

Algoritmos e Estrutura de Dados II

Algoritmos de Ordenação

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia



Roteiro


- 1 Objetivos
- 2 Referências bibliográficas
- 3 Shellsort
- 4 Mergesort
- 5 Heapsort
- 6 Quicksort

Objetivos


Esta aula tem como objetivos:


- 1 Apresentar os conceitos básicos sobre ordenação;
- 2 Explicitar os métodos mais eficientes de ordenação por comparação:
 - Shellsort
 - Mergesort
 - Heapsort
 - Quicksort
- 3 Exemplificar a execução dos algoritmos.

Referências bibliográficas

 KNUTH, D. E. *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*. Third. Reading, Mass.: [s.n.], 1997. ISBN 0201896834 9780201896831.

 OLIVEIRA, S. L. G. *Algoritmos e seus fundamentos*. 1. ed. Lavras: Editora UFLA, 2011.

 SEDGEWICK, R.; FLAJOLET, P. *An Introduction to the Analysis of Algorithms*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1996. ISBN 0-201-40009-X.

 ZIVIANI, N. *Projeto de Algoritmos: com implementações em Pascal e C*. São Paulo: Cengage Learning, 2011. ISBN 9788522110506.

Shellsort

Shellsort

- Proposto por Donald Shell em 1959.
- Trata-se de uma extensão do algoritmo de ordenação por inserção (Insertionsort);

Shellsort

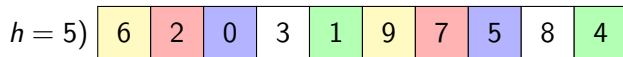
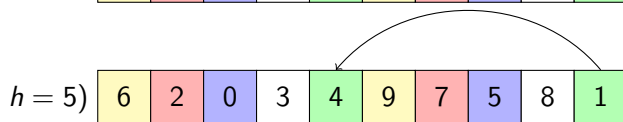
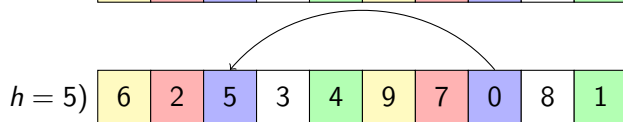
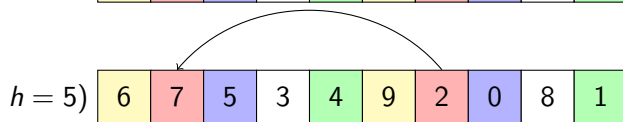
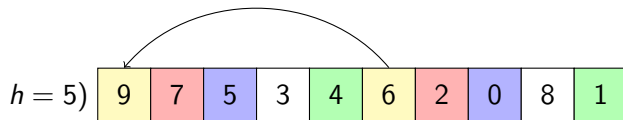
Motivação

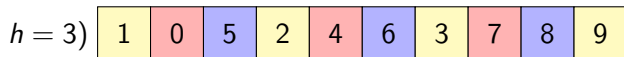
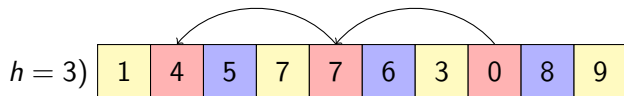
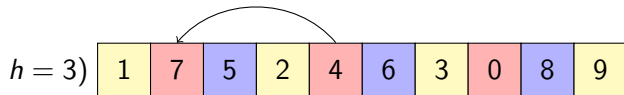
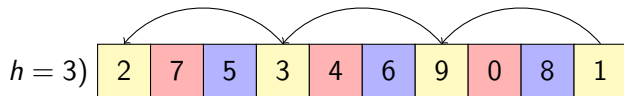
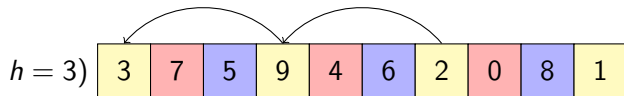
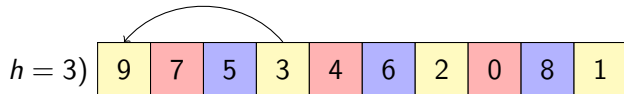
- Na ordenação por inserção troca-se apenas os elementos adjacentes para posicioná-lo na localização correta.
- Dessa forma, são efetuadas $n - 1$ comparações e movimentações no pior caso (quando o menor item está na última posição).
- O **Shellsort** contorna este problema permitindo trocas de registros distantes entre si.

Shellsort

Algoritmo

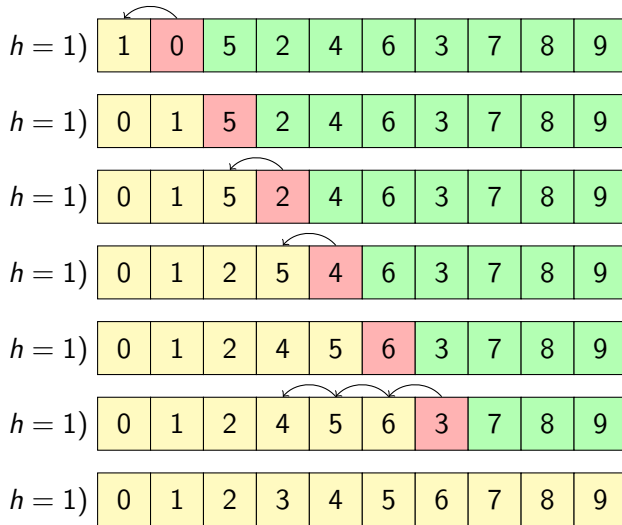
- Os elementos que estão a uma distância h , são ordenados utilizando o algoritmo de ordenação por inserção.
 - O elemento na posição x é comparado (e trocado) com o elemento na posição $x + h$.
 - O vetor resultante é dito estar h -ordenado.
- A cada iteração diminuiu-se a distância h , reaplicando o algoritmo de ordenação por inserção, até atingir $h = 1$.
- Quando $h = 1$, o algoritmo é equivalente ao algoritmo de ordenação por inserção (Insertionsort).





Shellsort

Exemplo



Shellsort

Escolha da distância h

- Qualquer sequência terminando com $h = 1$ garante ordenação correta.
- A escolha da distância h possui um forte impacto no desempenho do algoritmo.
- Exemplo de sequência ruim: 1, 2, 4, 8, 16.
 - Não compara elementos em posições pares com elementos em posições ímpares até a última iteração.

Shellsort

Escolha da distância h

- Sugestão de Knuth (1997) para a escolha de h :

$$h(s) = \begin{cases} 1 & \text{se } s = 1 \\ 3h(s-1) + 1 & \text{se } s > 1 \end{cases}$$

- Essa sequência corresponde aos valores: 1, 4, 13, 40, 121, 364, 1093, 3280, ...
- Knuth (1997)[p. 95] mostrou experimentalmente que esta sequência é difícil de ser batida por mais de 20% em eficiência.
- Outras sequências têm desempenho similar.

Shellsort

Pseudo-código

Algoritmo 1: Shellsort

Entrada: Vetor $V[0..n-1]$, tamanho n

Saída: Vetor V ordenado

```
1 início
2    $h \leftarrow 1$ 
3   enquanto ( $h < n$ ) faça
4      $h \leftarrow 3h + 1$ 
5   enquanto ( $h \geq 1$ ) faça
6      $h \leftarrow \frac{h}{3}$ 
7     para ( $i \leftarrow h$  até  $n - 1$ ) faça
8       chave  $\leftarrow V[i]$ 
9        $j \leftarrow i - h$ 
10      enquanto ( $j \geq 0$  AND  $V[j] > \text{chave}$ ) faça
11         $V[j + h] \leftarrow V[j]$ 
12         $j \leftarrow j - h$ 
13       $V[j + h] \leftarrow \text{chave}$ 
```

Shellsort

Complexidade

- A complexidade desse algoritmo ainda não é conhecida.
- Por enquanto ninguém foi capaz de encontrar uma fórmula fechada para sua função de complexidade.
- A sua análise contém alguns problemas matemáticos bem difíceis, como por exemplo, escolher a sequência de incrementos.
- O que se sabe é que cada incremento não deve ser múltiplo do anterior.

Shellsort

Análise

Conjecturas referentes ao número de comparações para a sequência de Knuth (1997):

- Conjectura 1: $C(n) = O(n^{1,25})$
- Conjectura 2: $C(n) = O(n(\log(n))^2)$

Shellsort

Vantagens x Desvantagens

- Vantagens
 - **Shellsort** é uma ótima opção para arquivos de tamanho moderado.
 - Sua implementação é simples e requer uma quantidade de código pequena.
- Desvantagens
 - O tempo de execução do algoritmo é sensível à ordem inicial dos elementos.
 - Não é estável.

Mergesort

Mergesort

- **Mergesort** é um algoritmo de ordenação recursivo.
- Recursivamente ordena as duas metades do vetor.
- Utiliza a estratégia de **Divisão e Conquista** (D&C).
- É um algoritmo eficiente: possui tempo de execução $O(n \log n)$.

Mergesort

Divisão e Conquista

Método de Divisão e Conquista

- **Divisão:** Divida o problema em duas ou mais partes, criando subproblemas menores.
- **Conquista:** Os subproblemas são resolvidos recursivamente usando D&C. Caso os subproblemas sejam suficientemente pequenos resolva-os de forma direta.
- **Combina:** Tome cada uma das partes e junte-as todas de forma a resolver o problema original.

Mergesort

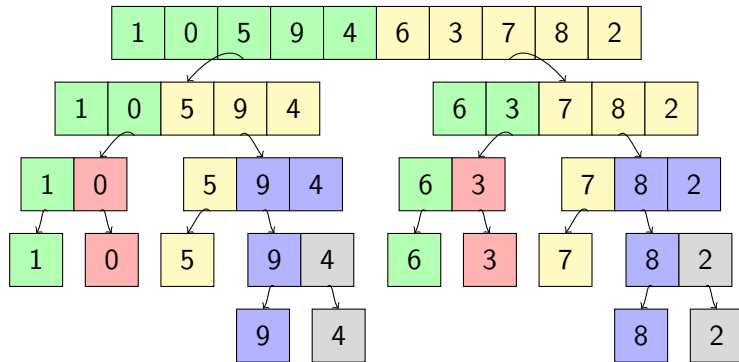
Divisão e Conquista

D&C no Mergesort:

- Caso o tamanho do vetor seja maior que 1.
 - ➊ Divida o vetor no meio.
 - ➋ Ordene a primeira metade recursivamente.
 - ➌ Ordene a segunda metade recursivamente.
 - ➍ Intercale as duas metades.
- Senão devolva o elemento, pois 1 elemento encontra-se ordenado.

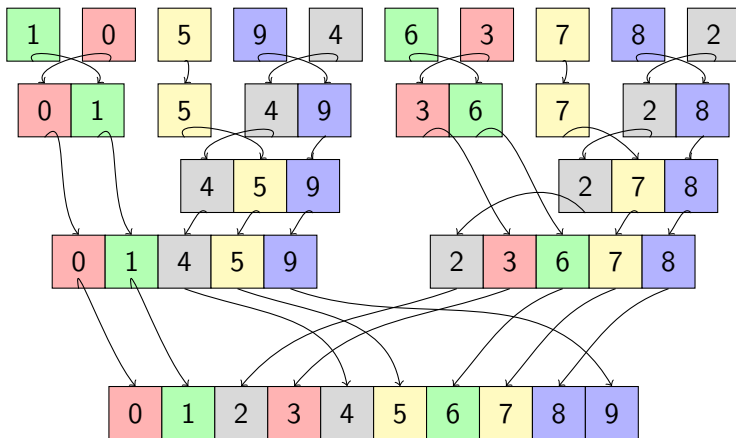
Mergesort

Exemplo



Mergesort

Exemplo



Mergesort

Pseudo-código

Algoritmo 2: MergesortOrdena

Entrada: Vetor $V[0..n-1]$, tamanho do vetor n .

Saída: Vetor V ordenado

```
1 início
2   Mergesort(V,0,n)
```

Algoritmo 3: Mergesort

Entrada: Vetor $V[i..f-1]$, início i de V , e o final f de V .

Saída: Vetor V ordenado

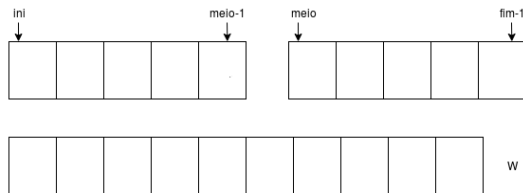
```
1 início
2   se  $(i < f - 1)$  então
3        $m \leftarrow \frac{(i+f)}{2}$ 
4       Mergesort(V,i,m)
5       Mergesort(V,m,f)
6       Merge(V, i, m, f)
```

Mergesort

Intercalação

A intercalação de dois vetores ordenados pode ser feito em tempo linear.

- Considere os dois vetores a serem intercalados:
 - O vetor da esquerda começa em `ini` e termina em `meio-1`.
 - O vetor da direita em `meio` e termina em `fim-1`.
- Ao realizar a intercalação, utiliza-se um vetor auxiliar W , de tamanho $(fim - ini)$, para armazenar os elementos intercalados.

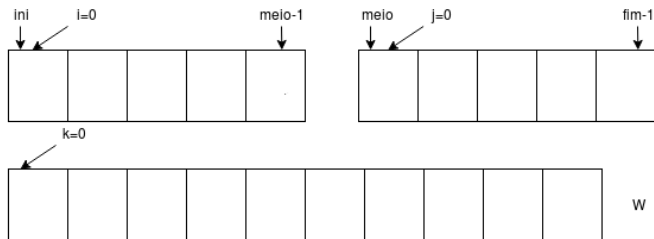


Mergesort

Intercalação

Utilizaremos três variáveis para percorrer os vetores:

- i percorre o vetor da esquerda, indicando o próximo elemento a ser inserido em W .
- j percorre o vetor da direita, indicando o próximo elemento a ser inserido em W .
- k indica a posição em que o elemento deve ser inserido em W .



Mergesort

Intercalação

- A intercalação acaba quando $i = \textit{meio}$ ou $j = \textit{fim}$.
- A cada passo, comparamos o elemento na posição i com o elemento na posição j , selecionando o menor e inserindo em W .
- Em seguida, incrementamos os respectivos índices (i ou j e k).

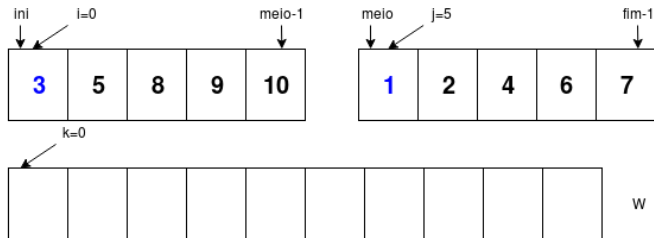
```
1  $i \leftarrow \textit{ini}; j \leftarrow \textit{meio}; k \leftarrow 0$ 
2 enquanto ( $i < \textit{meio}$  AND  $j < \textit{fim}$ ) faça
3     se  $V[i] \leq V[j]$  então
4          $W[k] \leftarrow V[i]$ 
5          $i \leftarrow i + 1$ 
6     senão
7          $W[k] \leftarrow V[j]$ 
8          $j \leftarrow j + 1$ 
9      $k \leftarrow k + 1$ 
```

Mergesort

Intercalação

Considere como exemplo a intercalação dos dois vetores.

- 1 Compara-se 3 com 1.

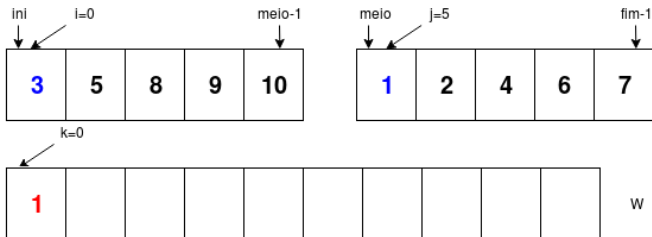


Mergesort

Intercalação

Considere como exemplo a intercalação dos dois vetores.

- 1 Compara-se 3 com 1.
- 2 Insere o menor (1) em W .

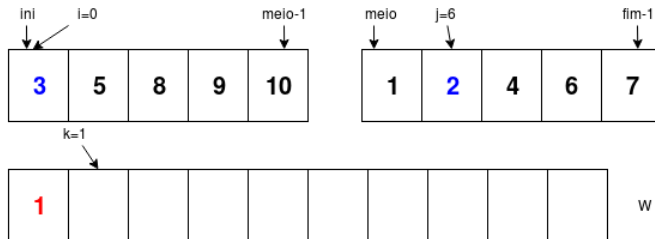


Mergesort

Intercalação

Considere como exemplo a intercalação dos dois vetores.

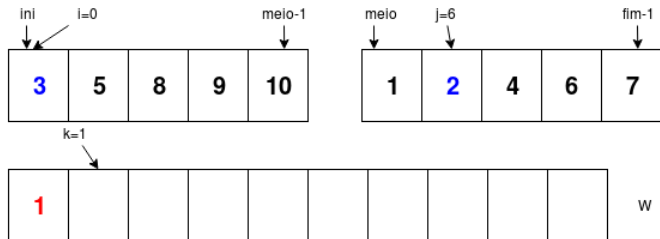
- 1 Compara-se 3 com 1.
- 2 Insere o menor (1) em W .
- 3 Por fim, incrementa-se j e k :
 - $j \leftarrow 5 + 1$
 - $k \leftarrow 0 + 1$



Mergesort

Intercalação

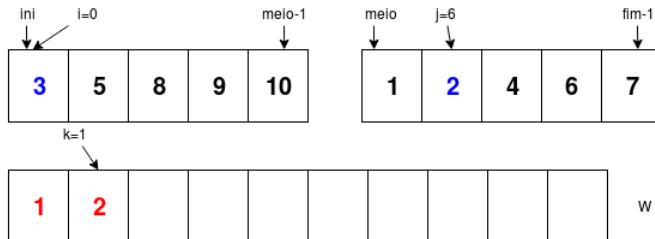
- 1 Compara-se 3 com 2.



Mergesort

Intercalação

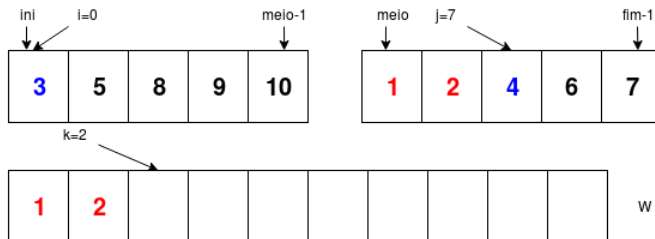
- 1 Compara-se 3 com 2.
- 2 Insere o menor (2) em W .



Mergesort

Intercalação

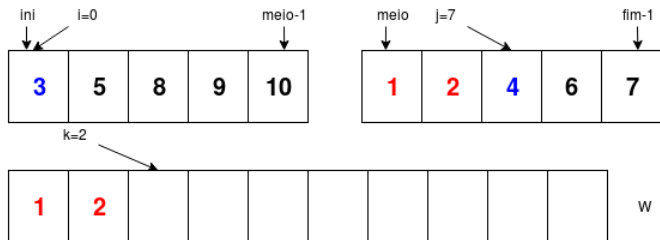
- 1 Compara-se 3 com 2.
- 2 Insere o menor (2) em W .
- 3 Por fim, incrementa-se j e k .
 - $j \leftarrow 6 + 1$
 - $k \leftarrow 1 + 1$



Mergesort

Intercalação

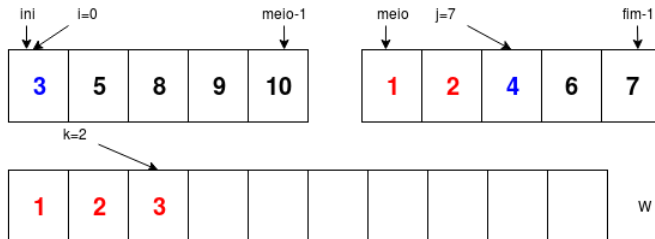
- 1 Compara-se 3 com 4.



Mergesort

Intercalação

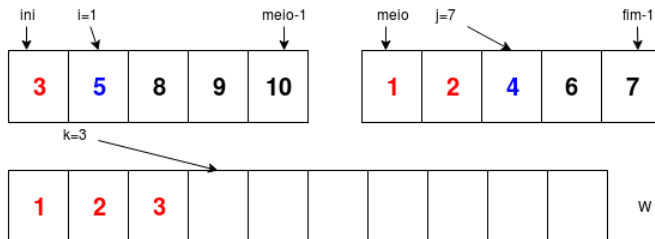
- 1 Compara-se 3 com 4.
- 2 Insere o menor (3) em W .



Mergesort

Intercalação

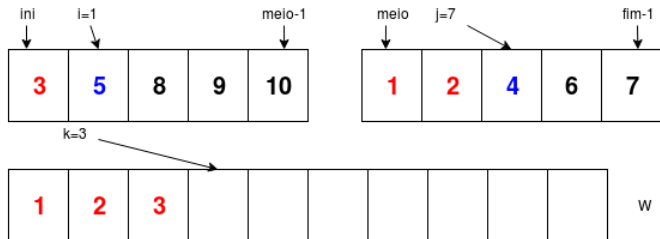
- 1 Compara-se 3 com 4.
- 2 Insere o menor (3) em W .
- 3 Em seguida, incrementa-se i e k .
 - $i \leftarrow 0 + 1$
 - $k \leftarrow 2 + 1$



Mergesort

Intercalação

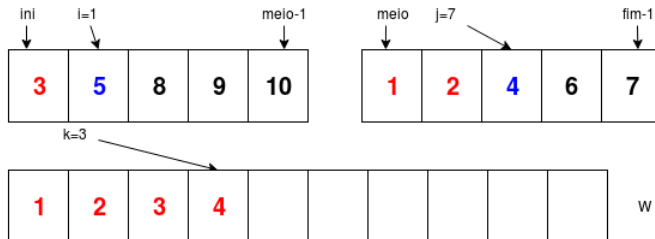
- 1 Compara-se 5 com 4.



Mergesort

Intercalação

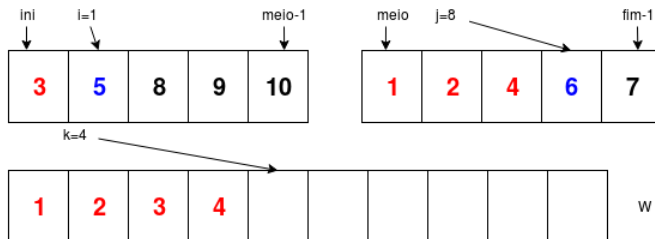
- 1 Compara-se 5 com 4.
- 2 Insere o menor (4) em W .



Mergesort

Intercalação

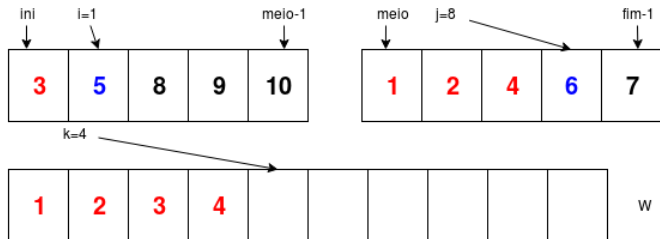
- 1 Compara-se 5 com 4.
- 2 Insere o menor (4) em W .
- 3 Em seguida, incrementa-se j e k .
 - $j \leftarrow 7 + 1$
 - $k \leftarrow 3 + 1$



Mergesort

Intercalação

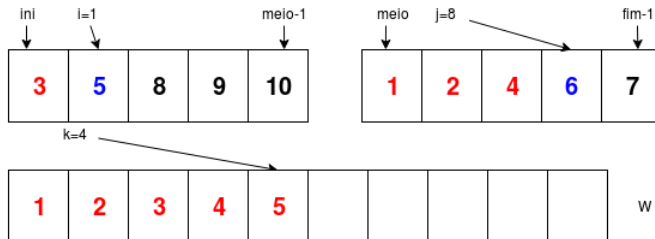
- 1 Compara-se 5 com 6.



Mergesort

Intercalação

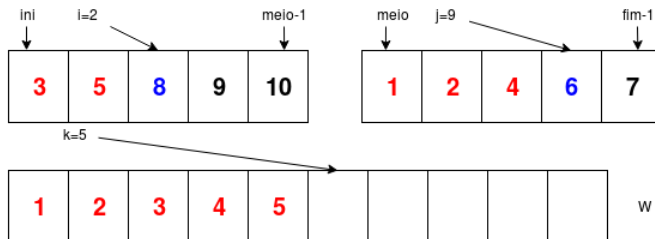
- 1 Compara-se 5 com 6.
- 2 Insere o menor (5) em W .



Mergesort

Intercalação

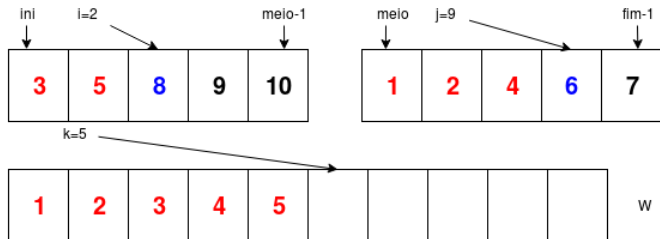
- 1 Compara-se 5 com 6.
- 2 Insere o menor (5) em W .
- 3 Em seguida, incrementa-se i e k .
 - $i \leftarrow 1 + 1$
 - $k \leftarrow 4 + 1$



Mergesort

Intercalação

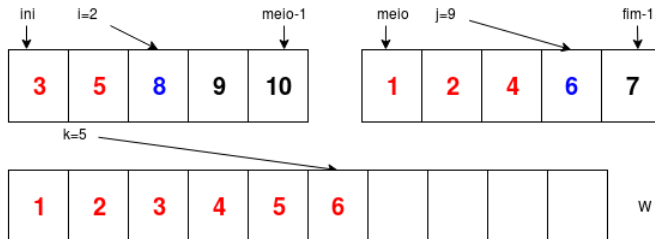
- 1 Compara-se 8 com 6.



Mergesort

Intercalação

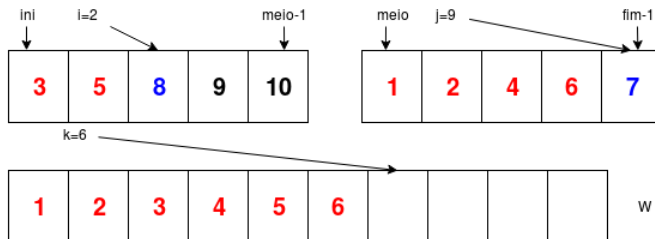
- 1 Compara-se 8 com 6.
- 2 Insere o menor (6) em W .



Mergesort

Intercalação

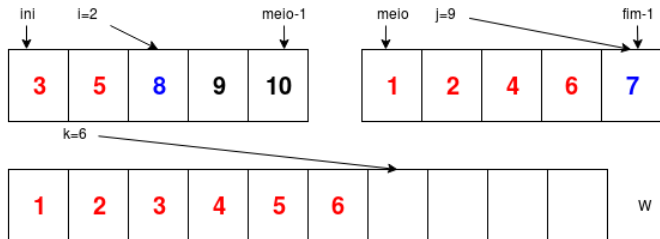
- 1 Compara-se 8 com 6.
- 2 Insere o menor (6) em W .
- 3 Em seguida, incrementa-se j e k .
 - $j \leftarrow 8 + 1$
 - $k \leftarrow 5 + 1$



Mergesort

Intercalação

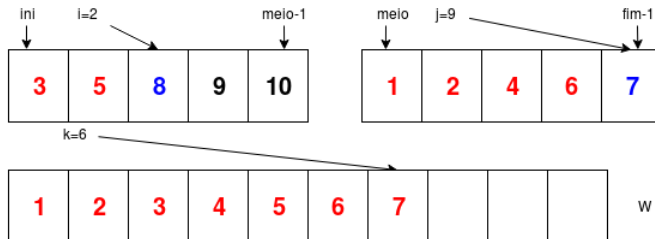
- 1 Compara-se 8 com 7.



Mergesort

Intercalação

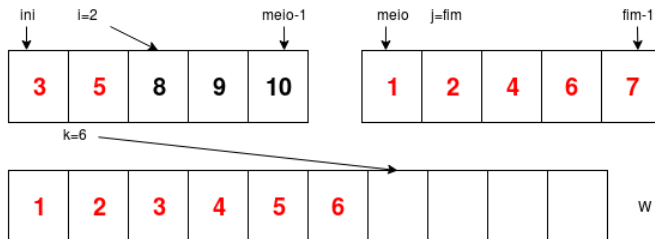
- 1 Compara-se 8 com 7.
- 2 Insere o menor (7) em W .



Mergesort

Intercalação

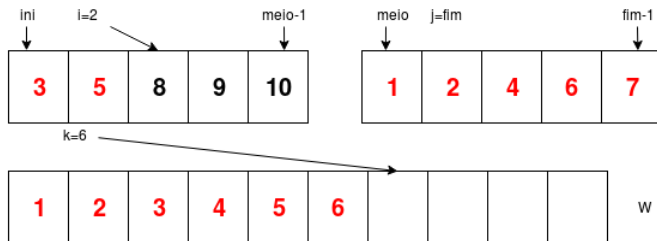
- 1 Compara-se 8 com 7.
- 2 Insere o menor (7) em W .
- 3 Em seguida, incrementa-se j e k .
 - $j \leftarrow 9 + 1$
 - $k \leftarrow 6 + 1$



Mergesort

Intercalação

- 1 Todos os elementos do vetor da direita foram inseridos em W .
- 2 A condição “**Enquanto** ($i < meio$ AND $j < fim$)” não é mais verdadeira.
 - pois $j = fim$, ou seja, atingiu o fim do vetor.
- 3 Mas ainda falta inserir os elementos que restam no vetor da esquerda.



Mergesort

Intercalação

- Quando um dos dois vetores não tiver mais elementos, deve-se copiar os elementos restantes para o vetor W .
- O código a seguir realiza essa operação para ambos os vetores (da esquerda e da direita).

```
1 enquanto ( $i < meio$ ) faça
```

```
2    $W[k] \leftarrow V[i]$ 
```

```
3    $i \leftarrow i + 1$ 
```

```
4    $k \leftarrow k + 1$ 
```

```
5 enquanto ( $j < fim$ ) faça
```

```
6    $W[k] \leftarrow V[j]$ 
```

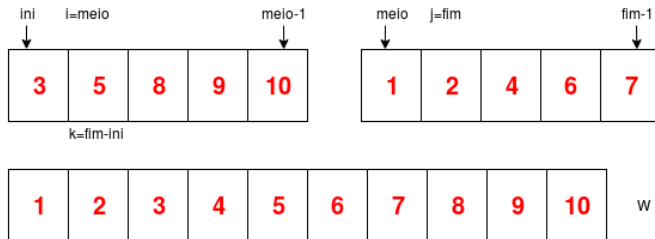
```
7    $j \leftarrow j + 1$ 
```

```
8    $k \leftarrow k + 1$ 
```

Mergesort

Intercalação

- Ao copiar todos os elementos restantes para W , o vetor W estará ordenado.



- Mas ainda resta copiar os elementos de W para V , pois o algoritmo ordena o vetor V .

Mergesort

Intercalação

- Por fim, copie os elementos de W para V .

```
1 para ( $i \leftarrow ini$  até  $fim - 1$ ) faça  
2    $V[i] \leftarrow W[i - ini]$ 
```

- O algoritmo completo fica conforme o pseudo-código a seguir:

Merge

Pseudo-código

Algoritmo 4: Merge

Entrada: Vetor $V[ini..fim - 1]$, ini , $meio$, fim .

Saída: Vetor V ordenado

```
1 início
2   // Considere o vetor auxiliar  $W[ini..fim-1]$ 
3    $i \leftarrow ini$ ;  $j \leftarrow meio$ ;  $k \leftarrow 0$ 
4   enquanto ( $i < meio$  e  $j < fim$ ) faça
5       se  $V[i] \leq V[j]$  então
6            $W[k] \leftarrow V[i]$ 
7            $i \leftarrow i + 1$ 
8       senão
9            $W[k] \leftarrow V[j]$ 
10           $j \leftarrow j + 1$ ;
11       $k \leftarrow k + 1$ 
12  enquanto ( $i < meio$ ) faça
13       $W[k] \leftarrow V[i]$ 
14       $i \leftarrow i + 1$ ;  $k \leftarrow k + 1$ 
15  enquanto ( $j < fim$ ) faça
16       $W[k] \leftarrow V[j]$ 
17       $j \leftarrow j + 1$ ;  $k \leftarrow k + 1$ 
18  para ( $i \leftarrow ini$  até  $fim - 1$ ) faça
19       $V[i] \leftarrow W[i - ini]$ 
```

Mergesort

Análise

- Quando um algoritmo contém chamadas recursivas, o cálculo de seu tempo de execução pode usar recorrências.
- Para o método de Divisão e Conquista.
- Seja $T(n)$ o tempo do algoritmo. Suponha que dividimos em a subproblemas de tamanho $\frac{n}{b}$ cada e seja $D(n)$ o tempo para dividir os subproblemas e $C(n)$ o tempo para combiná-los. Então

$$T(n) = \begin{cases} \Theta(1) & \text{se } n \leq c \\ aT(\frac{n}{b}) + D(n) + C(n) & \text{se } n > c \end{cases}$$

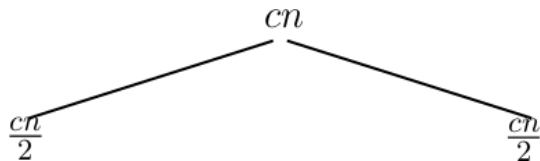
Mergesort

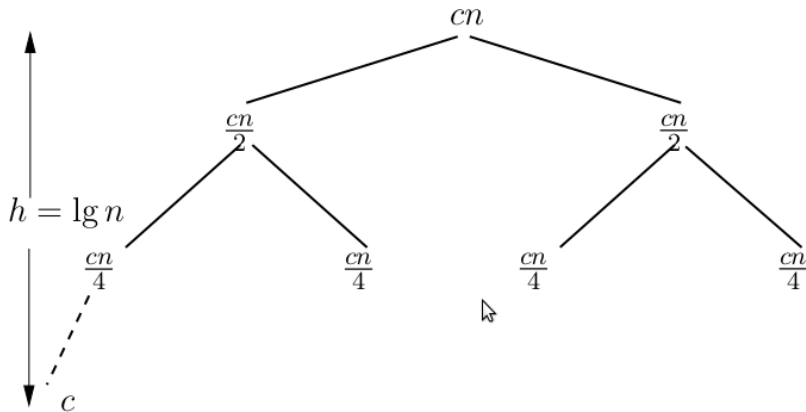
Análise

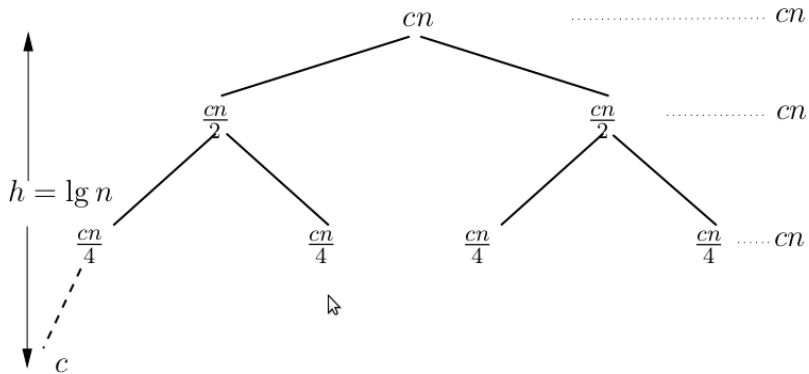
- Dividir: Tempo Constante $\Theta(1)$.
- Conquistar: Dois problemas de $\frac{n}{2}$ cada: $2T(\frac{n}{2})$.
- Combinar: Tempo do Merge: $\Theta(n)$.

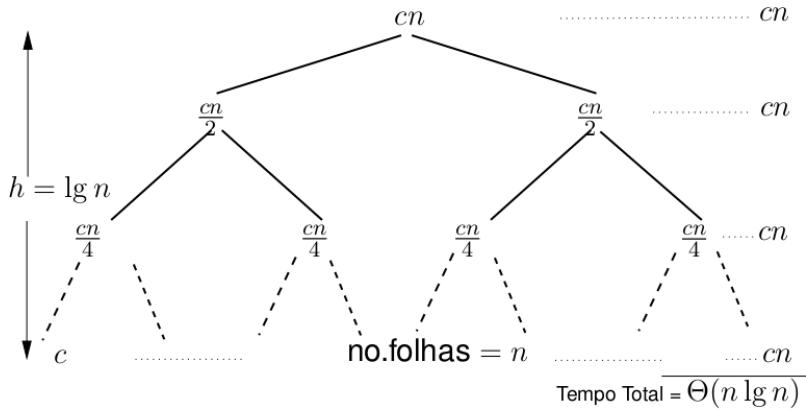
$$T(n) = \begin{cases} \Theta(1) & \text{se } n = 1 \\ 2T(\frac{n}{2}) + \Theta(n) & \text{se } n > 1 \end{cases}$$

- Como resolver? Árvore de Recursão!









Heapsort

Heapsort

Características

- Possui o mesmo princípio de funcionamento da ordenação por seleção (Selectionsort).
 - Selecione o menor item do vetor.
 - Troque-o pelo item da primeira posição.
 - Repita operação com os elementos restantes do vetor.
- Implementação direta.
 - Encontrar o menor elemento requer $n - 1$ comparações.
- Ideia:
 - Utilização de uma *fila de prioridades* implementada com um **Heap**.

Heapsort

Fila de Prioridades

- Uma **Fila de Prioridades** é um tipo abstrato de dados que permite executar as seguintes operações :
 - inserir um novo item;
 - remover o item com a maior chave (maior prioridade).
- A *chave* de cada item reflete a *prioridade* em que se deve tratar aquele item.
- Aplicações:
 - Sistemas operacionais, paginação de memória, ordenação, simulação de eventos.

Heapsort

Fila de Prioridades

Operações em uma **Fila de Prioridades**:

- Constrói uma *Fila de Prioridade* em um vetor de n itens.
- Insere um novo item.
- Retira o maior item.
- Altera a prioridade de um item.

Heapsort

Fila de Prioridades

Uma *fila de prioridades* pode ser representada utilizando:

- Lista encadeada ordenada
- Lista encadeada não ordenada
- **Heap**

| | Constrói | Insere | Retira máximo | Altera prioridade |
|---------------------------|---------------|-------------|---------------|-------------------|
| Lista ordenada | $O(N \log N)$ | $O(N)$ | $O(1)$ | $O(N)$ |
| Lista não ordenada | $O(N)$ | $O(1)$ | $O(N)$ | $O(1)$ |
| Heaps | $O(N)$ | $O(\log N)$ | $O(\log N)$ | $O(\log N)$ |

Heapsort

Fila de Prioridades

Existem dois tipos de **Heap**:

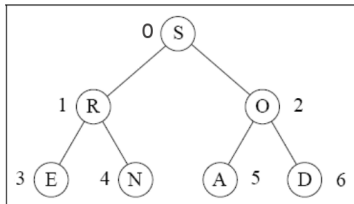
- **Heap Máximo** - pai maior que os filhos.
- **Heap Mínimo** - pai menor que os filhos.
- Entre os filhos não existe ordenação.

Heapsort

Fila de Prioridades

Como representar um **Heap**:

- Representação por Árvore Binária:
 - Pai maior (ou menor) que os filhos.
- Representação vetorial $V[0..n-1]$.
 - **Heap Máximo** - pai na posição i , filhos nas posições $2i+1$ e $2i+2$.
 - $V[i] \geq V[2i+1]$ e $V[i] \geq V[2i+2]$ para todo i .
 - **Heap Mínimo** - pai na posição i , filhos nas posições $2i+1$ e $2i+2$.
 - $V[i] \leq V[2i+1]$ e $V[i] \leq V[2i+2]$ para todo i .



| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| S | R | O | E | N | A | D |

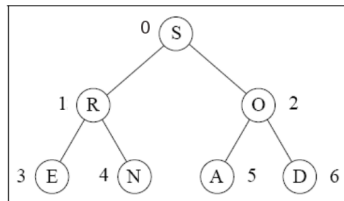
Heapsort

Fila de Prioridades

Representações:

- Correspondência entre representação em árvore e representação em vetor.
- Nós são numerados de 0 a n .
- O primeiro é chamado raiz.
- O nó $\frac{k}{2}$ é o pai do nó k , $1 < k < n$.
- Os nós $(2k + 1)$ e $(2k + 2)$ são filhos da esquerda e direita do nó k , para $0 \leq k < \frac{n}{2}$.

| | | | | | | |
|-------|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| <hr/> | | | | | | |
| S | R | O | E | N | A | D |



Heapsort

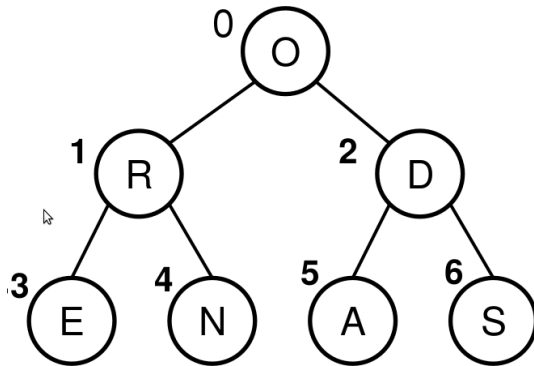
Fila de Prioridades

- Representação por meio de vetores é compacta.
- Permite caminhar pelos nós da árvore facilmente.
- Filhos de um nó i estão nas posições $2i + 1$ e $2i + 2$.
- O pai de um nó i está na posição $\frac{i-1}{2}$.
- A maior chave sempre está na posição 1.

Heapsort

Manutenção

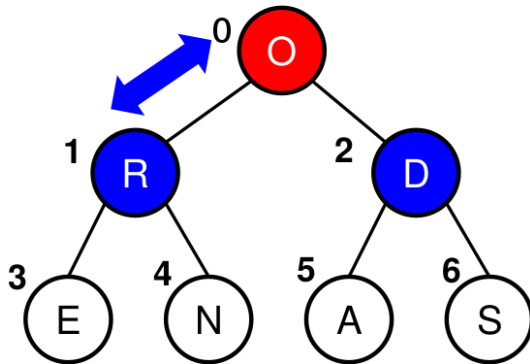
- Precisamos garantir que o valor da chave do pai é maior que dos filhos.
- Se tiver filho maior do que o pai, troca o maior filho com o pai.



Heapsort

Manutenção

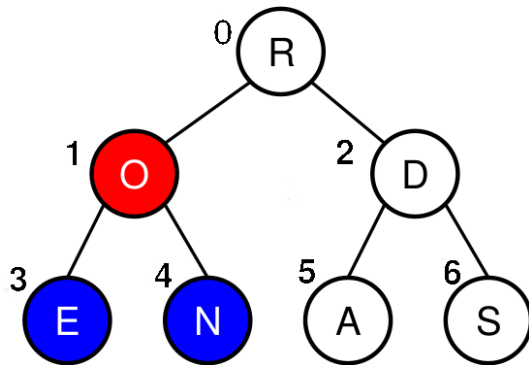
- Precisamos garantir que o valor da chave do pai é maior que dos filhos.
- Se tiver filho maior do que o pai, troca o maior filho com o pai.



Heapsort

Manutenção

- Precisamos garantir que o valor da chave do pai é maior que dos filhos.
- Se tiver filho maior do que o pai, troca o maior filho com o pai.



Heapsort

Manutenção

- Precisamos garantir que o valor da chave do pai é maior que dos filhos.
- Testa se os elementos $V[2i + 1]$ e $V[2i + 2]$ são menores ou igual a $V[i]$.
- Troca com o maior filho caso contrário

| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| O | R | D | E | N | A | |
| R | O | D | E | N | A | |

ConstroiHeap

Pseudo-código

Algoritmo 5: ConstroiHeap

Entrada: Vetor $V[i..n-1]$, raiz no nó i , tamanho do vetor n

Saída: Heap no vetor $V[i..n-1]$

```
1 início
2    $maior \leftarrow i$  // Inicializa maior como a raiz
3    $l \leftarrow 2i + 1$  // Filho da esquerda
4    $r \leftarrow 2i + 2$  // Filho da direita
5   // Se o filho da esquerda é maior que a raiz
6   se  $(l < n \text{ AND } V[l] > V[maior])$  então
7      $maior \leftarrow l$ 
8   // Se filho da direita é maior que a raiz
9   se  $(r < n \text{ AND } V[r] > V[maior])$  então
10     $maior \leftarrow r$ 
11   // Se maior não é a raiz
12   se  $(maior \neq i)$  então
13     Troca  $V[i] \leftrightarrow V[maior]$  // O maior passa a ser a raiz
14     ConstroiHeap( $V, maior, n$ ) // Cria o heap na sub-árvore
```

Heapsort

Pseudo-código

Algoritmo 6: Heapsort

Entrada: Vetor $V[0..n - 1]$, tamanho do vetor n

Saída: Vetor V ordenado

```
1 início
2   // Contrói o heap rearranjando o vetor
3   para (  $i \leftarrow \frac{n}{2} - 1$  decrescendo até  $i = 0$  ) faça
4     ConstroiHeap( $V, i, n$ )
5   // Extrai cada elemento, um por um, do heap
6   para (  $i \leftarrow n - 1$  decrescendo até  $i = 0$  ) faça
7     // Move a raiz atual para o fim do vetor.
8     Troca  $V[0] \leftrightarrow V[i]$ 
9     // Chama a função para recriar o heap
10    // no vetor reduzido
11    ConstroiHeap( $V, 0, i$ )
```

Heapsort

Análise

- A função `ConstroiHeap` é chamado recursivamente no máximo $\log n$ vezes.
- Na função `Heapsort`, a função `ConstroiHeap` é chamada dentro dos laços de repetição das linhas 3 e 6.
 - O laço da linha 3 é executado $\frac{n}{2}$ vezes.
 - O laço da linha 6 é executado n vezes.
- Logo, o Heapsort gasta um tempo proporcional a $O(n \log n)$, no pior caso.

Heapsort

Vantagens x Desvantagens

- Vantagens
 - Comportamento $O(n \log n)$.
- Desvantagens
 - Não é estável.
 - Não é tão rápido quanto o Quicksort.
 - A função **ConstroiHeap** realiza mais operações que a função **Particiona** do Quicksort.

Quicksort

Quicksort

Características

- Proposto por Hoare em 1960 e publicado em 1962.
- É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.
- Provavelmente é o mais utilizado.
- A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores.
- Os problemas menores são ordenados independentemente.
- Os resultados são combinados para produzir a solução final.

Método de Divisão e Conquista

- A parte mais delicada do método é o processo de partição.
- O vetor $V[Esq..Dir]$ é rearranjado por meio da escolha arbitrária de um pivô x .
- O vetor V é particionado em duas partes:
 - Parte esquerda: chaves $\leq x$.
 - Parte direita: chaves $\geq x$.

Quicksort

Ideia Geral

- Algoritmo para o particionamento:
 - 1 Escolha arbitrariamente um pivô x .
 - 2 Percorra o vetor a partir da **esquerda** para a **direita** até que $V[i] \geq x$.
 - 3 Percorra o vetor a partir da **direita** para a **esquerda** até que $V[j] \leq x$.
 - 4 Troque $V[i]$ com $V[j]$.
 - 5 Continue este processo até os apontadores i e j se cruzarem.

Quicksort

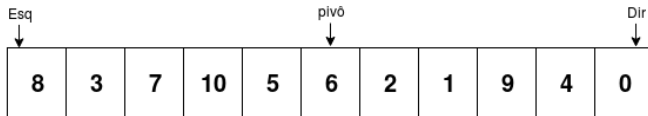
Ideia Geral

- Ao final do algoritmo de partição, o vetor $V[Esq..Dir]$ está particionado de tal forma que:
 - Os itens em $V[Esq], V[Esq + 1], \dots, V[j]$ são menores ou iguais a x .
 - Os itens em $V[i], V[i + 1], \dots, V[Dir]$ são maiores ou iguais a x .

Quicksort

Exemplo

- Como exemplo, considere o vetor $V[0..10]$, de tamanho $n = 11$, a seguir.

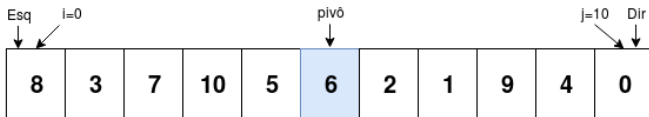


- As variáveis **Esq** e **Dir** delimitam o vetor.
- Vamos utilizar o **Pivô** como o elemento do meio.

Quicksort

Exemplo

- Inicializa-se as variáveis auxiliares i e j .
 - $i \leftarrow \text{Esq.}$
 - Irá percorrer o vetor da **Esq** para a **Dir**.
 - Ou seja, será incrementado: $i \leftarrow i + 1$.
 - $j \leftarrow \text{Dir.}$
 - Irá percorrer o vetor da **Dir** para a **Esq**.
 - Ou seja, será decrementado: $j \leftarrow j - 1$.
- Calcula-se o **Pivô** $x \leftarrow V[\frac{(\text{Esq} + \text{Dir})}{2}]$



Quicksort

Exemplo

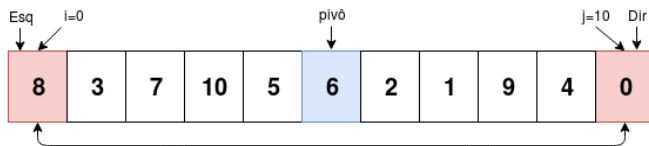
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **verdadeiro**, pára.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.



Quicksort

Exemplo

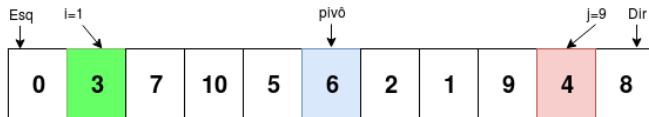
- Troca o elemento na posição i com o elemento na posição j .
 - $V[0] \leftrightarrow V[10]$.
- Em seguida, incrementa i e decrementa j .
 - $i \leftarrow 0 + 1$
 - $j \leftarrow 10 - 1$



Quicksort

Exemplo

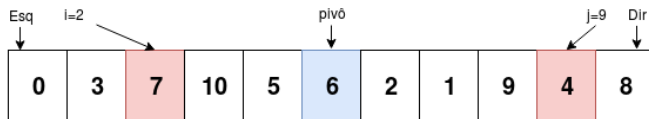
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **falso**, incrementa $i \leftarrow i + 1$.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.



Quicksort

Exemplo

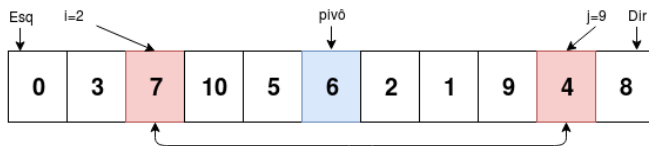
- Verifica se o elemento na posição i é maior que o **Pivô**.
- Verifica se o elemento na posição j é menor que o **Pivô**.
- Se **verdadeiro**, pára.



Quicksort

Exemplo

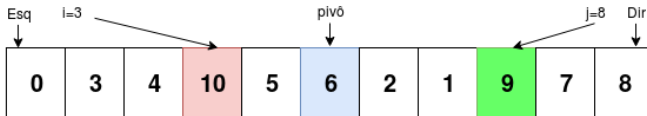
- Troca o elemento na posição i com o elemento na posição j .
 - $V[2] \leftrightarrow V[9]$.
- Em seguida, incrementa i e decrementa j .
 - $i \leftarrow 2 + 1$
 - $j \leftarrow 9 - 1$



Quicksort

Exemplo

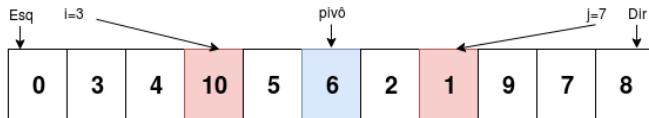
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **verdadeiro**, pára.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **falso**, decrementa $j \leftarrow j - 1$.



Quicksort

Exemplo

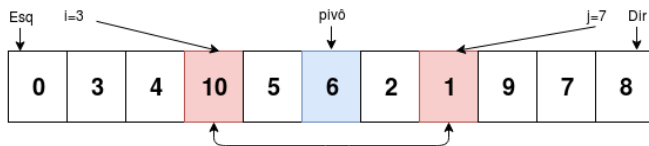
- Verifica se o elemento na posição i é maior que o **Pivô**.
- Verifica se o elemento na posição j é menor que o **Pivô**.
- Se **verdadeiro**, pára.



Quicksort

Exemplo

- Troca o elemento na posição i com o elemento na posição j .
 - $V[3] \leftrightarrow V[7]$.
- Em seguida, incrementa i e decrementa j .
 - $i \leftarrow 3 + 1$
 - $j \leftarrow 7 - 1$



Quicksort

Exemplo

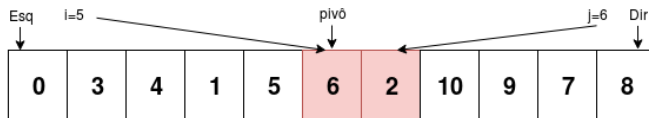
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **falso**, incrementa $i \leftarrow i + 1$.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.



Quicksort

Exemplo

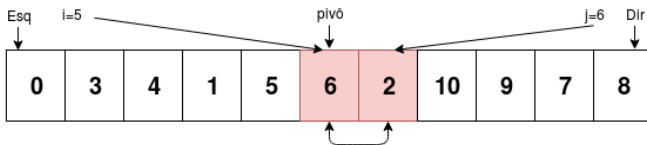
- Verifica se o elemento na posição i é maior que o **Pivô**.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.



Quicksort

Exemplo

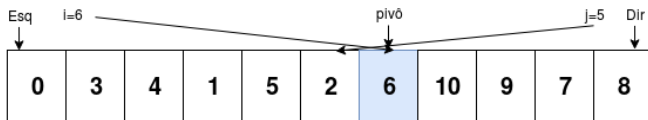
- Troca o elemento na posição i com o elemento na posição j .
 - $V[5] \leftrightarrow V[6]$.
- Em seguida, incrementa i e decrementa j .
 - $i \leftarrow 5 + 1$
 - $j \leftarrow 6 - 1$



Quicksort

Exemplo

- O processo é interrompido quando i e j se cruzam.
 - Ou seja, $i > j$.

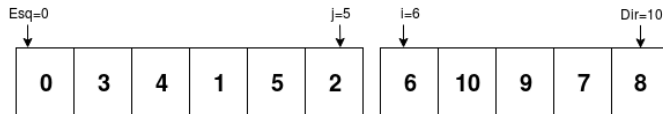


- Em seguida, os vetores são divididos em dois e ordena-se recursivamente cada vetor.
 - O vetor da esquerda $V[Esq..j]$, começa em Esq e termina em j .
 - O vetor da direita $V[i..Dir]$, começa em i e termina em Dir .

Quicksort

Exemplo

- Realiza-se uma chamada recursiva para o vetor da esquerda $V[Esq..j]$
 - Quicksort(V,0,5)
- e uma chamada recursiva para o vetor da direita $V[i..Dir]$
 - Quicksort(V,6,10)

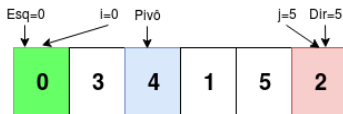


Quicksort

Exemplo

Ordenando recursivamente o vetor da esquerda:

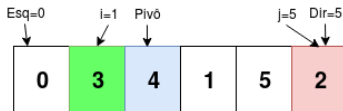
- Inicializa-se as variáveis auxiliares i e j .
 - $i \leftarrow \text{Esq.}$
 - $j \leftarrow \text{Dir.}$
- Calcula-se o **Pivô** $x \leftarrow V[\frac{(\text{Esq} + \text{Dir})}{2}]$
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **falso**, incrementa $i \leftarrow i + 1$.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.



Quicksort

Exemplo

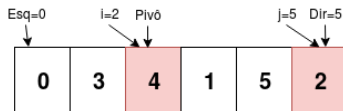
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **falso**, , incrementa $i \leftarrow i + 1$.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.



Quicksort

Exemplo

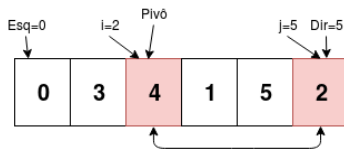
- Verifica se o elemento na posição i é maior que o **Pivô**.
- Verifica se o elemento na posição j é menor que o **Pivô**.
- Se **verdadeiro**, pára.



Quicksort

Exemplo

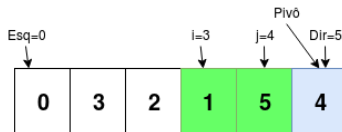
- Troca o elemento na posição i com o elemento na posição j .
 - $V[2] \leftrightarrow V[5]$.
- Em seguida, incrementa i e decrementa j .
 - $i \leftarrow 2 + 1$
 - $j \leftarrow 5 - 1$



Quicksort

Exemplo

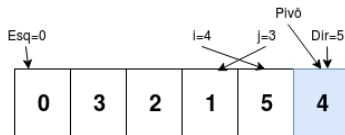
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **falso**, incrementa $i \leftarrow i + 1$.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **falso**, decrementa $j \leftarrow j - 1$.



Quicksort

Exemplo

- O processo é interrompido quando i e j se cruzam.
 - Ou seja, $i > j$.

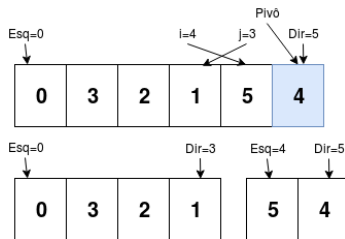


- Em seguida, os vetores são divididos em dois e ordena-se recursivamente cada vetor.
 - O vetor da esquerda $V[Esq..j]$, começa em Esq e termina em j .
 - O vetor da direita $V[i..Dir]$, começa em i e termina em Dir .

Quicksort

Exemplo

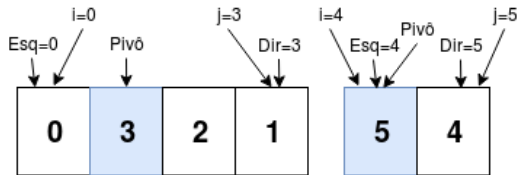
- Realiza-se uma chamada recursiva para o vetor da esquerda $V[Esq..j]$
 - Quicksort($V,0,3$)
- e uma chamada recursiva para o vetor da direita $V[i..Dir]$
 - Quicksort($V,4,5$)



Quicksort

Exemplo

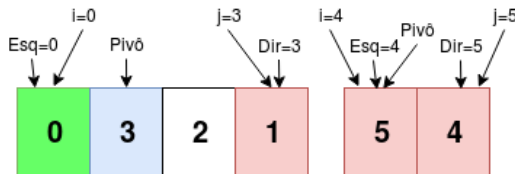
- Inicializa-se as variáveis e calcula-se o pivô em cada vetor.



Quicksort

Exemplo

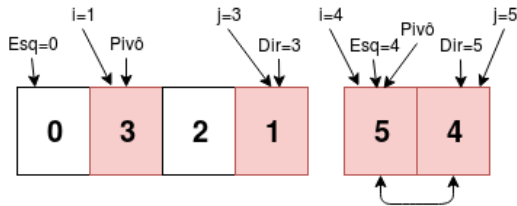
- Vetor da Esquerda:
 - Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **falso**, incrementa $i \leftarrow i + 1$.
 - Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.
- Vetor da Direita:
 - Realiza-se a mesma verificação: ambas são **verdadeiras**.



Quicksort

Exemplo

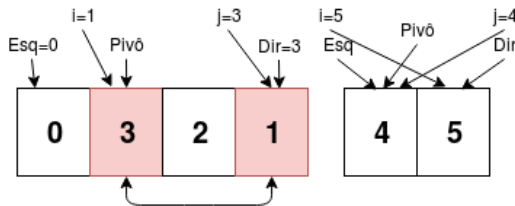
- Vetor da Esquerda:
 - Verifica os elementos na posição i e j em relação ao pivô.
 - Ambos os testes são **verdadeiros**, o laço de repetição pára.
- Vetor da Direita:
 - Realiza-se a troca dos elementos.
 - Incrementa $i \leftarrow i + 1$ e decrementa $j \leftarrow j - 1$.



Quicksort

Exemplo

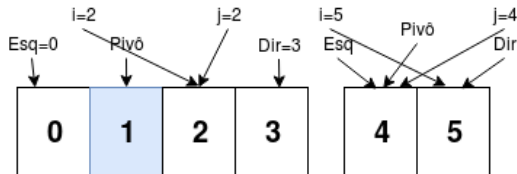
- Vetor da Esquerda:
 - Realiza-se a troca dos elementos.
 - Incrementa $i \leftarrow i + 1$ e decrementa $j \leftarrow j - 1$.
- Vetor da Direita:
 - i e j se cruzaram, laço de repetição pára.



Quicksort

Exemplo

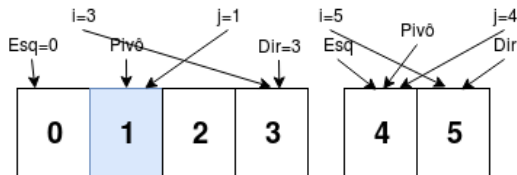
- Vetor da Esquerda:
 - i e j apontam para o mesmo elemento (Não precisa trocar).
 - Incrementa $i \leftarrow i + 1$ e decrementa $j \leftarrow j - 1$.
- Vetor da Direita:
 - Ordenado.



Quicksort

Exemplo

- Vetor da Esquerda:
 - i e j se cruzaram, laço de repetição pára.
 - O vetor está ordenado.
- Vetor da Direita:
 - Ordenado.

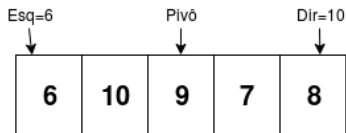


Quicksort

Exemplo

Ainda falta ordenar o vetor da direita:

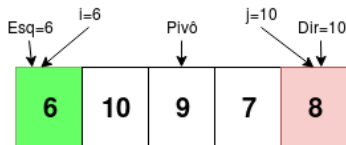
- Inicializa as variáveis i e j e calcula o pivô.



Quicksort

Exemplo

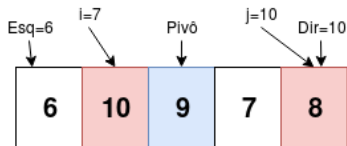
- Verifica se o elemento na posição i é maior que o **Pivô**.
 - Se **falso**, incrementa $i \leftarrow i + 1$.
- Verifica se o elemento na posição j é menor que o **Pivô**.
 - Se **verdadeiro**, pára.



Quicksort

Exemplo

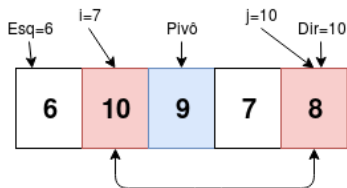
- Verifica se o elemento na posição i é maior que o **Pivô**.
- Verifica se o elemento na posição j é menor que o **Pivô**.
- Se **verdadeiro**, pára.



Quicksort

Exemplo

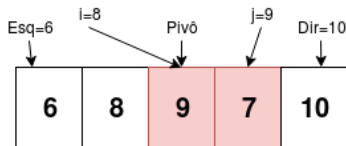
- Realiza-se a troca dos elementos.
- Incrementa $i \leftarrow i + 1$ e decrementa $j \leftarrow j - 1$.



Quicksort

Exemplo

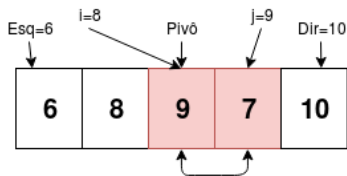
- Verifica se o elemento na posição i é maior que o **Pivô**.
- Verifica se o elemento na posição j é menor que o **Pivô**.
- Se **verdadeiro**, pára.



Quicksort

Exemplo

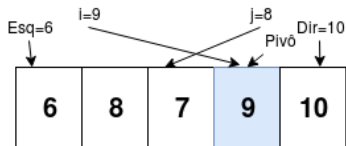
- Realiza-se a troca dos elementos.
- Incrementa $i \leftarrow i + 1$ e decrementa $j \leftarrow j - 1$.



Quicksort

Exemplo

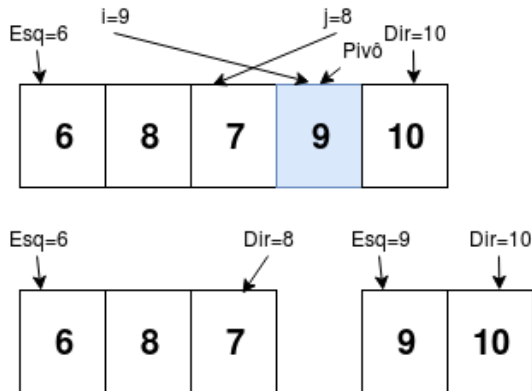
- i e j se cruzaram, laço de repetição pára.
- Realiza as chamadas recursivas.



Quicksort

Exemplo

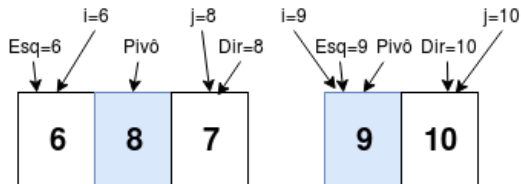
- Realiza uma chamada recursiva para o vetor da esquerda $V[Esq..j]$.
- E uma chamada recursiva para o vetor da direita $V[i..Dir]$.



Quicksort

Exemplo

- Inicializa as variáveis i e j .
- Calcula o pivô.



Quicksort

Exemplo

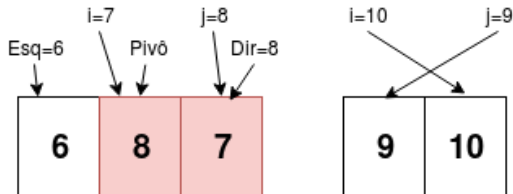
- Vetor da Esquerda:
 - Compara os elementos nas posições i e j com o pivô.
 - Incrementa $i \leftarrow i + 1$.
- Vetor da Direita:
 - Compara os elementos nas posições i e j com o pivô.
 - As condições são **falsas**, incrementa i e decrementa j .



Quicksort

Exemplo

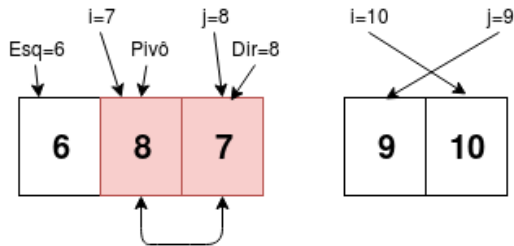
- Vetor da Esquerda:
 - Compara os elementos nas posições i e j com o pivô.
 - As condições são **verdadeiras**, o laço de repetição pára.
- Vetor da Direita:
 - i e j se cruzaram, o laço de repetição pára.
 - O vetor está ordenado.



Quicksort

Exemplo

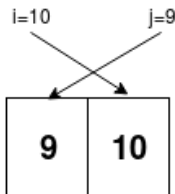
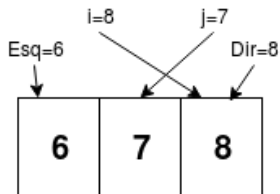
- Vetor da Esquerda:
 - Realiza-se a troca dos elementos.
 - Incrementa $i \leftarrow i + 1$ e decrementa $j \leftarrow j - 1$.
- Vetor da Direita:
 - Ordenado.



Quicksort

Exemplo

- Vetor da Esquerda:
 - i e j se cruzaram, o laço de repetição pára.
 - O vetor está ordenado.
- Vetor da Direita:
 - Ordenado.



QuicksortOrdena

Pseudo-código

Algoritmo 7: QuicksortOrdena

Entrada: Vetor $V[0..n-1]$, tamanho n

Saída: Vetor V ordenado

```
1 início
2   Quicksort( $V, 0, n-1$ )
```

Algoritmo 8: Quicksort

Entrada: Vetor $V[Esq..Dir]$, Esq , Dir

Saída: Vetor V ordenado

```
1 início
2    $(i, j) \leftarrow \text{Particiona}(V, Esq, Dir)$ 
3   se  $(Esq < j)$  então
4     Quicksort( $V, Esq, j$ )
5   se  $(i < Dir)$  então
6     Quicksort( $V, i, Dir$ )
```

Quicksort

Pseudo-código

Algoritmo 9: Particiona

Entrada: Vetor $V[Esq..Dir]$, Esq, Dir

Saída: Vetor V ordenado

```
1 início
2    $i \leftarrow Esq; j \leftarrow Dir$ 
3    $x \leftarrow V[\frac{(i+j)}{2}]$ 
4   repita
5     enquanto (  $x > V[i]$  ) faça
6        $i \leftarrow i + 1$ 
7     enquanto (  $x < V[j]$  ) faça
8        $j \leftarrow j - 1$ 
9     se (  $i \leq j$  ) então
10      Trocar  $V[i] \leftrightarrow V[j]$ 
11       $i \leftarrow i + 1$ 
12       $j \leftarrow j - 1$ 
13 até (  $i > j$  );
14 retorna (  $i, j$  )
```

Quicksort

Pseudocódigo

- O laço de repetição da função Particiona é extremamente simples.
- Razão pela qual o algoritmo Quicksort é tão rápido.

Quicksort

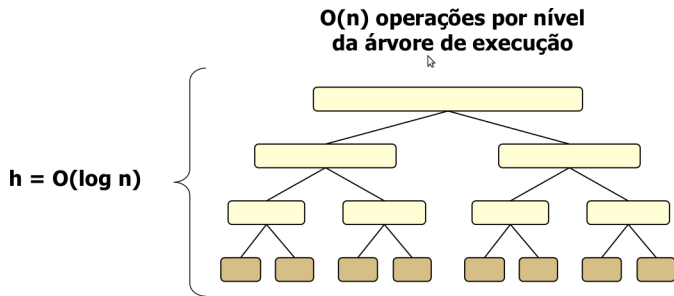
Análise

- Qual o pior caso para o Quicksort?
- Por que?
 - Qual sua ordem de complexidade?
 - Qual o melhor caso?
 - O algoritmo é estável?

Quicksort

Análise

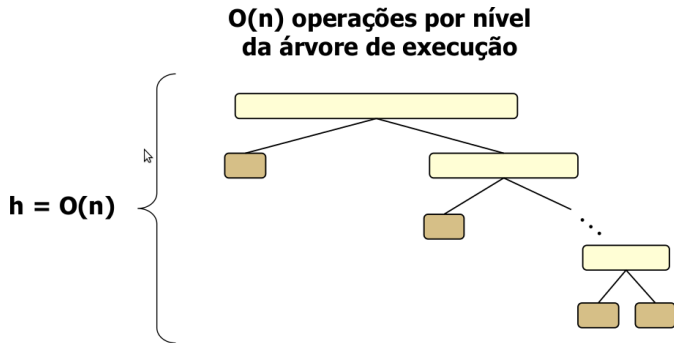
- Melhor caso: $C(n) = 2C(\frac{n}{2}) + n = n \log n$
- Ocorre quando o problema é sempre dividido em subproblemas de igual tamanho após a partição.



Quicksort

Análise

- Pior caso: $C(n) = O(n^2)$
- O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo já ordenado.



Quicksort

Análise

- O pior caso pode ser evitado empregando pequenas modificações no algoritmo.
- Para isso basta escolher três itens quaisquer do vetor e usar a mediana dos três como pivô.

Quicksort

Análise

- Caso médio de acordo com Sedgewick e Flajolet (1996)[p. 17):
- $C(n) \approx 1,386n \log n - 0,846n$.
- Isso significa que em média o tempo de execução do QuickSort é cerca de $O(n \log n)$.

Quicksort

Vantagens x Desvantagens

- Vantagens:
 - É extremamente eficiente para ordenar arquivos de dados.
 - Necessita de apenas uma pequena pilha como memória auxiliar.
 - Requer $O(n \log n)$ comparações em média (caso médio) para ordenar n itens.
- Desvantagens:
 - Tem um pior caso $O(n^2)$ comparações.
 - Sua implementação é delicada e difícil: um pequeno engano pode levar a efeitos inesperados para algumas entradas de dados.
 - O método não é estável.

Quicksort

Melhorias

- Pivô - mediana de três ou mediana de cinco.
- Não empilhar quando tem apenas um item.
- Usar algoritmo de inserção (Insertionsort) para vetores pequenos.
- Escolha correta do lado a ser empilhado primeiro.
- Resultado: melhoria no tempo de execução de 25% a 30%.

Algoritmos e Estrutura de Dados II

Algoritmos de Ordenação

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

