alpha ed/tech

ESTRUTURA DE DADOS

MÓDULO 07 | AULA 12

alpha
<ed/tech>

ORDENAÇÃO

A ordenação e busca são operações fundamentais em ciência da computação, com vastas aplicações no processamento e análise de dados. Esta aula abordará técnicas avançadas de ordenação e busca, explorando algoritmos eficientes e suas aplicações em problemas do mundo real.

Ordenação Avançada

Algoritmos de Ordenação

- **QuickSort:** Um algoritmo de divisão e conquista que seleciona um elemento como pivô e particiona o array em torno do pivô.
- **MergeSort:** Também baseado em divisão e conquista, divide o array em metades, ordena cada metade e depois mescla as metades ordenadas.
- **HeapSort:** Utiliza uma estrutura de dados heap para organizar os elementos do array e extrair o maior ou menor elemento repetidamente.

Comparação de Algoritmos

- Análise de complexidade de tempo e espaço.
- Casos de uso ideal para cada algoritmo.
- Vantagens e desvantagens.

alpha <ed/tech>

ORDENAÇÃO

QUICKSORT

QuickSort é um algoritmo de ordenação eficiente que segue o princípio de divisão e conquista. Ele escolhe um elemento do array para ser o pivô e particiona o array em torno desse pivô, de modo que os elementos menores que o pivô são movidos para antes dele e os elementos maiores são movidos para depois. Esse processo é repetido recursivamente para as sub-partes do array até que o array esteja completamente ordenado.

Etapas do QuickSort

1. **Escolha do Pivô:** Selecionar um elemento do array como pivô. A escolha do pivô pode afetar o desempenho do algoritmo, e diferentes estratégias podem ser usadas, como escolher o primeiro elemento, o último, o mediano ou um elemento aleatório.
2. **Particionamento:** Reorganizar o array de forma que todos os elementos menores que o pivô venham antes dele, e todos os elementos maiores venham depois. Ao final do particionamento, o pivô estará em sua posição final ordenada no array.
3. **Recursão:** Aplicar recursivamente o mesmo processo aos sub-arrays formados pelos elementos à esquerda e à direita do pivô.

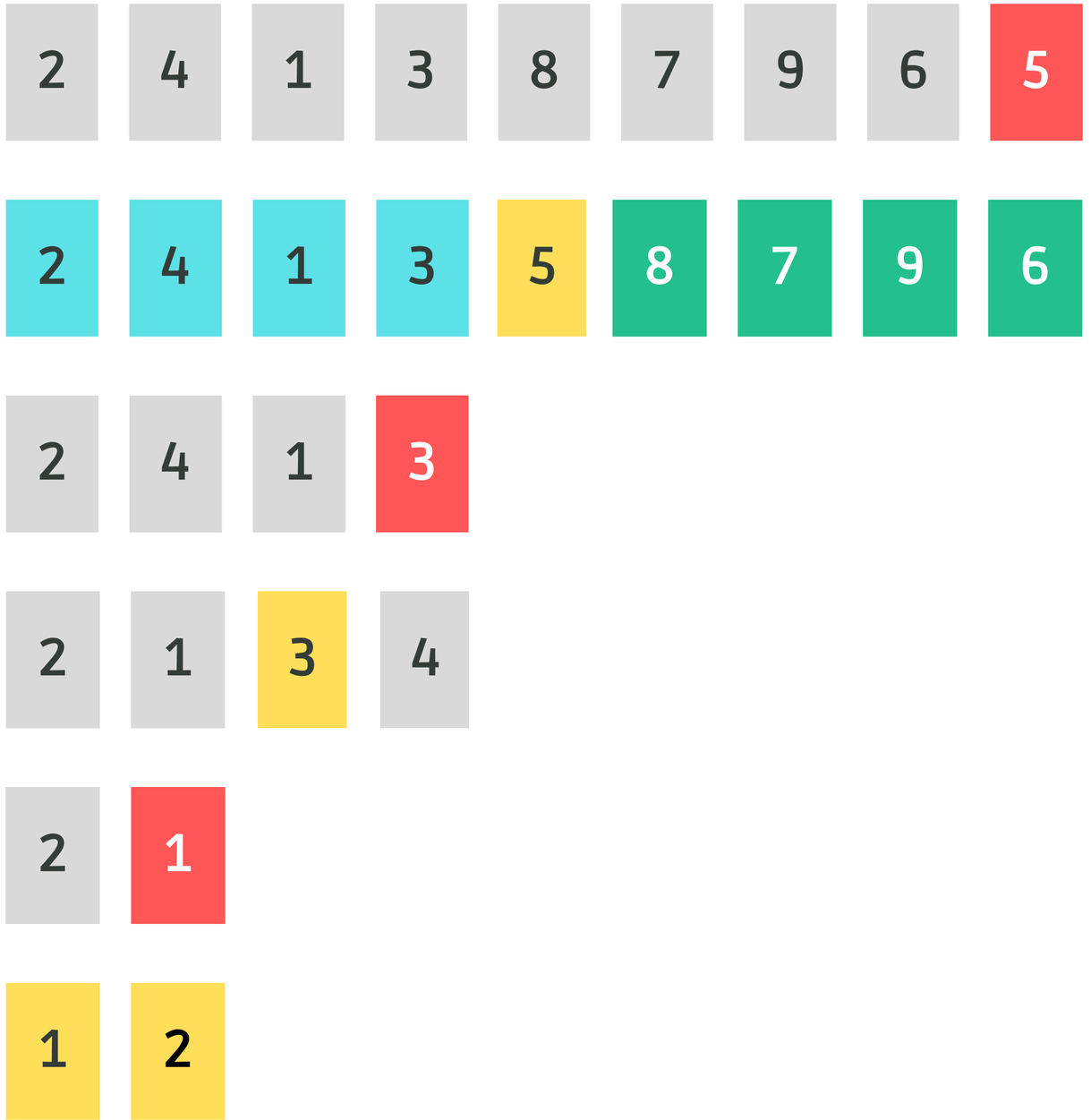
alpha <ed/tech>

ORDENAÇÃO

QUICKSORT

Complexidade

- **Melhor caso:** O(n log n), quando o particionamento divide o array de maneira equilibrada.
- **Pior caso:** O(n^2), quando o particionamento resulta em um lado muito maior que o outro a cada passo, como quando o array já está ordenado ou o pivô é sempre o menor ou maior elemento.
- **Caso médio:** O(n log n), o que faz do QuickSort uma escolha eficiente para muitas aplicações.


2 4 1 3 8 7 9 6 5
2 4 1 3 5 8 7 9 6
2 4 1 3
2 1 3 4
2 1
1 2

O código utiliza compreensões de lista para criar os sub-arrays menores e maiores que são, então, ordenados recursivamente. O elemento na posição zero do array é escolhido como pivô para simplicidade, mas em implementações mais complexas, outras estratégias de escolha do pivô podem ser adotadas (neste exemplo utilizamos o último elemento do array) para otimizar o desempenho em diferentes conjuntos de dados.

alpha
<ed/tech>

ORDENAÇÃO

MERGESORT

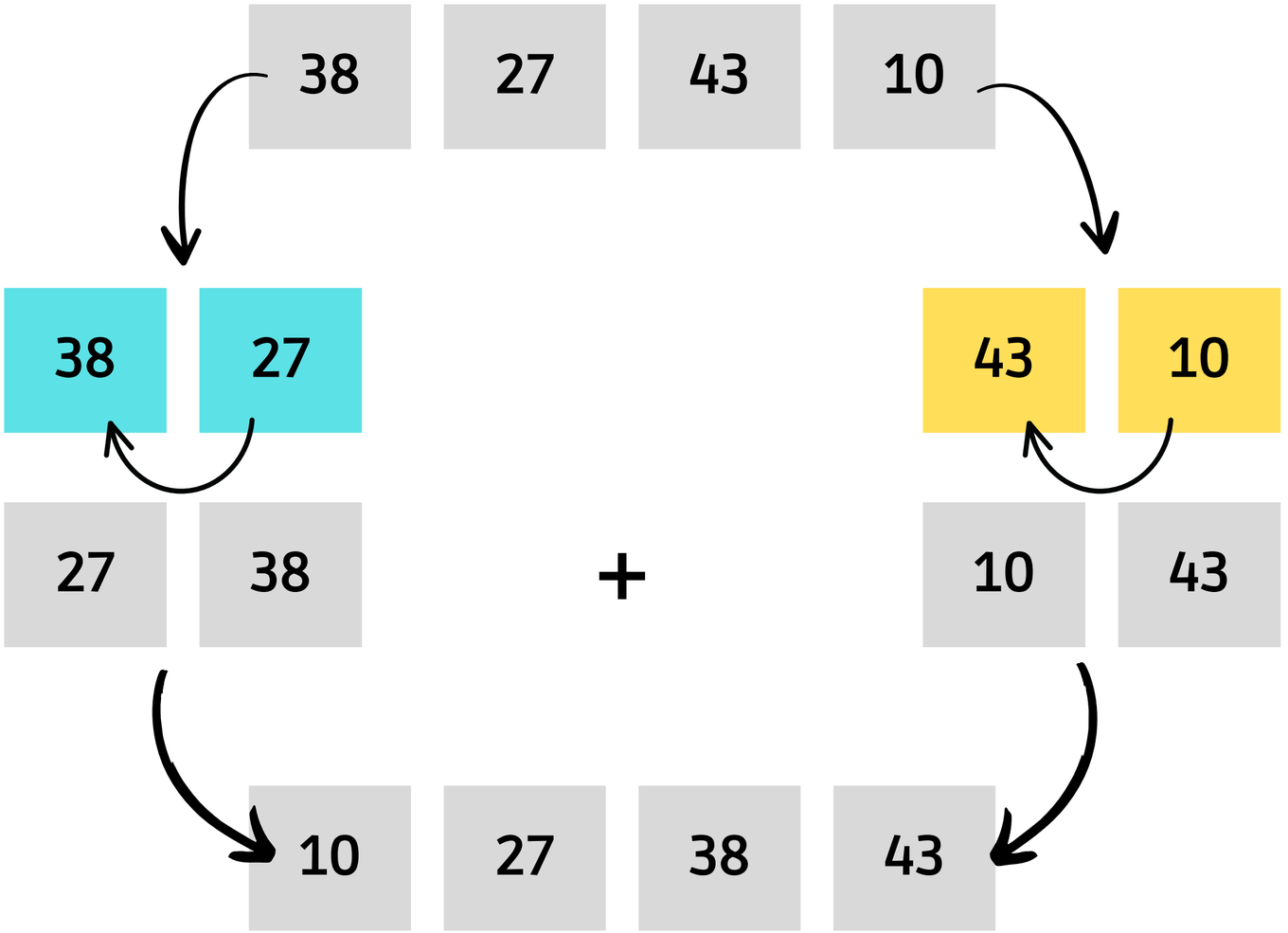
MergeSort é um algoritmo de ordenação que utiliza o princípio de divisão e conquista para ordenar elementos de um array ou lista. O processo envolve dividir a estrutura de dados original em partes menores, ordenar essas partes (se necessário) e, por fim, mesclar as partes ordenadas em uma única estrutura ordenada.

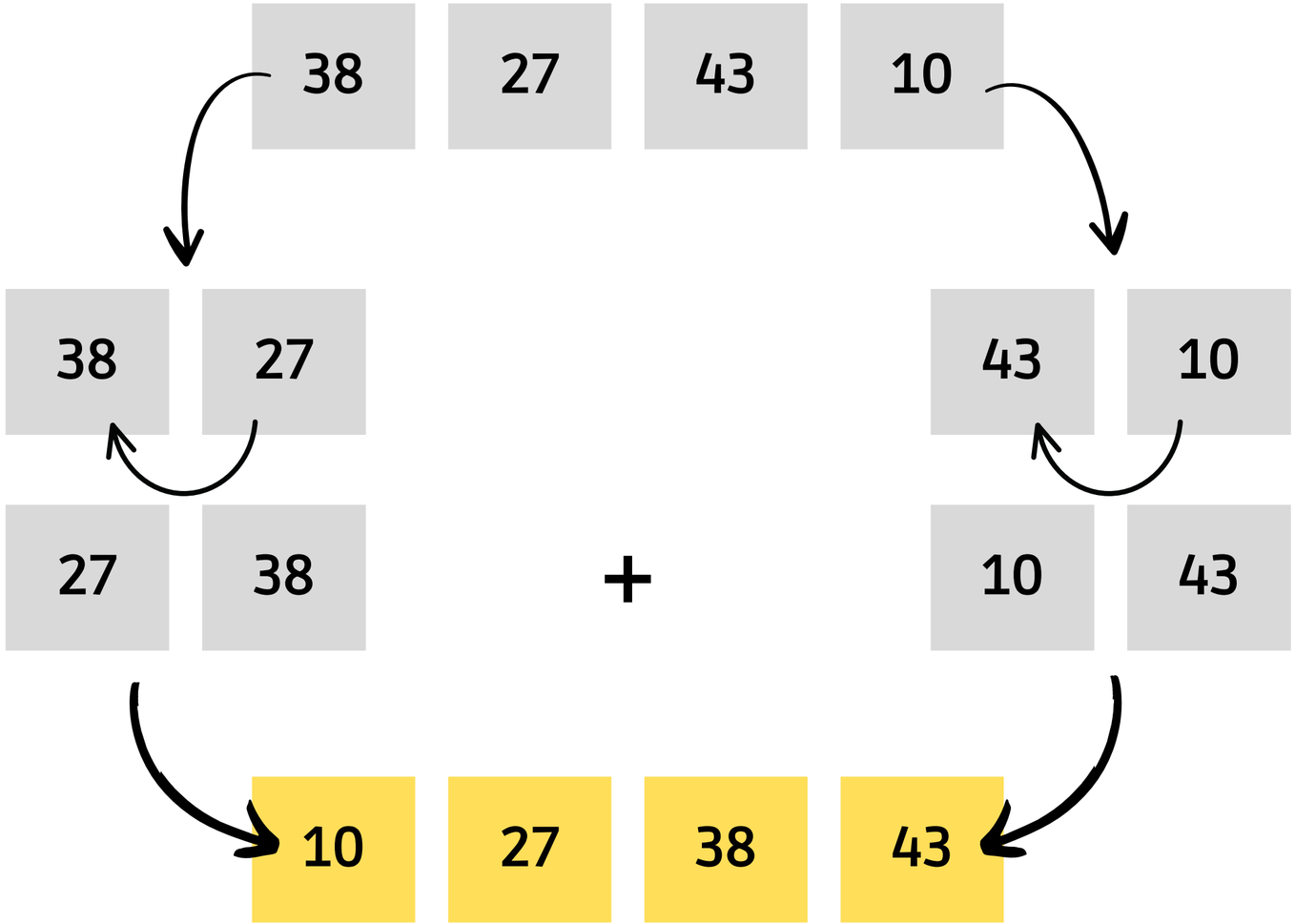
Etapas do MergeSort

1. **Divisão:** O array é dividido em duas metades até que cada sub-array contenha um único elemento ou nenhum elemento (o que naturalmente o torna ordenado).
2. **Conquista:** As metades são recursivamente ordenadas utilizando o MergeSort.
3. **Combinação:** As metades ordenadas são então combinadas (ou mescladas) em uma única estrutura ordenada.

Complexidade

- **Tempo:** O(n log n) em todos os casos (melhor, médio e pior), tornando-o eficiente e previsível em termos de tempo de execução.
- **Espaço:** O(n), devido à necessidade de armazenar arrays temporários durante a mesclagem.


38
27
43
10
38
27
43
10
27
38
+
10
43
10
27
38
43



A principal vantagem do MergeSort é sua eficiência e previsibilidade em termos de desempenho, funcionando bem em uma variedade de conjuntos de dados, independentemente da ordem inicial dos elementos. Contudo, seu ponto de atenção é o uso adicional de memória, necessário para os arrays temporários durante a etapa de mesclagem.

O MergeSort é especialmente útil em situações onde a previsibilidade do tempo de execução é crítica, como em aplicações de tempo real, ou quando trabalha-se com grandes conjuntos de dados que não cabem inteiramente na memória (ordenando-se por meio de acesso a disco, por exemplo).

alpha
<ed/tech>

ORDENAÇÃO

HEAPSORT

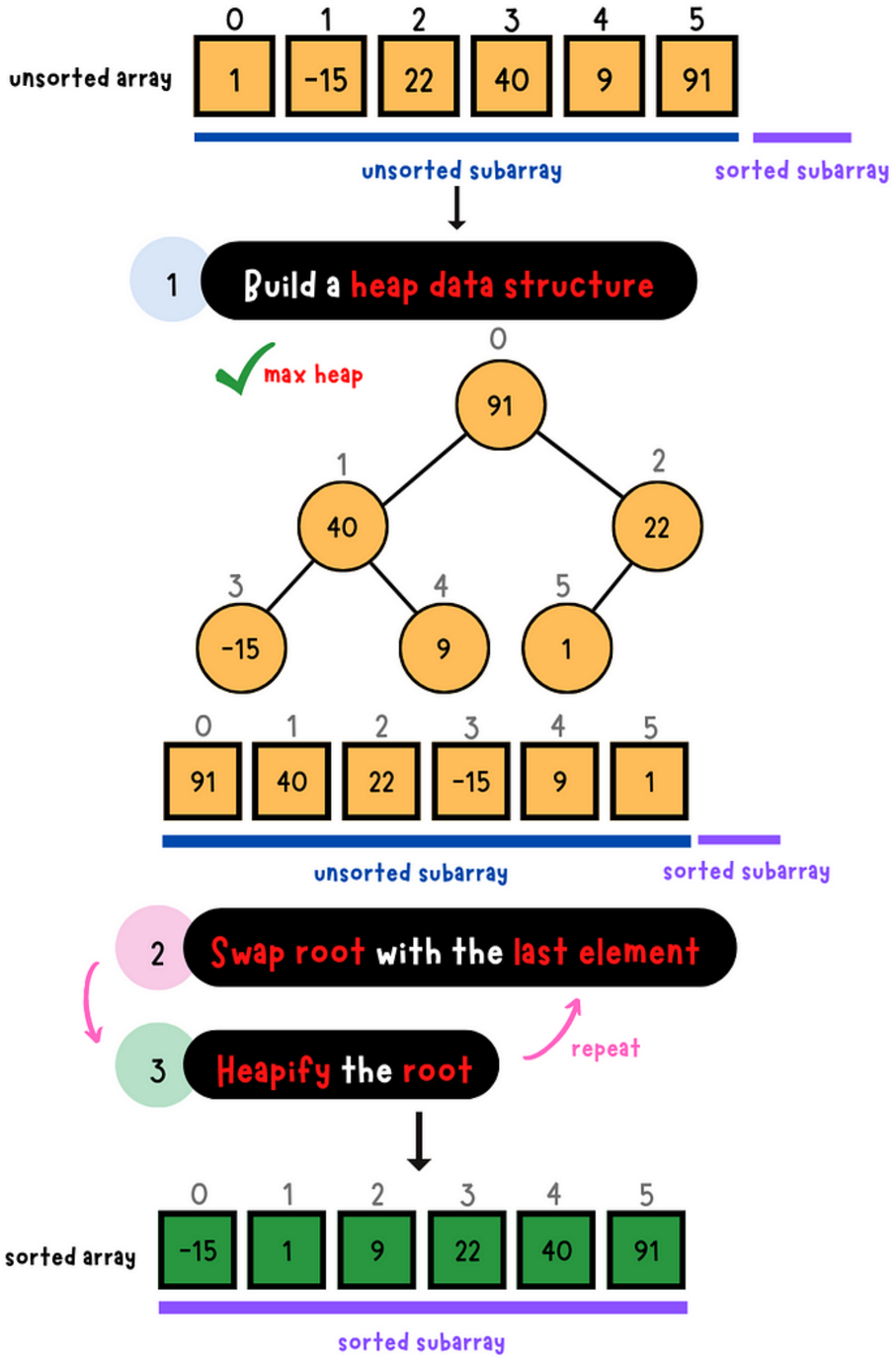
HeapSort é um algoritmo de ordenação eficiente que utiliza uma estrutura de dados chamada heap para organizar os elementos do array. Um heap é uma árvore binária completa onde cada nó pai é menor (min heap) ou maior (max heap) que seus nós filhos. O HeapSort aproveita as propriedades do max heap para ordenar elementos em ordem crescente (ou min heap para ordem decrescente).

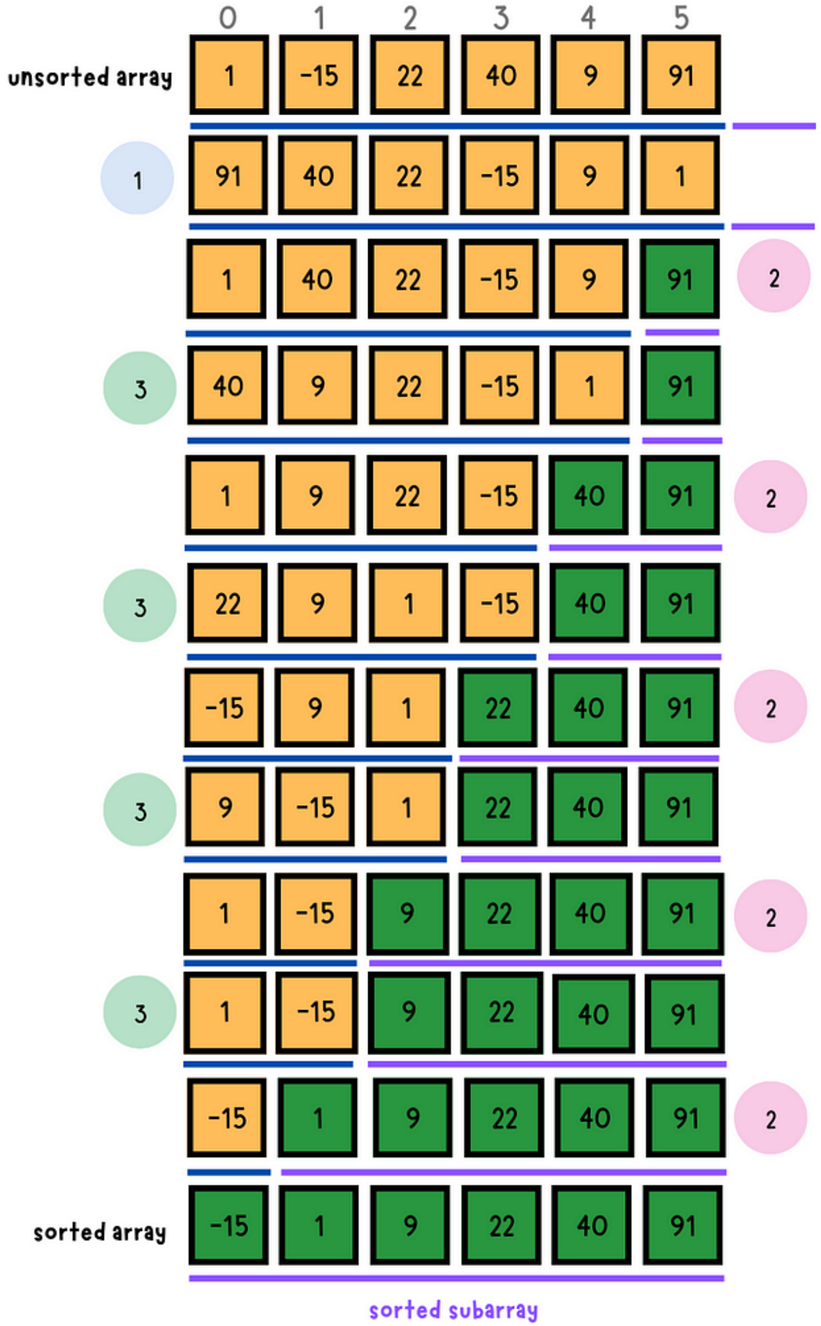
Etapas do HeapSort

1. **Construção do Heap:** Transformar o array em um max heap, onde o maior elemento é o root do heap.
2. **Ordenação:** Remover repetidamente o elemento máximo do heap (root), colocá-lo na parte ordenada do array e reajustar o heap para manter sua propriedade de max heap. Isso é feito até que todos os elementos sejam removidos do heap e inseridos no array em ordem.

Complexidade

- **Tempo:** O(n log n) tanto no melhor quanto no pior caso, tornando-o bastante eficiente para grandes conjuntos de dados.
- **Espaço:** O(1) em termos de espaço extra, pois o HeapSort pode ser realizado in-place no array de entrada.


unsorted array
0 1 2 3 4 5
1 -15 22 40 9 91
unsorted subarray
sorted subarray
1 Build a heap data structure
max heap
0 91
1 40
2 22
3 -15
4 9
5 1
0 1 2 3 4 5
91 40 22 -15 9 1
unsorted subarray
sorted subarray
2 Swap root with the last element
3 Heapify the root
repeat
sorted array
0 1 2 3 4 5
-15 1 9 22 40 91
sorted subarray


0 1 2 3 4 5
unsorted array
1 -15 22 40 9 91
1 91 40 22 -15 9 1
1 40 22 -15 9 91 2
3 40 9 22 -15 1 91
1 9 22 -15 40 91 2
3 22 9 1 -15 40 91
-15 9 1 22 40 91 2
3 9 -15 1 22 40 91
1 -15 9 22 40 91 2
3 1 -15 9 22 40 91
-15 1 9 22 40 91 2
sorted array
-15 1 9 22 40 91
sorted subarray