Fatec
Rio Claro

Estrutura de Dados

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Classificação
(*Sorting*)

Fatec
Rio Claro

Classificar significa reorganizar dados de forma que sejam em ordem ascendente ou decrescente. A classificação é um dos algoritmos mais importantes da ciência da computação e é amplamente utilizada em algoritmos relacionados ao banco de dados.

Para vários aplicativos, se os dados forem classificados, poderão ser recuperados com eficiência, por exemplo, se for uma coleção de nomes, números de telefone ou itens em uma lista de tarefas simples.

Prática

Fatec
Rio Claro

Observe o código **ordenacao.py**

Sorting

Algoritmos de Classificação

Fatec
Rio Claro

Classificação significa organizar todos os itens em uma lista em ordem ascendente ou decrescente. Podemos comparar diferentes algoritmos de classificação com quanto tempo e espaço de memória são necessários para usá -los.

O tempo gasto por um algoritmo muda, dependendo do tamanho da entrada. Além disso, alguns algoritmos são relativamente fáceis de implementar, mas podem ter um desempenho ruim em relação à complexidade do tempo e do espaço, enquanto outros algoritmos são um pouco mais complexos para implementar, mas podem ter um bom desempenho ao classificar listas mais longas de dados.

Merge Sort

Merge Sort

Fatec
Rio Claro

Merge sort é um algoritmo para ordenar uma lista de n números naturais em ordem crescente. Primeiramente, a lista de elementos fornecida é dividida iterativamente em partes iguais até que cada sublista contenha um elemento, e então essas sublistas são combinadas para criar uma nova lista em ordem ordenada.

Merge Sort

Fatec
Rio Claro

Essa abordagem de programação para resolução de problemas baseia-se na metodologia de dividir para conquistar e enfatiza a necessidade de decompor um problema em subproblemas menores do mesmo tipo ou formato do problema original. Esses subproblemas são resolvidos separadamente e, em seguida, os resultados são combinados para obter a solução do problema original.

Merge Sort

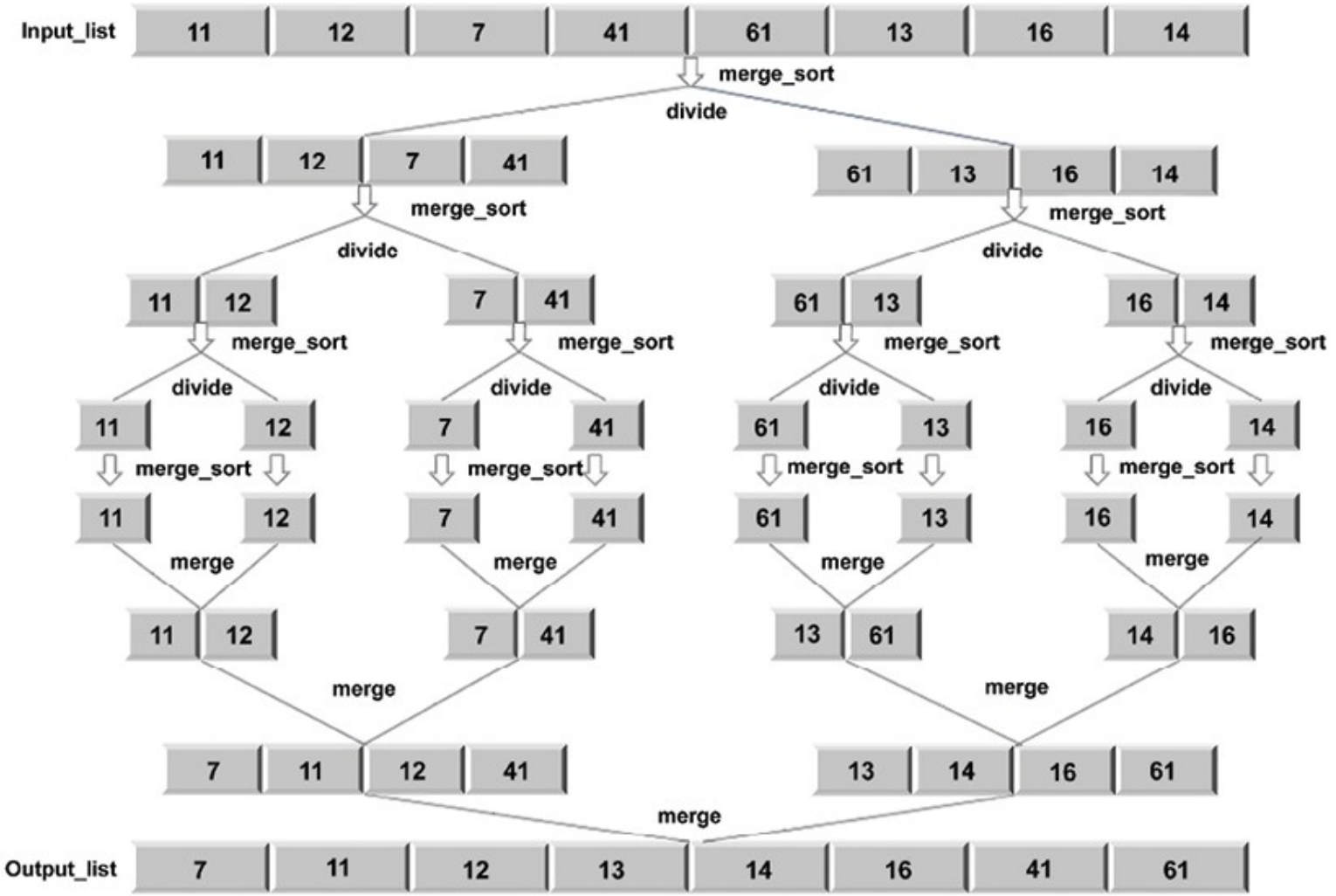
Fatec
Rio Claro

Neste caso, dada uma lista de elementos não ordenados, dividimos a lista em duas metades aproximadas. Continuamos a dividir a lista em metades recursivamente.

Após um tempo, a sublista criada como resultado da chamada recursiva conterá apenas um elemento. Nesse ponto, começamos a mesclar as soluções na etapa de conquista ou mesclagem.

Merge Sort

Fatec
Rio Claro

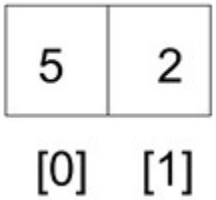


Bubble Sort

Bubble Sort

Fatec
Rio Claro

Vamos fazer uma lista com apenas dois elementos, {5, 2}, para entender o conceito de tipo de bolha:

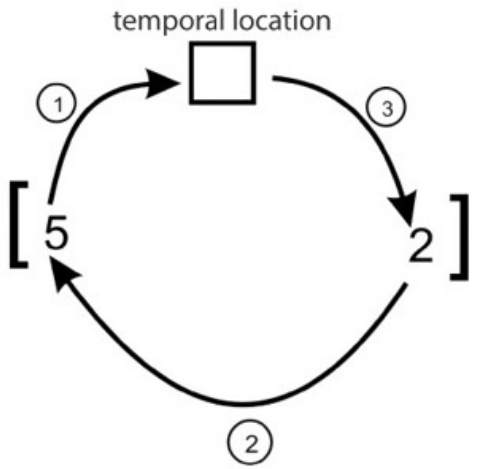


Para classificar esta lista de dois elementos, primeiro, comparamos 5 e 2; Como 5 é maior que 2, isso significa que eles não estão na ordem correta, então trocamos esses valores para colocá -los na ordem correta.

Bubble Sort

Fatec
Rio Claro

Para trocar esses dois números, primeiro, movemos o elemento armazenado no índice 0 em uma variável temporária, então o elemento armazenado no índice 1 é copiado para o índice 0 e, finalmente, o primeiro elemento armazenado na variável temporária é armazenado no índice 1



Bubble Sort

Fatec
Rio Claro

Vamos considerar outro exemplo para entender o funcionamento do algoritmo de classificação de bolhas e classificar uma lista não ordenada de seis elementos, como {45, 23, 87, 12, 32, 4}. Na primeira iteração, começamos a comparar os dois primeiros elementos, 45 e 23, e trocamos -os, pois 45 devem ser colocados após 23.

Bubble Sort

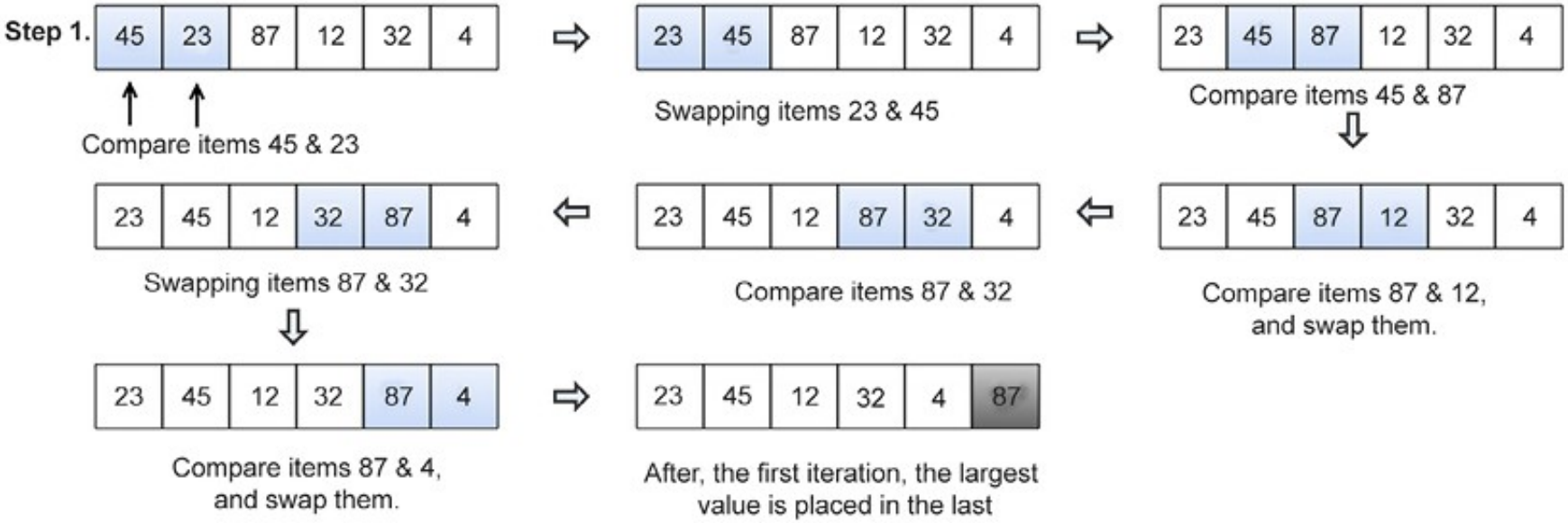
Fatec
Rio Claro

Em seguida, comparamos os próximos valores adjacentes, 45 e 87, para verificar se eles estão na ordem correta.

Como 87 é um valor mais alto que 45, não precisamos trocá-los. Trocamos dois elementos se eles não estiverem na ordem correta.

Bubble Sort

Fatec
Rio Claro



Bubble Sort

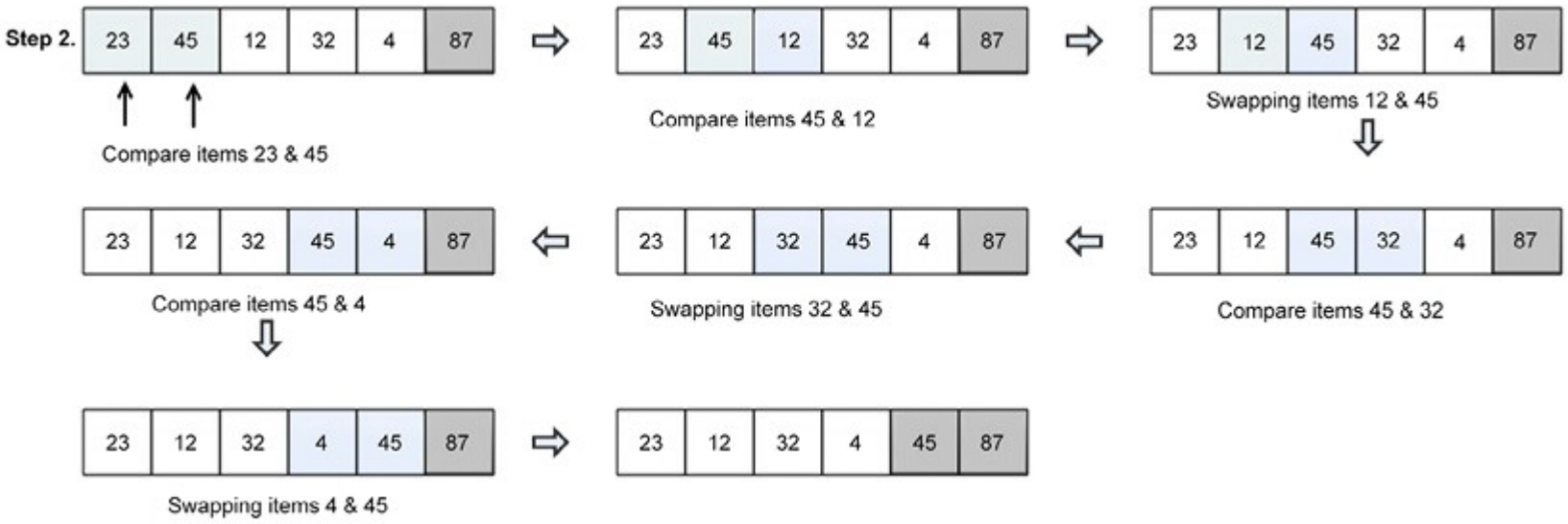
Fatec
Rio Claro

Após a primeira iteração, precisamos apenas organizar os elementos restantes (n-1); repetimos o mesmo processo comparando os elementos adjacentes para os cinco elementos restantes.

Após a segunda iteração, o segundo maior elemento, 45, é colocado na penúltima posição da lista, como mostrado na figura.

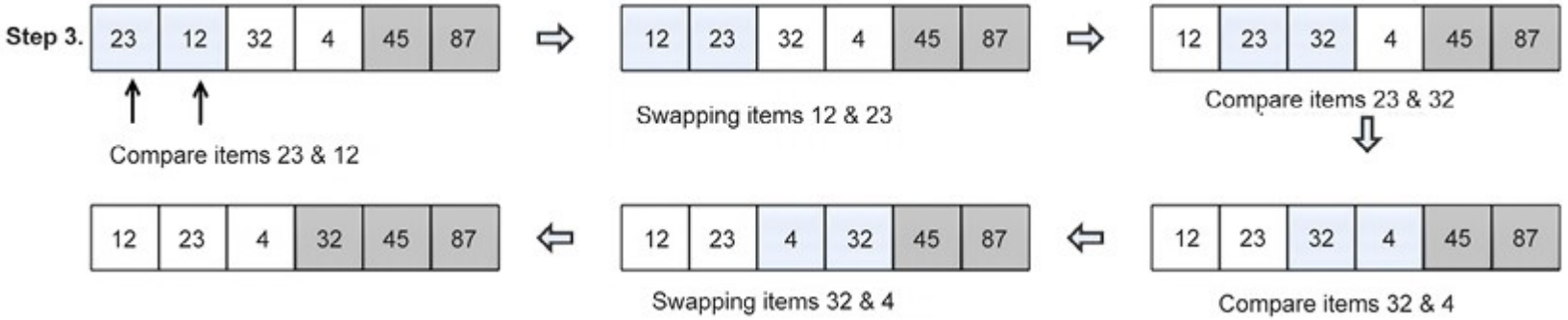
Bubble Sort

Fatec
Rio Claro



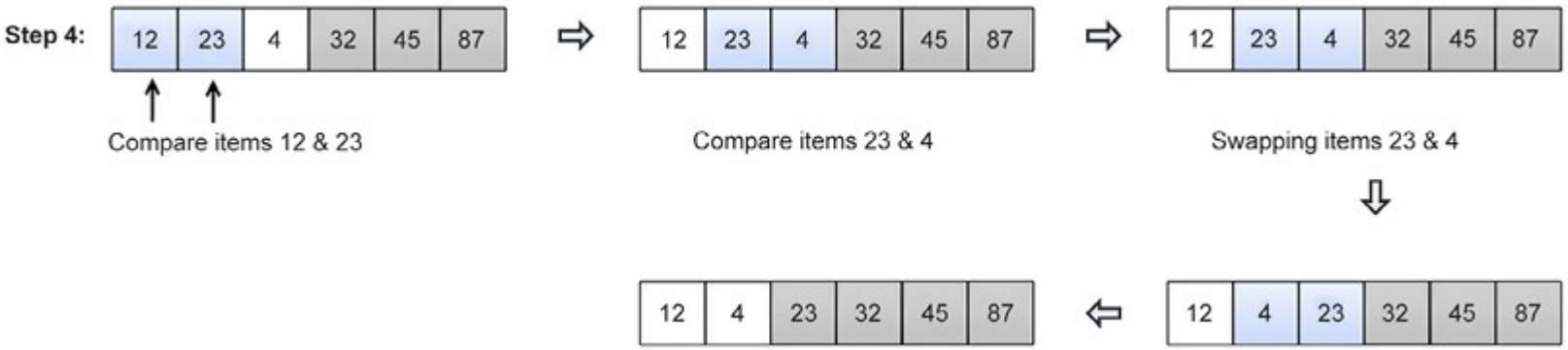
Bubble Sort

Fatec
Rio Claro



Bubble Sort

Fatec
Rio Claro



Bubble Sort

Fatec
Rio Claro

Bubble Sort

Fatec
Rio Claro

O algoritmo bubble sort não é um algoritmo de ordenação eficiente, pois fornece uma complexidade de tempo de execução de pior caso de $O(n^2)$ e uma complexidade de melhor caso de $O(n)$.

O pior caso ocorre quando queremos ordenar a lista fornecida em ordem crescente e a lista fornecida está em ordem decrescente, e o melhor caso ocorre quando a lista fornecida já está ordenada; nesse caso, não haverá necessidade de troca.

Bubble Sort

Fatec
Rio Claro

Geralmente, o algoritmo bubble sort não deve ser usado para ordenar listas grandes.

O algoritmo bubble sort é adequado para aplicações onde o desempenho não é importante ou o comprimento da lista fornecida é curto e, além disso, código curto e simples é preferível.

O algoritmo bubble sort tem bom desempenho em listas relativamente pequenas.

Prática

Fatec
Rio Claro

Observe o código **bubble_sort.py**

Insertion Sort

Insertion Sort

Fatec
Rio Claro

A ideia da ordenação por inserção é manter duas sublistas (uma sublista é parte da lista original maior), uma ordenada e outra não ordenada, nas quais os elementos são adicionados um a um da sublista não ordenada à sublista ordenada.

Assim, pegamos elementos da sublista não ordenada e os inserimos na posição correta na sublista ordenada, de forma que esta sublista permaneça ordenada.

Insertion Sort

Fatec
Rio Claro

Vamos considerar um exemplo para entender o funcionamento do algoritmo de ordenação por inserção. Digamos que precisamos ordenar uma lista de seis elementos: {45, 23, 87, 12, 32, 4}. Primeiramente, começamos com um elemento, supondo que ele esteja ordenado, e então pegamos o próximo elemento, 23, da sublista não ordenada e o inserimos na posição correta na sublista ordenada.

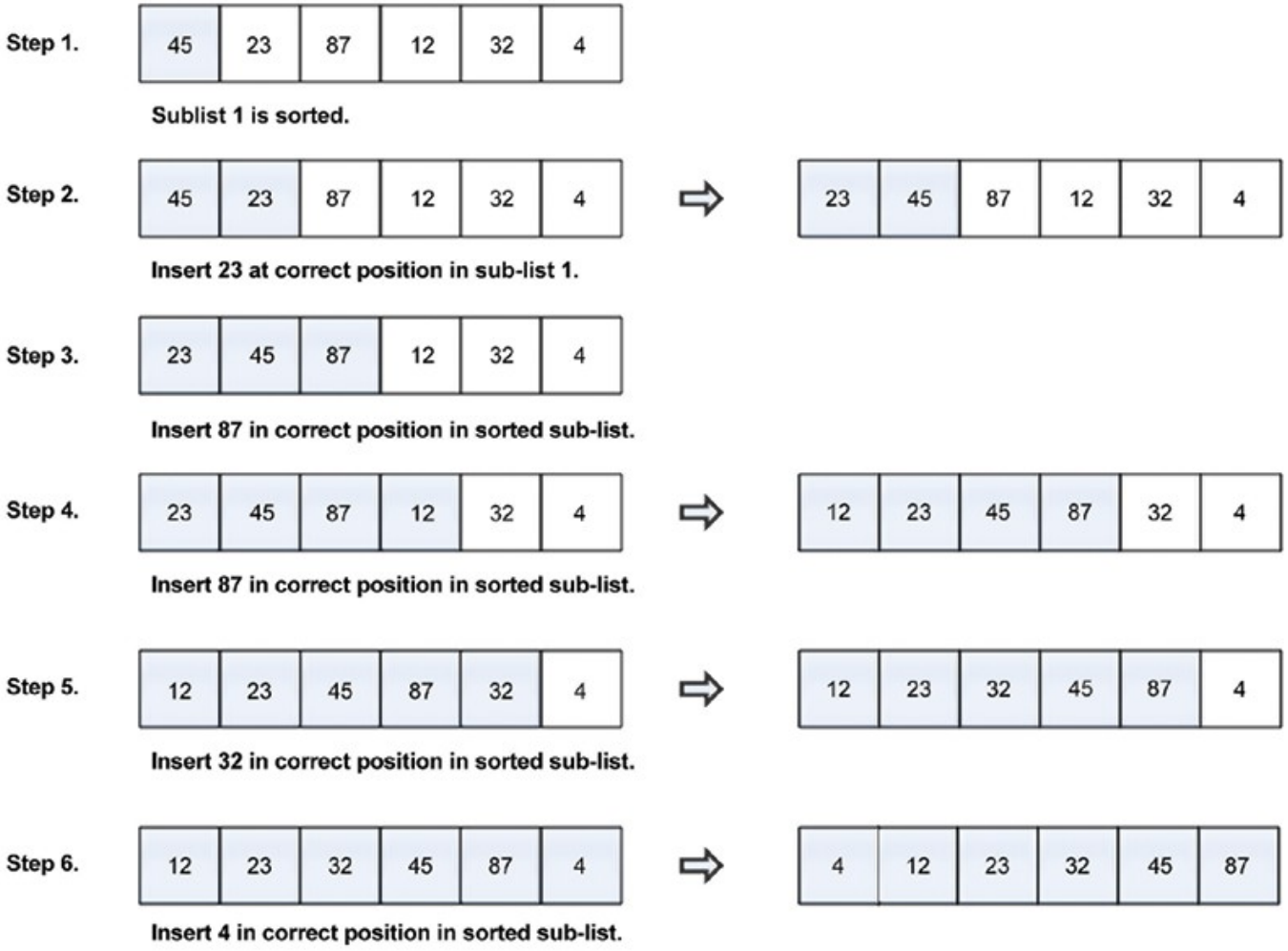
Insertion Sort

Fatec
Rio Claro

Na próxima iteração, pegamos o terceiro elemento, 87, da sublista não ordenada e o inserimos novamente na sublista ordenada na posição correta. Seguimos o mesmo processo até que todos os elementos estejam na sublista ordenada.

Insertion Sort

Fatec
Rio Claro



Insertion Sort

Fatec
Rio Claro

Para entender a implementação do algoritmo de ordenação por inserção, vamos pegar outro exemplo de cinco elementos, {5, 1, 100, 2, 10}, e examinar o processo com uma explicação detalhada.

O algoritmo começa usando um loop for para executar entre os índices 1 e 4. Começamos do índice 1 porque consideramos o elemento armazenado no índice 0 como estando no subarray ordenado e os elementos entre os índices 1 e 4 como sendo da sublista não ordenada.

Insertion Sort

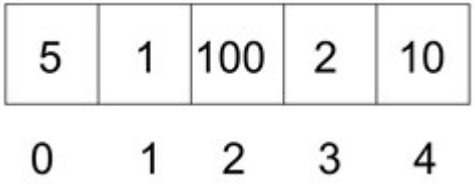
Fatec
Rio Claro

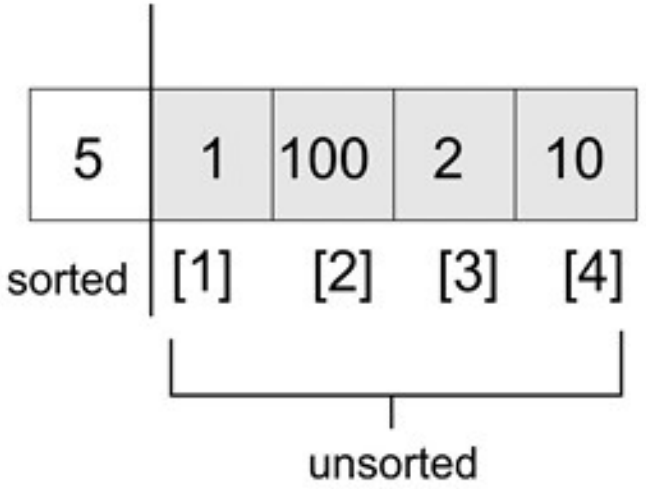
No início da execução de cada execução do loop for, o elemento em unsorted_list[index] é armazenado na variável insert_value.

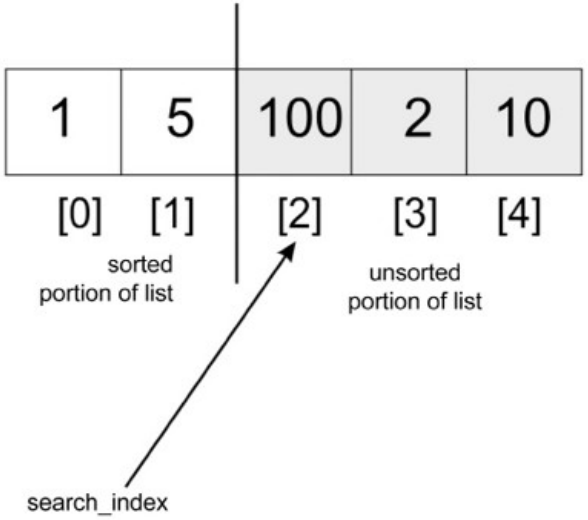
Posteriormente, quando encontrarmos a posição apropriada na parte ordenada da sublista, insert_value será armazenado nesse índice na sublista ordenada.

Insertion Sort

Fatec
Rio Claro







Selection Sort

Selection Sort

Fatec
Rio Claro

Outro algoritmo de ordenação popular é a ordenação por seleção.

O algoritmo de ordenação por seleção começa encontrando o menor elemento da lista e o troca com os dados armazenados na primeira posição da lista.

Assim, ele ordena a sublista até o primeiro elemento. Esse processo é repetido por (n-1) vezes para ordenar n itens.

Selection Sort

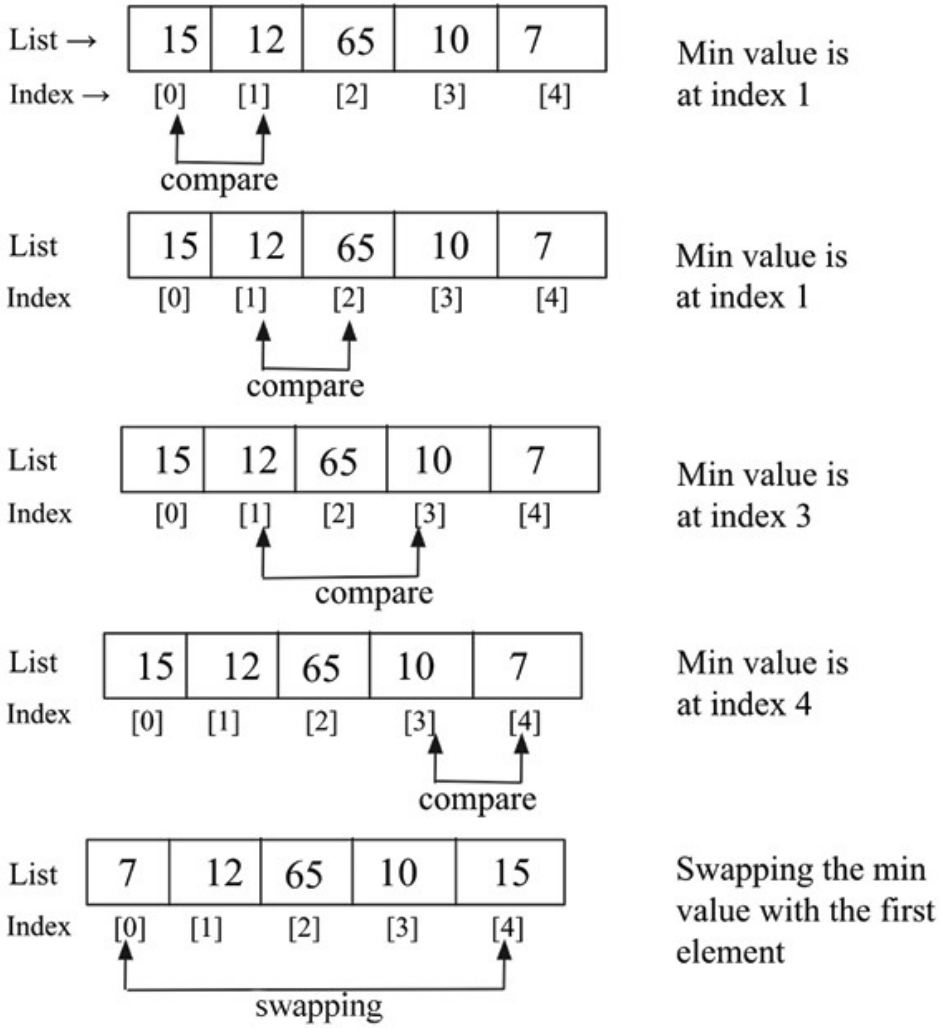
Fatec
Rio Claro

Em seguida, o segundo menor elemento, que é o menor elemento da lista restante, é identificado e trocado com a segunda posição da lista. Isso faz com que os dois elementos iniciais sejam ordenados.

O processo é repetido e o menor elemento restante na lista é trocado pelo elemento no terceiro índice da lista. Isso significa que os três primeiros elementos agora estão ordenados.

Selection Sort

Fatec
Rio Claro



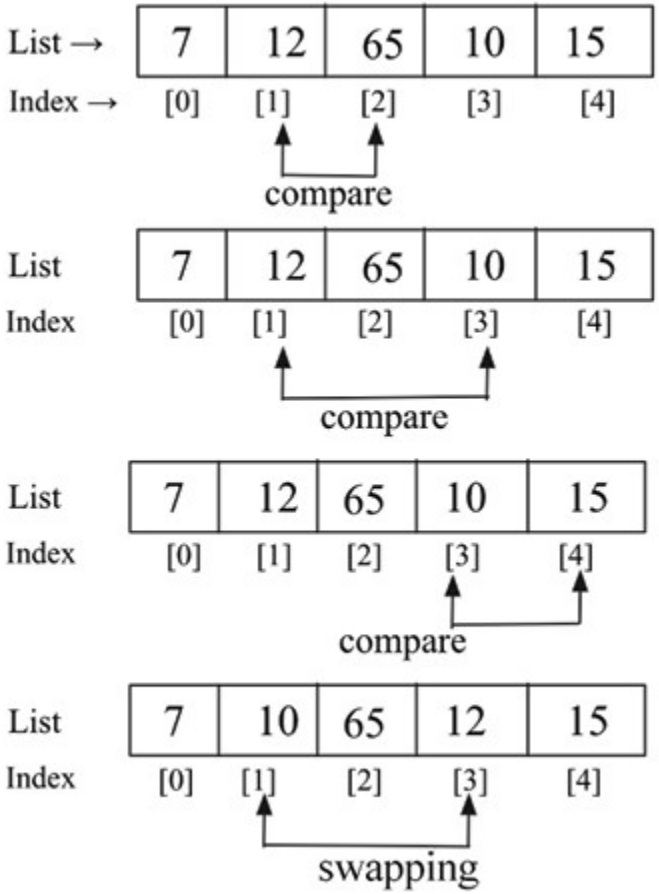
Selection Sort

Fatec
Rio Claro

Na primeira iteração da ordenação por seleção, começamos no índice 0, buscamos o menor item da lista e, quando o menor elemento é encontrado, ele é trocado pelo primeiro elemento de dados da lista no índice 0. Simplesmente repetimos esse processo até que a lista esteja completamente ordenada. Após a primeira iteração, o menor elemento será colocado na primeira posição da lista.

Selection Sort

Fatec
Rio Claro



Selection Sort

Fatec
Rio Claro

Na ordenação por seleção, (n-1) comparações são necessárias na primeira iteração, e (n-2) comparações são necessárias na segunda iteração, e (n-3) comparações são necessárias na terceira iteração, e assim por diante. Portanto, o número total de comparações necessárias é: (n-1) + (n-2) + (n-3) + ... + 1 = n(n-1) / 2, que é quase igual a n2.

Selection Sort

Fatec
Rio Claro

Assim, a complexidade de tempo do pior caso da ordenação por seleção é O(n2). A situação de pior caso é quando a lista de elementos fornecida é ordenada inversamente.

O algoritmo de ordenação por seleção fornece a complexidade de tempo de execução do melhor caso de O(n2).

O algoritmo de ordenação por seleção pode ser usado quando temos uma lista pequena de elementos.

QuickSort

QuickSort

Fatec
Rio Claro

O Quicksort é um algoritmo de ordenação eficiente. O algoritmo baseia-se na classe de algoritmos de divisão e conquista, semelhante ao algoritmo Merge Sort, em que dividimos um problema em partes menores, muito mais simples de resolver, e, além disso, os resultados finais são obtidos pela combinação das saídas de problemas menores (conquista).

QuickSort

Fatec
Rio Claro

O conceito por trás do Quicksort é o particionamento de uma determinada lista ou array. Para particionar a lista, primeiro selecionamos um elemento de dados da lista fornecida, chamado de elemento pivô.

QuickSort

Fatec
Rio Claro

Podemos escolher qualquer elemento como elemento pivô na lista. No entanto, para simplificar, usaremos o primeiro elemento do array como elemento pivô. Em seguida, todos os elementos da lista são comparados com este elemento pivô.

Ao final da primeira iteração, todos os elementos da lista são organizados de forma que os elementos menores que o elemento pivô sejam organizados à esquerda do pivô, e os elementos maiores que o elemento pivô sejam organizados à direita do pivô.

QuickSort

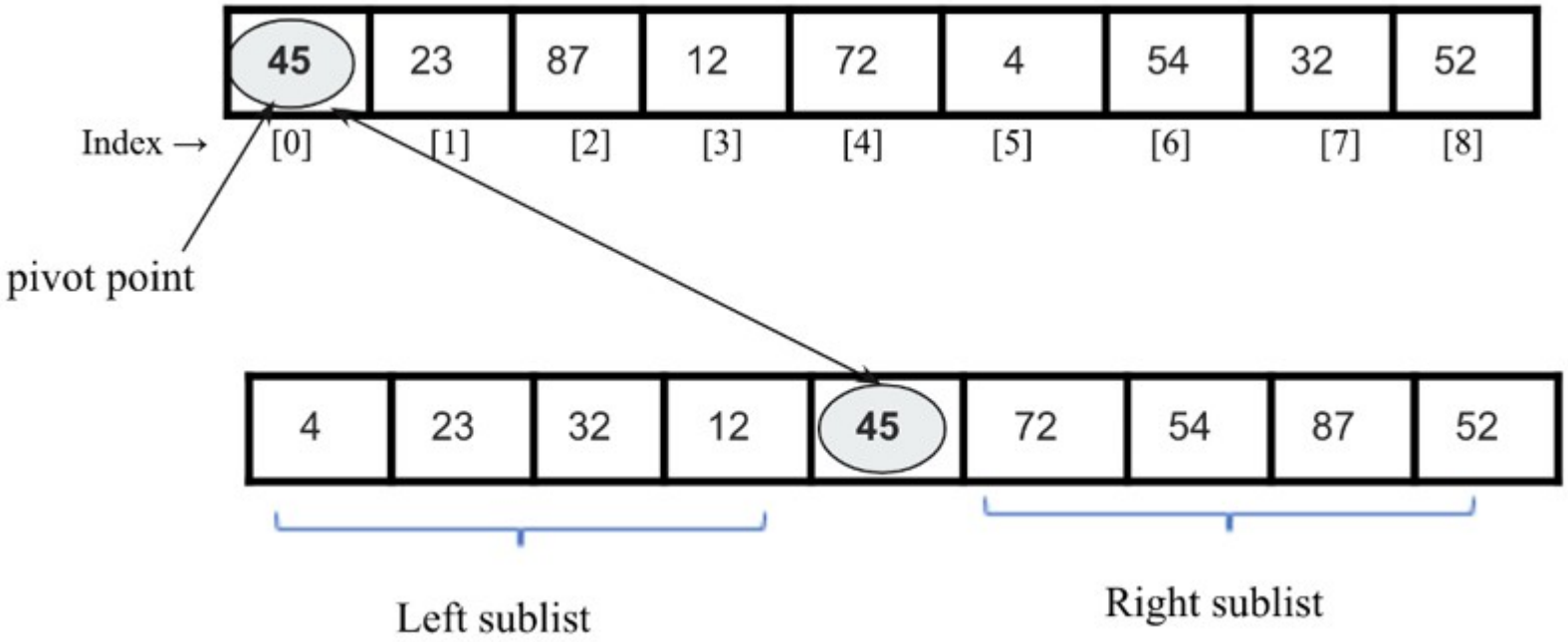
Fatec
Rio Claro

Neste algoritmo, primeiramente, particionamos a lista fornecida de elementos de dados não classificados em duas sublistas de forma que todos os elementos do lado esquerdo desse ponto de partição (também chamado de pivô) sejam menores que o pivô, e todos os elementos do lado direito do pivô sejam maiores.

Isso significa que os elementos da sublista esquerda e da sublista direita não serão classificados, mas o elemento pivô estará em sua posição correta na lista completa.

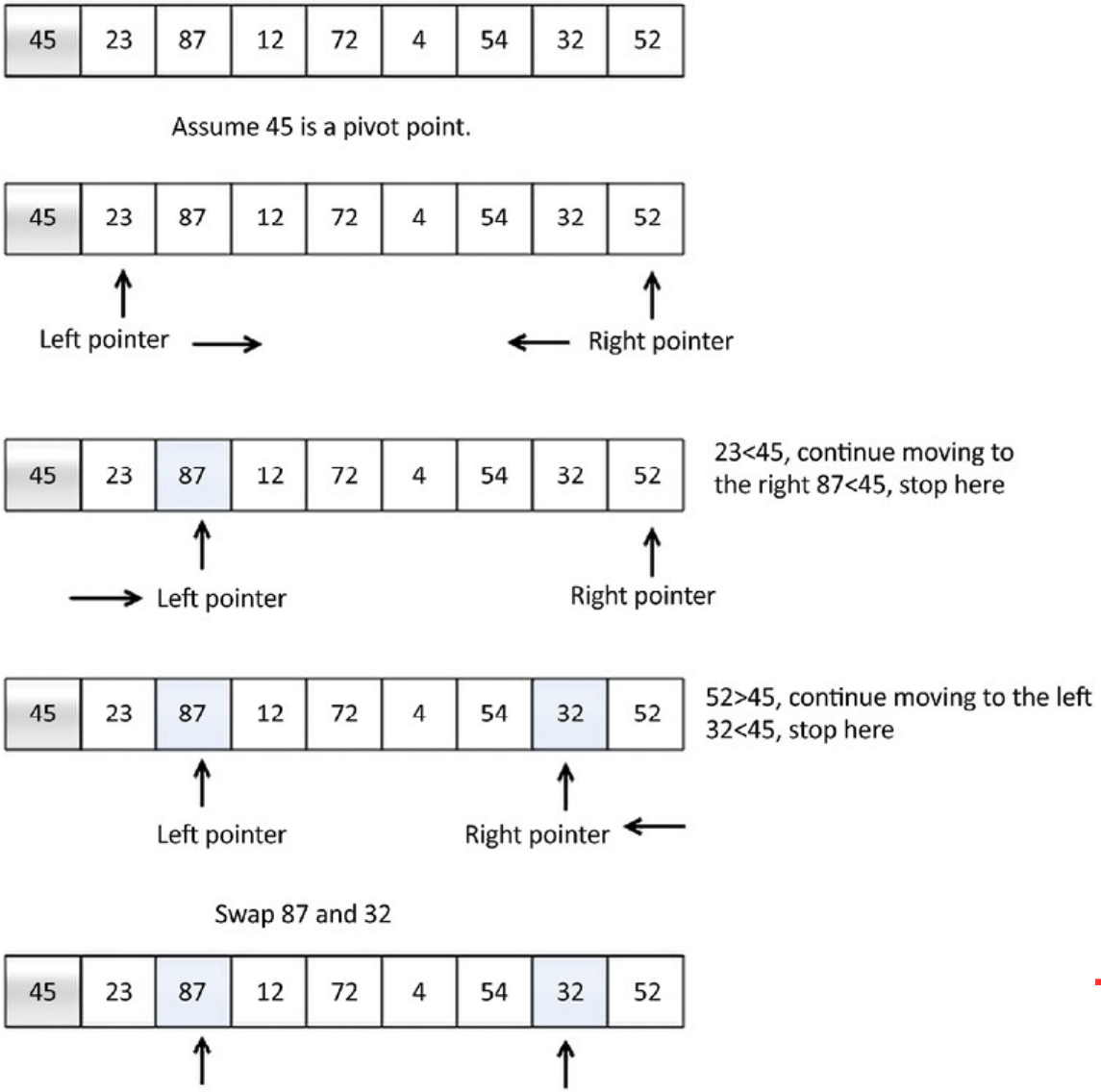
QuickSort

Fatec
Rio Claro



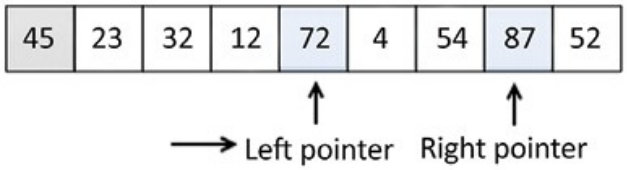
QuickSort

Fatec
Rio Claro

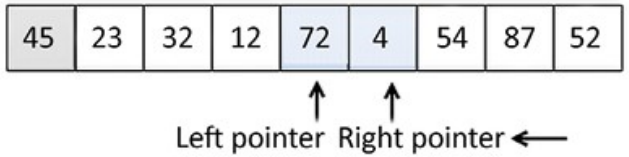


QuickSort

Fatec
Rio Claro

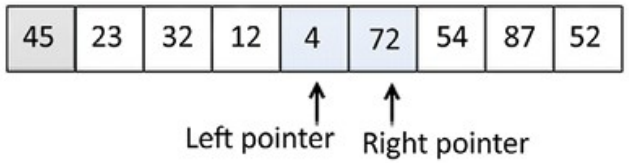


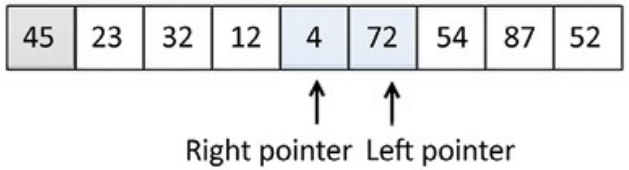
12<45, continue moving to the right 72>45, stop here

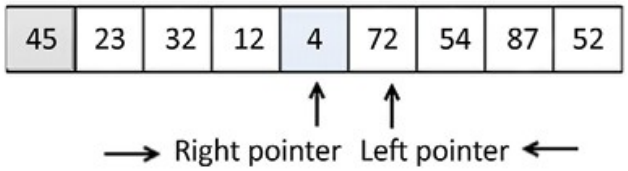


54>45, continue moving to the left 4<45, stop here

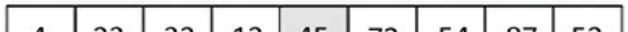
Swap 72 and 4







72>45, stop here
4<45, stop here
Swap 45 and 4.



QuickSort

Fatec
Rio Claro

No algoritmo quicksort, o algoritmo de partição leva tempo O(n). Como o algoritmo quicksort segue o paradigma de dividir para conquistar, ele leva tempo O(logn); portanto, a complexidade média geral do algoritmo quicksort é O(n) * O(logn) = O(nlogn). O algoritmo quicksort fornece uma complexidade de tempo de execução de pior caso de $O(n^2)$. A complexidade de pior caso para o algoritmo quicksort seria quando ele seleciona o pior ponto de pivô todas as vezes, e uma das partições sempre tem um único elemento.

QuickSort

Fatec
Rio Claro

Por exemplo, se a lista já estiver ordenada, a complexidade de pior caso ocorreria se a partição escolhesse o menor elemento como um ponto de pivô. Quando a complexidade de pior caso ocorre, o algoritmo quicksort pode ser aprimorado usando o quicksort randomizado.

O algoritmo quicksort é eficiente quando a lista de elementos fornecida é muito longa; ele funciona melhor em comparação com os outros algoritmos mencionados para ordenação em tais situações.

TimSort

TimSort

Fatec
Rio Claro

O Timsort é usado como o algoritmo de ordenação padrão em todas as versões do Python a partir da versão 2.3.

O algoritmo Timsort é um algoritmo ideal para listas longas do mundo real, baseado em uma combinação dos algoritmos de ordenação por mesclagem e ordenação por inserção.

TimSort

Fatec
Rio Claro

O algoritmo Timsort utiliza o melhor dos dois algoritmos; a ordenação por inserção funciona melhor quando o array é parcialmente ordenado e seu tamanho é pequeno, e o método de mesclagem do algoritmo de ordenação por mesclagem funciona rapidamente quando precisamos combinar listas pequenas ordenadas.

TimSort

Fatec
Rio Claro

O conceito principal do algoritmo Timsort é que ele usa o algoritmo de ordenação por inserção para ordenar pequenos blocos (também conhecidos como chunks) de elementos de dados e, em seguida, usa o algoritmo de ordenação por mesclagem para mesclar todos os chunks ordenados.

A principal característica do algoritmo Timsort é que ele aproveita elementos de dados já ordenados, conhecidos como "execuções naturais", que ocorrem com muita frequência em dados do mundo real.

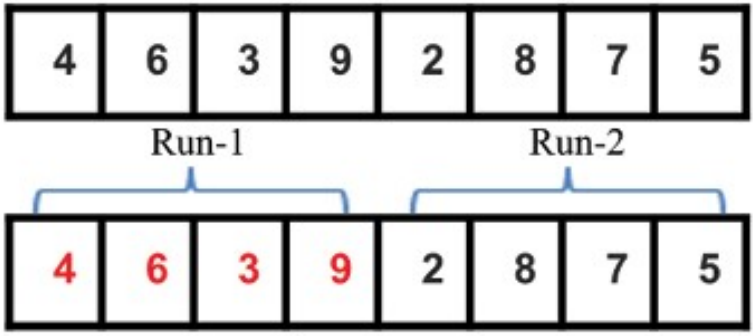
TimSort

Fatec
Rio Claro

Vamos dar um exemplo para entender o funcionamento do algoritmo Timsort. Digamos que temos o array [4, 6, 3, 9, 2, 8, 7, 5]. Ordenamos usando o algoritmo Timsort; aqui, para simplificar, tomamos o tamanho da execução como 4. Então, dividimos o array fornecido em duas execuções, *run1* e *run2*.

TimSort

Fatec
Rio Claro



We apply insertion sort to sort this run 1.

Run 1 is sorted, and we apply insertion sort on run 2.

Both run 1 and run 2 are sorted. We apply merge method to sort the complete list.

After merging run 1 and run 2 we get the sorted array.

TimSort

Fatec
Rio Claro

Em seguida, ordenamos a execução 1 usando o algoritmo de ordenação por inserção e, em seguida, ordenamos a execução 2 usando o algoritmo de ordenação por inserção. Uma vez que todas as execuções tenham sido ordenadas, usamos o método de mesclagem do algoritmo de ordenação por mesclagem para obter a lista ordenada final completa.

TimSort

Fatec
Rio Claro

O Timsort é a melhor escolha para ordenação, mesmo que o comprimento da lista fornecida seja curto. Nesse caso, ele usa o algoritmo de ordenação por inserção, que é muito rápido para listas menores, e o algoritmo Timsort funciona rapidamente para listas longas devido ao método de mesclagem; portanto, o algoritmo Timsort é uma boa escolha para ordenação devido à sua adaptabilidade para ordenar matrizes de qualquer comprimento em uso no mundo real.

Síntese

Síntese

Fatec
Rio Claro

Exploramos algoritmos de ordenação importantes e populares que são muito úteis para muitas aplicações do mundo real.

Discutimos os algoritmos merge sort, bubble sort, insertion sort, selection sort, quicksort e Timsort, além de explicar sua implementação em Python.

Em geral, o algoritmo quicksort tem melhor desempenho do que os outros algoritmos de ordenação, e o algoritmo Timsort é a melhor escolha para uso em aplicações do mundo real.

Síntese

Fatec
Rio Claro

Algorithm	worst-case	average-case	best-case
Bubble sort	`O(n2)`	`O(n2)`	`O(n)`
Insertion sort	`O(n2)`	`O(n2)`	`O(n)`
Selection sort	`O(n2)`	`O(n2)`	`O(n2)`
Quicksort	`O(n2)`	`O(n log n)`	`O(n log n)`
Timsort	`O(n log n)`	`O(n log n)`	`O(n)`

Fatec
Rio Claro

Dúvidas

Prof. Orlando Saraiva Júnior
orlando.nascimento@fatec.sp.gov.br

Prática

Prática

Fatec
Rio Claro

Visite o site https://pythonfluente.com/2/#sort_x_sorted

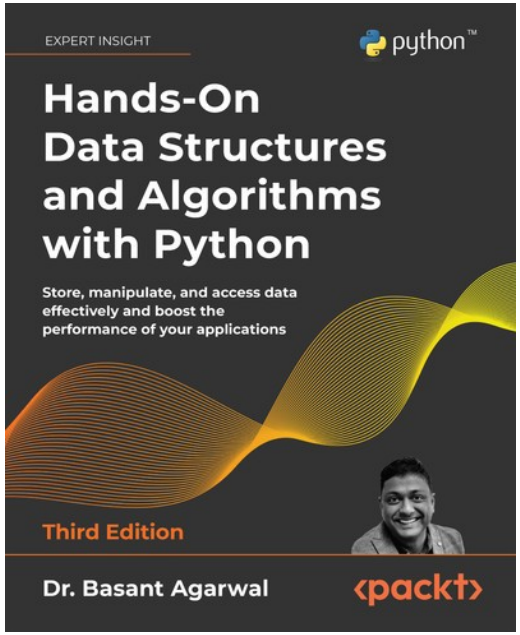
Neste site, Luciano Ramalho explica list.sort e a função embutida sorted

Visite o site https://pt.wikipedia.org/wiki/Timsort

Uma explicação sobre o algori

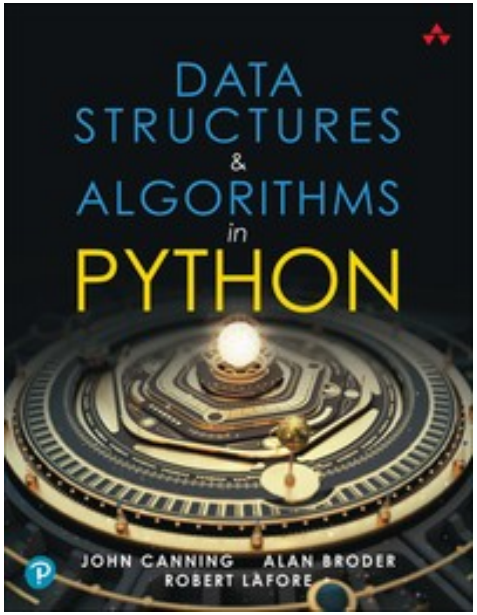
Bibliografia

AGARWAL, Basant. Hands-On Data Structures and Algorithms with Python. 3. ed. Birmingham: Packt Publishing, 2022.



Bibliografia

CANNING, John; BRODER, Alan; LAFORE, Robert. Data structures & algorithms in Python. Boston: Addison-Wesley Professional, 2019.



Bibliografia

RAMALHO, Luciano. Fluent Python: clear, concise, and effective programming. 2. ed. Sebastopol, CA: O'Reilly Media, Inc., 2022. 1014 p.