Ordenação

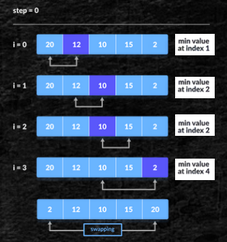
Brute Force — É uma abordagem em que utilizamos da definição e descrição do problema e de conceitos teóricos para solucionar problemas de forma mais direta. É um modo mais simples, geralmente.

Decrease-and-Conquer — Envolve a divisão de um problema em instâncias menores, resolvendo cada instância menor e, em seguida, combinando as soluções para resolver o problema original. Isso é feito reduzindo gradualmente o tamanho do problema até que seja resolvido.

Divide-and-Conquer — é uma estratégia de resolução de problemas que divide um problema complexo em subproblemas menores, resolve cada subproblema de forma independente e, em seguida, combina as soluções dos subproblemas para obter a solução do problema original. Geralmente, essa abordagem é implementada de forma recursiva, dividindo o problema em partes cada vez menores até chegar a casos base facilmente resolúveis.

Selection Sort

Utilizando busca linear, localizamos o menor elemento da lista. Este elemento trocará de lugar com o elemento que estiver mais a esquerda. O processo se repete até todos os elementos estarem ordenados.



Pseudocódigo

```
ALGORITHM SelectionSort(A[0..n − 1])
//Sorts a given array by selection sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 0 to n − 2 do
min ← i
for j ← i + 1 to n − 1 do
if A[j ] < A[min] min ← j
swap A[i] and A[min]
```

Número de Execuções

sendo n a quantidade de elementos

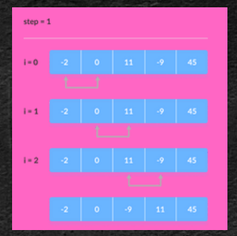
$$C(n) = \sum_{i=0}^{n-2} (n-1-i)$$

Bubble Sort

Começamos da direita do array. Comparamos o número mais a direita com o que está a sua esquerda. O menor número é colocado à esquerda (eles trocam de posição).

Pior Caso $S(worst) = C(n) = \frac{(n-1) * n}{2} \in$ big O (n^2)

O número de trocas é igual ao de comparações/execções



Pseudocódigo

```
ALGORITHM BubbleSort(A[0..n − 1])
//Sorts a given array by bubble sort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 0 to n − 2 do
    for j ← 0 to n − 2 − i do
        if A[j + 1] < A[j ] swap A[j ] and A[j + 1]
```

Número de Execuções

sendo n a quantidade de elementos

$$C(n) = \frac{(n-1) * n}{2}$$

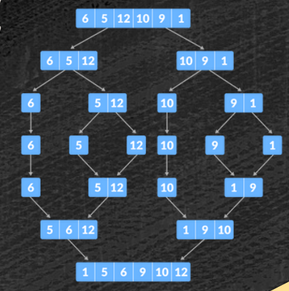
Merge Sort

Pseudocódigo

```
ALGORITHM Mergesort(A[0..n − 1])
//Sorts array A[0..n − 1] by recursive mergesort
//Input: An array A[0..n − 1] of orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
if n > 1
copy A[0..⌊n/2⌋ − 1] to B[0..⌊n/2⌋ − 1]
copy A[⌊n/2⌋..n − 1] to C[0..⌈n/2⌉ − 1]
Mergesort(B[0..⌊n/2⌋ − 1])
Mergesort(C[0..⌈n/2⌉ − 1])
Merge(B, C, A) //see below

ALGORITHM Merge(B[0..p − 1], C[0..q − 1], A[0..p + q − 1])
//Merges two sorted arrays into one sorted array
//Input: Arrays B[0..p − 1] and C[0..q − 1] both sorted
//Output: Sorted array A[0..p + q − 1] of the elements of B and C
i ← 0; j ← 0; k ← 0
while i<p and j<q do
if B[i] ≤ C[j ]
A[k]← B[i]; i ← i + 1
else A[k]← C[ j ]; j ← j + 1
k ← k + 1
if i = p
copy C[j..q − 1] to A[k..p + q − 1]
else copy B[i..p − 1] to A[k..p + q − 1]
```

Baseado na técnica "dividir para conquistar", ele vai dividir o array dado em duas metades, ordenando cada uma delas de forma recursiva e, em seguida, mesclando as duas metades menores em um único array ordenado.



Número de Execuções

$C(n) = 2C(n/2) + Cmerge(n)$ for $n > 1$, $C(1) = 0$.

$Cworst(n) = n \log_2 n - n + 1$.

Pior Caso

Inserction Sort

O processo é iterativo e começa com o segundo elemento do array. Você compara esse elemento com o primeiro e os reorganiza se necessário. Em seguida, você compara o terceiro elemento com os dois primeiros e os reorganiza, e assim por diante, até que todo o array esteja ordenado. É um algoritmo eficiente para arrays pequenos, mas pode ser lento em grandes conjuntos de dados.

Pseudocódigo

```
ALGORITHM InsertionSort(A[0..n − 1])
//Sorts a given array by insertion sort
//Input: An array A[0..n − 1] of n orderable elements
//Output: Array A[0..n − 1] sorted in nondecreasing order
for i ← 1 to n − 1 do
v ← A[i]
j ← i − 1
while j ≥ 0 and A[j ] > v do
A[j + 1]← A[j ]
j ← j − 1
A[j + 1] ← v
```

Pior caso C worst (n) = $\frac{(n-1) * n}{2}$

Caso médio Cmédio(n) ≈ $n^2/4$

Melhor caso C best (n) = n - 1