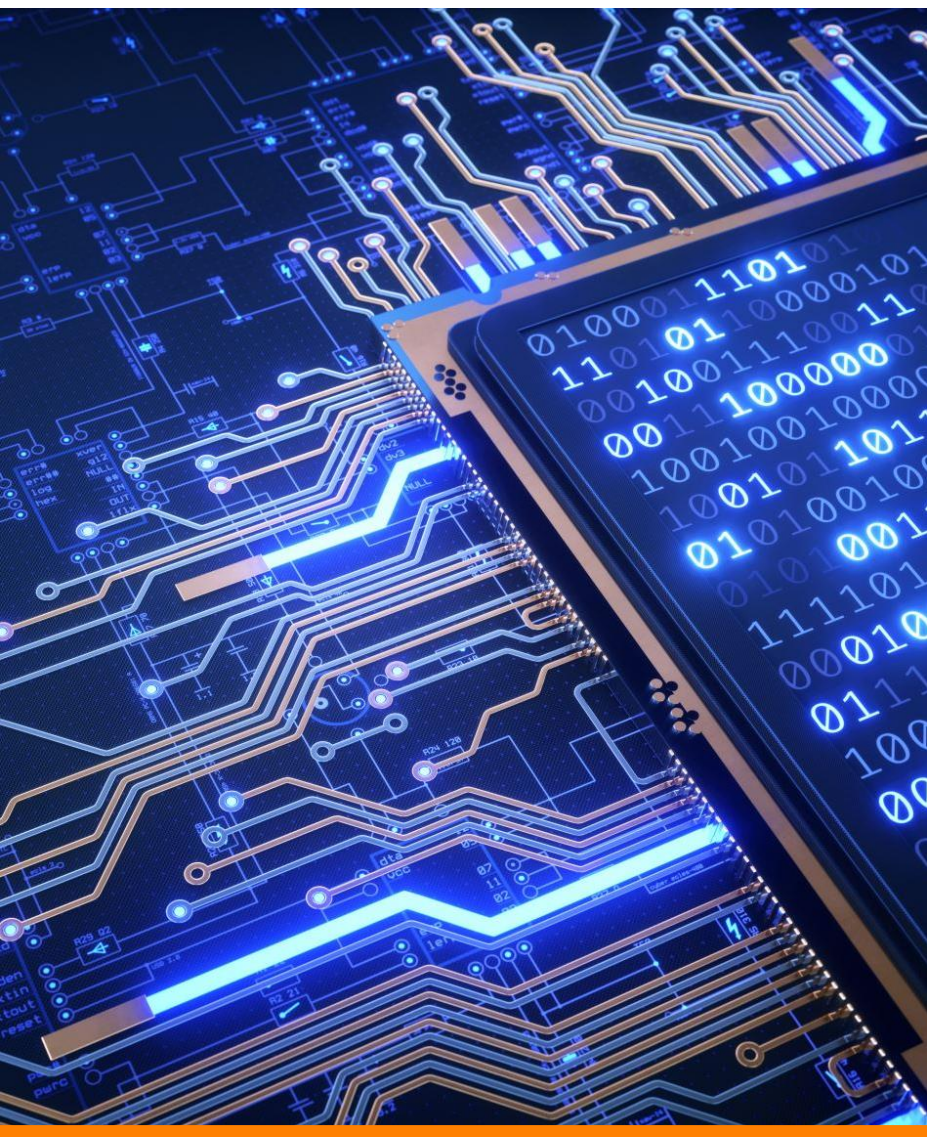
Algoritmos recursivos y algoritmos iterativos



Llamaremos *algoritmos recursivos* a aquellos que realizan llamadas recursivas para llegar al resultado, y *algoritmos iterativos* a aquellos que llegan a un resultado a través de una iteración mediante un ciclo definido o indefinido.



Todo algoritmo recursivo puede expresarse como iterativo y viceversa. Sin embargo, según las condiciones del problema a resolver podrá ser preferible utilizar la solución recursiva o la iterativa.



Memorización (memoización)

- Técnica de optimización usada principalmente para acelerar programas de computadoras a través de almacenar los resultados de llamadas “costosas” a funciones y retornando el resultado almacenado en el caché cuando los mismos parámetros de entrada vuelven a ocurrir.
- Se considera un caso específico de optimización.

Usos de la memorización



Se usa para acelerar el trabajo de traductores (parsers), es decir, en programas que interpretan lenguajes.



Es útil en todos los problemas donde el resultado de una operación reciente volverá a ser preguntado de nuevo.



Algunos aceleradores de traducción de lenguajes almacenan los resultados de las llamadas a funciones automáticamente (auto memoización), para que la función no sea ejecutada la próxima vez que sea llamada.



De aquí evolucionaron los búferes y se diseñó el algoritmo de reemplazo de páginas para sistemas operativos.



Pivote

Puede ser una variable temporal para solucionar un problema, como un intercambio de valores.

También se define como un elemento específico de un arreglo o conjunto de datos que tomamos como referencia para hacer un determinado algoritmo a partir de él.

Python no requiere pivotes para intercambio de valores

```
a = int(input("Ingresa el valor de a: "))
b = int(input("Ingresa el valor de b: "))

(a,b) = (b,a)

print('a: ',a)
print('b: ',b)

c = int(input("Ingresa el valor de c: "))

(a,b,c) = (c,a,b)

print('a: ',a)
print('b: ',b)
print('c: ',c)
```



Divide y vencerás (DYV)



Método basado en la solución recursiva de un problema dividiéndolo en dos o más subproblemas de igual tipo o similar.



El proceso continúa hasta que éstos llegan a ser lo suficientemente sencillos como para que se resuelvan directamente.



Al final, las soluciones de los problemas se combinan para dar una solución al problema original.



Usos: algoritmos eficientes de ordenamiento (Quicksort, mergesort), multiplicar números grandes, análisis sintácticos y la transformada discreta de Fourier.



Algoritmo DYV

Función DYV(problema):

 Si el problema es trivial:
 resuélvelo

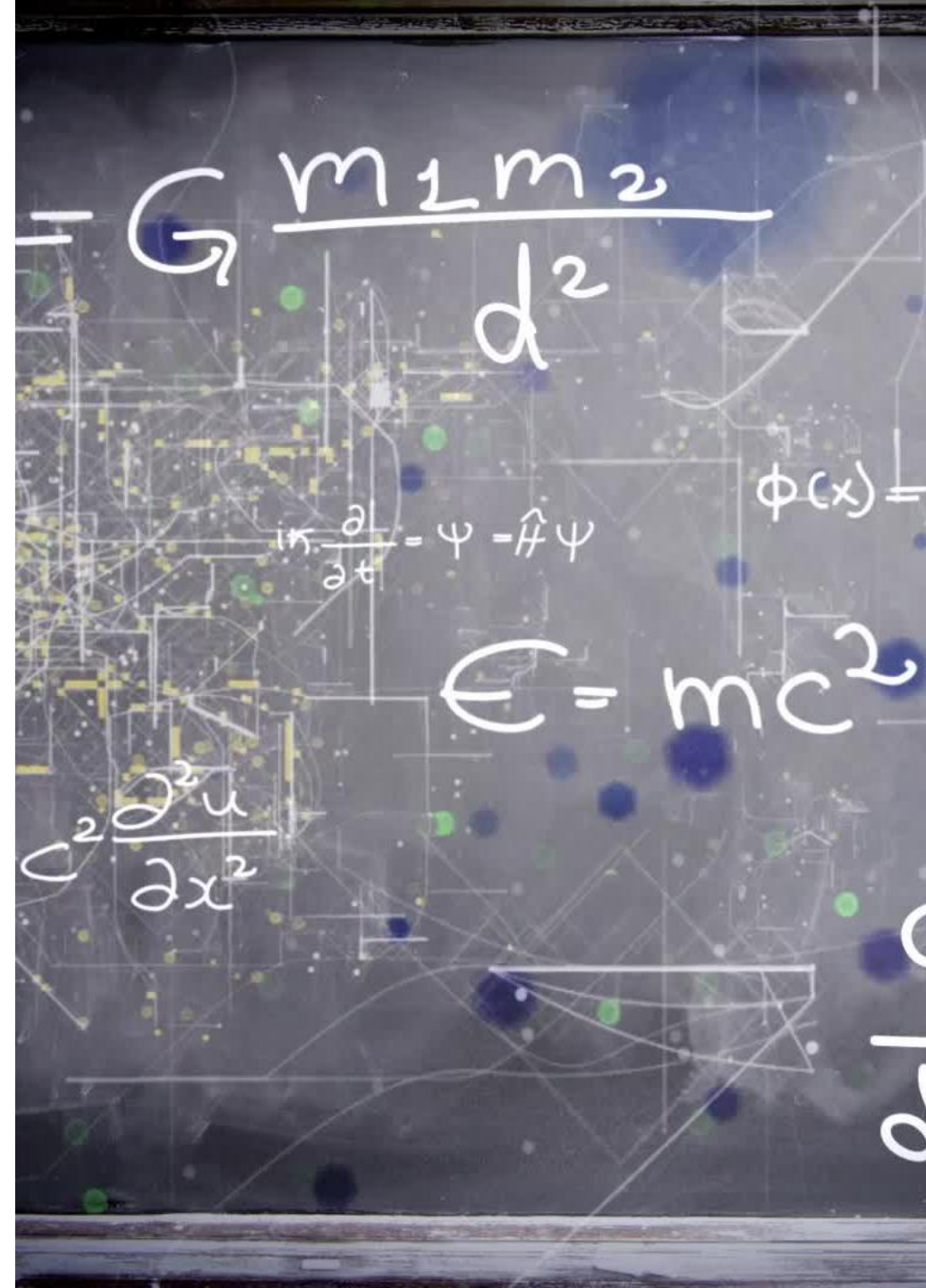
 Si no es trivial:

 Descomponer el problema en n
subproblemas

 Para $I = 1$ hasta n hacer

 DYV(subproblema _{i})

 Combinar soluciones





PROCESO DE TRES PASOS – DIVIDE Y
VENCERÁS



1. Divide / Break



Este paso implica dividir el problema en subproblemas más pequeños.



Los subproblemas deben representar una parte del problema original.



Este paso generalmente toma un enfoque recursivo para dividir el problema hasta que ningún subproblema sea más divisible.



En esta etapa, los subproblemas se vuelven de naturaleza atómica pero aún representan una parte del problema real



2. Vencer / Resolver



Este paso recibe muchos subproblemas más pequeños que deben resolverse.



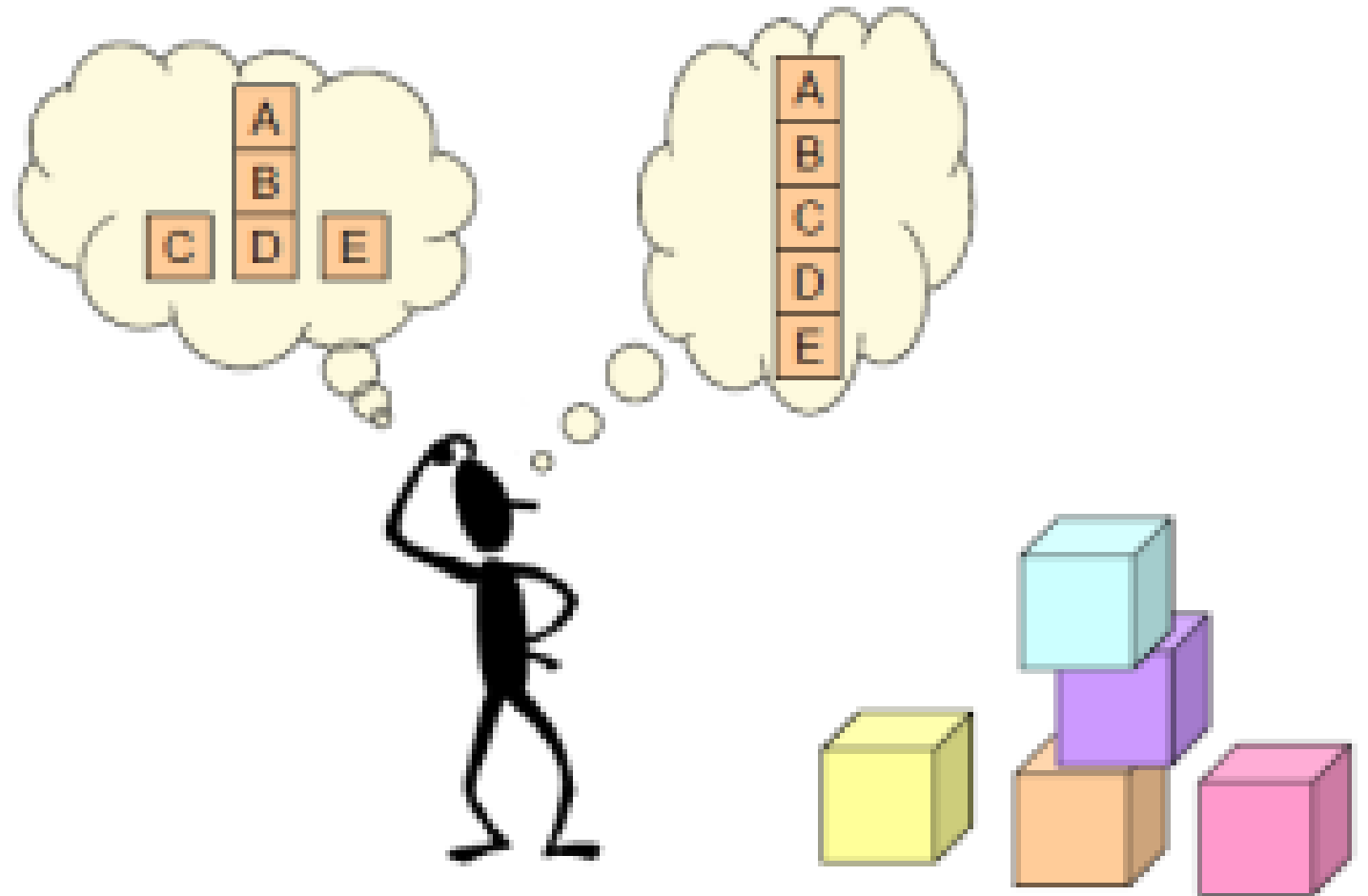
En general, a este nivel, los problemas se consideran "resueltos" por sí mismos

3. Fusionar / Combinar

- Cuando se resuelven los subproblemas más pequeños, esta etapa los combina de manera recursiva hasta que forman una solución del problema original.
- Este enfoque algorítmico funciona de forma recursiva y los pasos de conquistar y fusionar funcionan tan cerca que aparecen como uno solo.



Algoritmos de ordenamiento



Algoritmos de ordenamiento



Existen múltiples algoritmos para ordenamiento de arreglos, listas y otras estructuras de datos.



Su complejidad puede variar drásticamente, así como sus tiempos de ejecución.



En general, los algoritmos más sencillos de comprender y programar son los menos eficientes y viceversa.



Además, existen algoritmos que funcionan de forma excelente con cantidades grandes de elementos, pero se vuelven ineficientes con cantidades pequeñas.



Algoritmos de ordenamiento

En la ordenación de datos la *eficiencia* es el factor que mide la calidad y rendimiento de un algoritmo

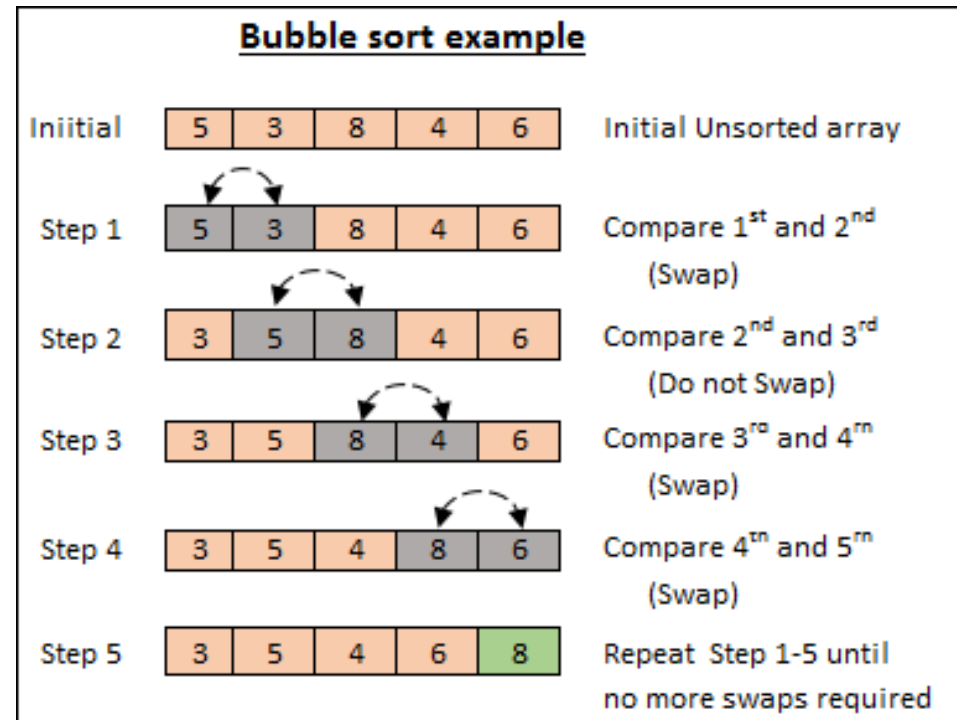
Para ordenar una lista de valores se deben considerar 2 criterios para decidir que algoritmo de ordenación utilizar:

Los métodos de ordenación pueden ser internos o externos, según que los elementos a ordenar estén en memoria principal o en memoria secundaria

El tiempo de ejecución

Cantidad de memoria utilizada

Método de la burbuja



El primer elemento se compara con el segundo, el segundo con el tercero y así sucesivamente; Si el elemento a comparar es más pequeño (o grande) que el primero, los elementos se intercambian.

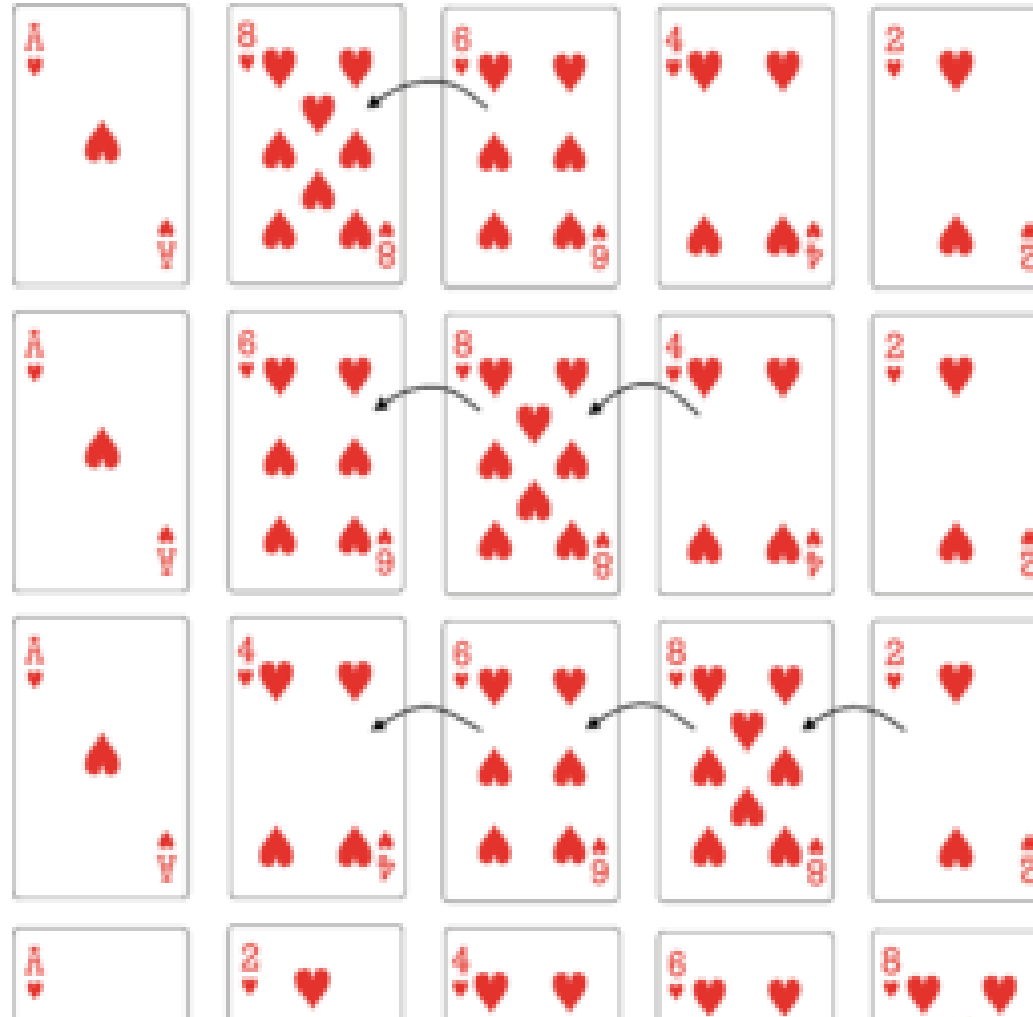
Después de la primera iteración, el elemento máximo (o mínimo) se coloca en la última posición. El proceso se repite para el segundo elemento y así sucesivamente.

Algoritmo del método de la Burbuja

```
procedimiento DeLaBurbuja ( $a_0, a_1, a_2, \dots, a_{(n-1)}$ )  
  para  $i \leftarrow 1$  hasta  $n - 1$  hacer  
    para  $j \leftarrow 0$  hasta  $n - i - 1$  hacer  
      si  $a_{(j)} > a_{(j+1)}$  entonces  
         $aux \leftarrow a_{(j)}$   
         $a_{(j)} \leftarrow a_{(j+1)}$   
         $a_{(j+1)} \leftarrow aux$   
      fin si  
    fin para  
  fin para  
fin procedimiento
```

Ordenamiento por inserción

ES UN MÉTODO MUY
SIMILAR A CUANDO
JUGAMOS CARTAS



—

4

3

2

10

12

1

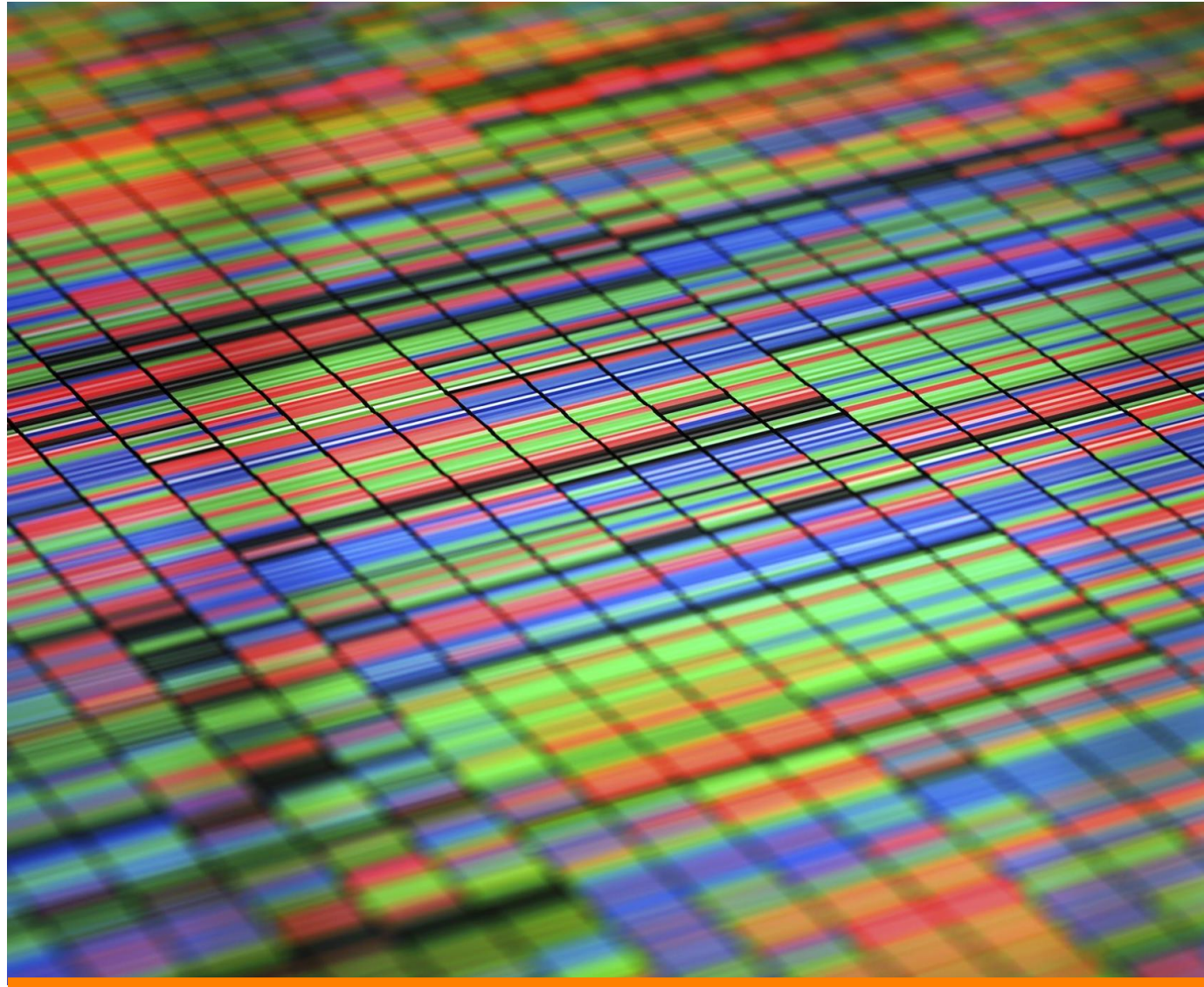
Algoritmo del método de inserción

Insertion-Sort (A)

1. **For** $j \leftarrow 2$ to n
2. **do** $chave \leftarrow A[j]$
3. $i \leftarrow j - 1$
4. **While** $i > 0$ e $A[i] > chave$
5. **do** $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i - 1$
7. $A[i+1] \leftarrow chave$

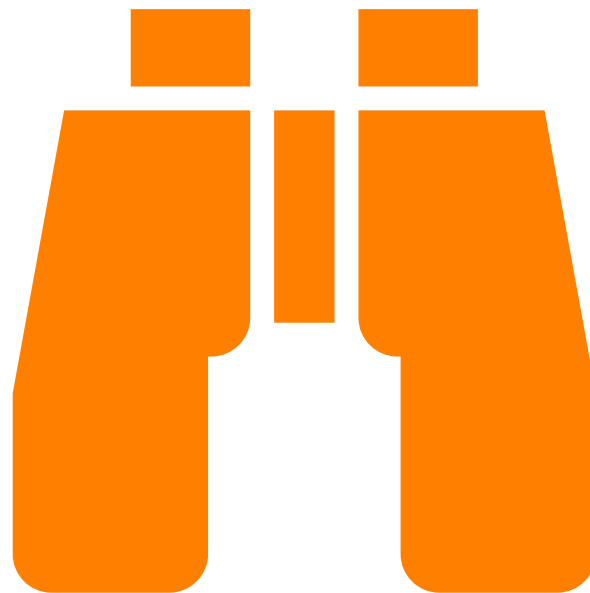
Merge Sort

- La ordenación por combinación es un algoritmo de ordenación que divide una lista en segmentos de un elemento y, a continuación, fusiona esos segmentos en una lista ordenada.
- Es un algoritmo de divide y vencerás que es de naturaleza recursiva.
- La ordenación por fusión funciona dividiendo una matriz de entrada por la mitad, luego ordenando cada mitad y fusionando las dos mitades.



Algoritmo de ordenamiento Quick Sort

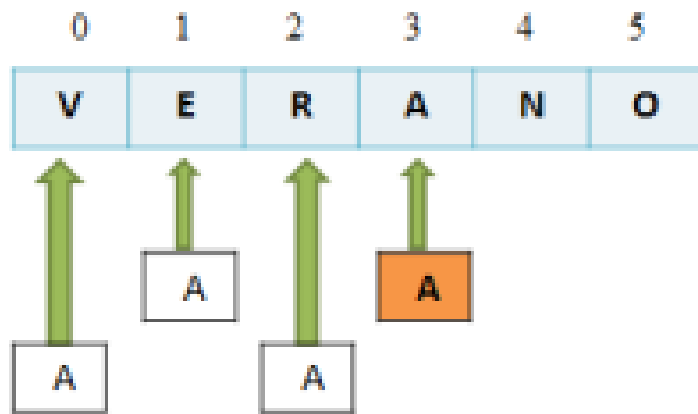
- Quick Sort es un algoritmo de ordenamiento muy veloz y eficiente.
- El algoritmo Quick Sort utiliza la técnica Divide y Vencerás.
- Quick Sort es un algoritmo de ordenamiento in-place, lo que significa que no se requiere memoria adicional.



BÚSQUEDA



Búsqueda secuencial o lineal



INDICE=3

- En este tipo de búsqueda, se realiza una búsqueda secuencial de todos los elementos uno por uno. Se verifica cada elemento y si se encuentra una coincidencia, se devuelve ese elemento en particular; de lo contrario, la búsqueda continúa hasta el final de la estructura de datos.

Búsqueda de interpolación



Este algoritmo de búsqueda funciona en la posición de sondeo del valor requerido.

Para que este algoritmo funcione correctamente, la recopilación de datos debe estar ordenada y distribuida equitativamente.

Inicialmente, la posición de la sonda es la posición del elemento más intermedio de la colección.

Si se produce una coincidencia, se devuelve el índice del elemento. Si el elemento central es mayor que el elemento, la posición de la sonda se calcula nuevamente en la matriz secundaria a la derecha del elemento central. De lo contrario, el elemento se busca en la submatriz a la izquierda del elemento central.

Este proceso también continúa en la submatriz hasta que el tamaño de la submatriz se reduce a cero.

Búsqueda binaria



En la búsqueda binaria tomamos una lista ordenada de elementos y comenzamos a buscar un elemento en el medio de la lista.



Si el valor de búsqueda coincide con el valor medio en la lista, completamos la búsqueda. De lo contrario, eliminamos la mitad de la lista de elementos eligiendo si procede con la mitad derecha o izquierda de la lista según el valor del elemento buscado.



Esto es posible ya que la lista está ordenada y es mucho más rápida que la búsqueda lineal.



Aquí dividimos la lista dada y conquistamos eligiendo la mitad adecuada de la lista.

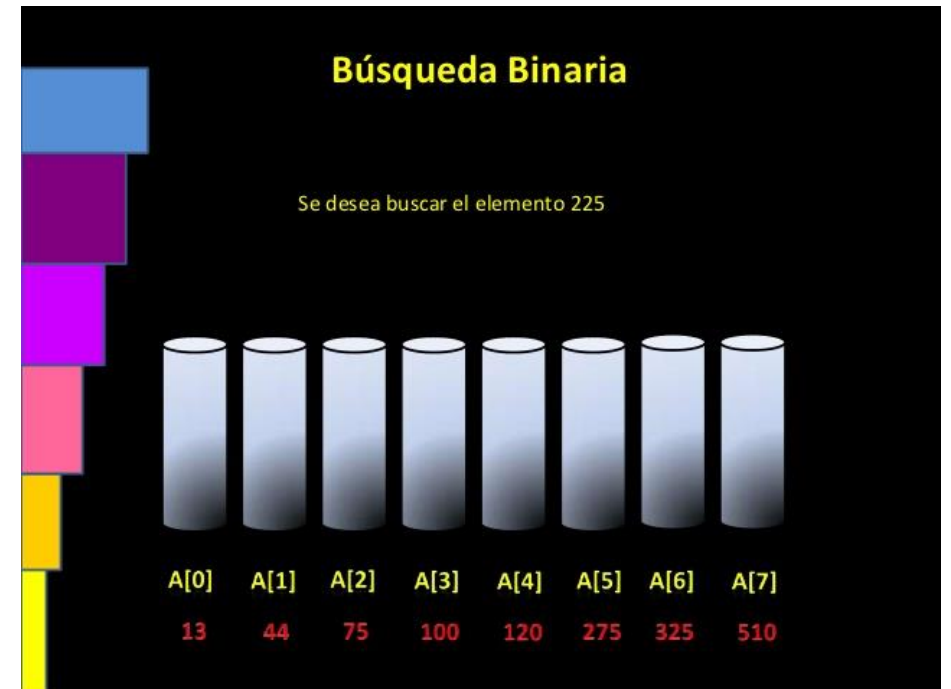


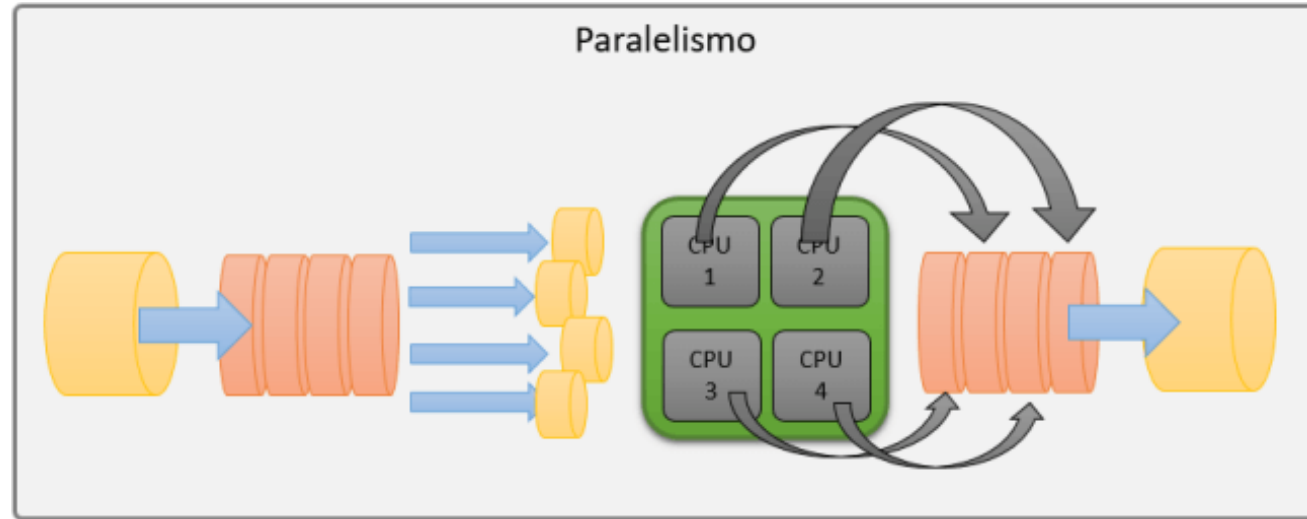
Repetimos esta aproximación hasta que encontremos el elemento o concluyamos sobre su ausencia en la lista.



Búsqueda binaria usando recursividad

- Usamos una lista ordenada de elementos y diseñamos una función recursiva para incluir la lista junto con el índice inicial y final como entrada.
- Luego, la función de búsqueda binaria se llama a sí misma hasta encontrar el elemento buscado o concluye sobre su ausencia en la lista.





PARALELISMO

Paralelismo

- El Paralelismo en la informática, es una función que realiza el procesador para ejecutar varias tareas al mismo tiempo. Es decir, puede realizar varios cálculos simultáneamente, basado en el principio de dividir los problemas grandes para obtener varios problemas pequeños, que son posteriormente solucionados en paralelo.
- El empleo de la computación paralela se convierte cada día en más grande y rápida, muchos problemas considerados anteriormente muy largos y costosos se han podido solucionar. El paralelismo se ha utilizado para muchas temáticas diferentes, desde bioinformática para hacer plegamiento de proteínas, hasta economía para hacer simulaciones en matemática financiera.



Paralelismo

- En Python el procesamiento en paralelo puede aumentar la cantidad de tareas realizadas por su programa, lo que reduce el tiempo de procesamiento general.
- Estos ayudan a manejar problemas a gran escala



Paralelismo

