

Universidade Paulista – UNIP

Ciência da Computação

**APS: Atividade Prática Supervisionada**

**Tema: Sistema para análise de performance de**

**algoritmos de ordenação de dados**

|  |  |
| --- | --- |
| *Daniela Ferreira* | *D394BC-7* |
| *Leonardo Henrique Machado* | *D41186-1* |
| *Lucas Merlin de Oliveira* | *D22398-4* |
| *Maycon Ravyer de A. Brasil* | *D2602E-0* |
| *Paulo Henrique Paiva da Silva* | *D4122I-8* |

**São Paulo**

**2018**

**ÍNDICE**

[**1.** **INTRODUÇÃO** 3](#_Toc528824752)

[**2.** **REFERENCIAL TEÓRICO** 5](#_Toc528824753)

[**2.1.** **Ordenação Bolha *(Bubble sort)*** 5](#_Toc528824754)

[**2.2.** **Busca Binária** 6](#_Toc528824755)

[**3.** **DESENVOLVIMENTO** 8](#_Toc528824756)

[**4.** **RESULTADOS E DISCUSSÃO** 9](#_Toc528824757)

[**5.** **CONSIDERAÇÕES FINAIS** 10](#_Toc528824758)

[**6.** **REFERÊNCIAS BIBLIOGRÁFICAS** 11](#_Toc528824759)

[**7.** **CÓDIGO FONTE** 12](#_Toc528824760)

[**8.** **FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS** 13](#_Toc528824761)

# 

# **INTRODUÇÃO**

É inegável o fato de que os métodos de programação, de alguma forma, estão se atualizando. Os modelos antigos estão sendo substituídos e/ou incrementados por novas técnicas e aplicações lógicas a fim de sanar as necessidades tecnológicas atuais.

Porém, ainda assim os métodos antigos ainda são utilizados em massa, seja por conta de sistemas, seja pela aplicabilidade ou até mesmo a “exatidão” em seus resultados.

Em diversas aplicações do dia a dia de um programador, sejam cientificas ou comerciais, iremos nos deparar com problemas de ordenação de elementos, como por exemplo, efetuar a ordenação de uma sequência de números em ordem crescente ou decrescente, ordenar nomes de uma lista de clientes em ordem alfabética, ordenar a valor do salário dos funcionários de um determinado departamento da empresa, entre outros eventos. Para ordenar tais elementos da maneira mais eficaz possível se faz necessário o uso de um algoritmo de ordenação.

No mercado atual existe um número alto de algoritmos de ordenações disponibilizados, obter o conhecimento deles e as suas aplicações é algo crucial para ser um programador respeitado no mercado de trabalho. Conhecendo estes algoritmos, o programador poderá efetuar a melhor escolha do algoritmo a ser utilizado de acordo com a necessidade de cliente e/ou projeto, melhorando o desempenho de sua aplicação.

Com essa ideia em mente, serão apresentadas algumas metodologias de ordenação de dados, que são classificados em:

• Ordenação Interna: onde todos os elementos a serem ordenados cabem na memória principal e qualquer registro pode ser imediatamente acessado.

• Ordenação Externa: onde os elementos a serem ordenados não cabem na memória principal e os registros são acessados sequencialmente ou em grandes blocos.

Essas ordenações se dão por meio de algoritmos, que por sua vez serão avaliadas e comparadas por meios gráficos. Dessa forma, será apresentado o método mais eficiente e nos mais variados testes. Mas antes, vale relembrar um conceito importante:

* ALGORITMO

***“Um algoritmo nada mais é do que uma instrução que mostra passo a passo os procedimentos necessários para a resolução de uma tarefa. Ele não responde a pergunta “o que fazer?”, mas sim “como fazer”. Em termos mais técnicos, um algoritmo é uma sequência lógica, finita e definida de instruções que devem ser seguidas para resolver um problema ou executar uma tarefa*.”**

Trecho disponível em https://www.tecmundo.com.br/

programacao/2082-o-que-e-algoritmo-.html <Acesso em 14/10/2018>

Explicando em termos simples, um algoritmo pode ser comparado á uma receita de bolo. A receita de bolo te mostra o passo-a-passo para fazer e como ele deverá ficar. Em programação, o algoritmo será responsável pelas instruções do que o programa deverá fazer.

# **REFERENCIAL TEÓRICO**

Após a breve explicação sobre o conceito de ordenação e algoritmos, abaixo será apresentado os métodos existentes. Posteriormente será mostrada sua execução na prática:

# ***BubbleSort (Ordenação Bolha)***

O método de ordenação bolha ou *bubble sort*, é um dos métodos de ordenação mais simples inventados até o momento, pode ser aplicado em vetores e listas dinâmicas. A ideia principal deste algoritmo é percorrer o vetor diversas vezes e a cada volta levar para o topo do vetor o maior elemento do mesmo. Essa movimentação dos elementos lembra a forma que as bolhas procuram o seu próprio nível, com base nisto gerou-se o nome deste algoritmo.

No melhor dos casos este método executa **n** operações relevantes, sendo n o número total de elementos do vetor. No pior dos casos serão feitas **n²** operações. Este algoritmo não é recomentado para programas e aplicações que necessitem de velocidade ou que operem com uma quantidade elevada de dados.

Este algoritmo percorre a lista de itens do vetor do início ao fim, analisando a ordem dos elementos dois a dois e os trocando de lugar caso seja necessário. Ele irá percorrer a lista até que nenhum elemento tenha sido alterado de lugar na última passagem pelo vetor.

Abaixo temos um exemplo em pseudocódigo deste método:

**var x,y,aux : INTEIRO**

**PARA x=0 ATE vet.tamanho PASSO 1  
    PARA y=x+1 ATE vet.tamanho PASSO 1  
     SE (vet[x] > vet[y]) ENTAO  
        var aux = vet[x]  
       vet[x] = vet[y]**

**vet[y] = aux  
FIMSE**

**FIMPARA**

**FIMPARA**

# **HeapSort**

*“O HeapSort é uma estrutura de dados composta por chaves que suporta suas operações básicas: inserção de um novo item e remoção do item com a maior chave, e a chave de cada item reflete a prioridade (ordem) em que se deve tratar aquele item/elemento.”*

*Trecho disponível em https://homepages.dcc.ufmg.br/*

*~cunha/teaching/20121/aeds2/heapsort.pdf <acesso em 22/10/2018>*

Para entender oHeapSort, é importante saber e/ou lembrar o que é uma árvore binária:

A árvore binária é um conjunto de registros que satisfazem as condições impostas, e registos por sua vez podem ser classificados como um “pacote” de variáveis que podem ser de tipos diferentes, e cada campo do registro é composto por uma variável. Na árvore, os registros são denominados “nós”, e cada nó contém o endereço de acesso para a memória.

*“O conjunto de índices de qualquer vetor  v [1...m]  pode ser encarado como uma árvore binária da seguinte maneira*:

* + *O índice 1 é a raiz da árvore;*
  + *O pai de qualquer índice  f  é  f/2  (1 não tem pai);*
  + *O filho esquerdo de um índice  p  é  2p  (esse filho só existe se 2p ≤ m );*
  + *O filho direito de  p  é  2p+1  (esse filho só existe se 2p+1 ≤ m ).*

*(Levando em consideração que o índice do vetor será sempre inteiro)”*

*Trecho disponível em https://www.ime.usp.br/~pf/*

*algoritmos/aulas/hpsrt.html <acesso em 22/10/2018>*

No caso, o algoritmo do HeapSort usa um *HEAP, que é uma árvore binária com as seguintes propriedades:*

* *“O valor de cada nó não é menor que os valores armazenados em cada filho.”*
* *“A árvore é perfeitamente balanceada e as folhas no último nível estão todas nas posições mais à esquerda.”*

*“Um heap (= monte) é um vetor em que o valor de todo pai é maior ou igual ao valor de cada um de seus dois filhos. Mais precisamente, um vetor v [1...m] é um heap se v[f/2] ≥ v[f] para f = 2, . . . , m . (a expressão f/2 deve ser entendida como [f/2] )”*

*Trecho disponível em https://www.ime.usp.br/~pf/*

*algoritmos/aulas/hpsrt.html <acesso em 22/10/2018>*

Concluindo, o HeapSort ordena os elementos com base na comparação em relação à estrutura de dados do *heap binário,* (árvore binária, que no HeapSort é apresentado em forma de vetor) no qual o elemento maior é colocado na base para comparações (para ordenação crescente) ou o valor mínimo na base (para ordenação decrescente), esses são classificados como Heap máximo e Heap mínimo, respectivamente, sendo os valores de comparação a raiz da “árvore”. E para a ordenação completa, esse mesmo processo de comparação é aplicado para os elementos restantes.

Esse método não está restrito apenas à ordenação, ele pode ser aplicado em, por exemplo, paginação de memória, simulação de eventos, sistemas operacionais e etc. Sendo que sua vantagem é uma ordenação rápida e o comportamento de “dividir e conquistar” no caso de muitos elementos (O(n lg n)), em contrapartida é um algoritmo não estável e não é o mais rápido existente para a ordenação.

# **QuickSort**

É o método tecnicamente mais popular para ordenação. Basicamente o QuickSort utiliza o método “Dividir e Conquistar” *(Divide and Conquer).* Um elemento de valor qualquer dentro do array (dependendo de como serão construídas as partições) é escolhido como “pivô”, e esse pivô é a chave da ordenação escolhida, e a partir disso acontece uma partição da matriz dada.

O principal método do QuickSort é o *Partition(),* cujo segue a regra de que:

* *Todos os valores menores que o pivô fiquem antes deste;*
* *Todos os valores maiores que o pivô fiquem depois deste;*

*Trecho disponível em https://www.dcc.fc.up.pt/~pbv/*

*aulas/progimp/teoricas/teorica19.html <acesso em 13/11/2018>*

E todo esse processo deve ser feito em tempo linear. E recursivamente devem-se ordenar as subsequências.

Também existem versões variadas do QuickSort, cujo selecionam o pivô de forma diferente, como por exemplo:

* *“Sempre escolha o primeiro elemento como pivô;*
* *Escolha sempre o último elemento como pivô (implementado abaixo);*
* *Escolha um elemento aleatório como pivô;*
* *Escolha mediana como pivô.”*

*Trecho disponível em https://www.geeksforgeeks.org/quick-sort/*

*<acesso em 13/11/2018>*

Geralmente é recomendado a escolha de um valor mediano como pivô.

O processo de ordenação é separado em processos específicos. Pode-se dizer que logo após que o algoritmo de partição é executado, uma boa parte do trabalho já é facilitado, já que os valores estão divididos entre menor e maior que o pivô definido, ao passo que esse é processo mais delicado do algoritmo. E cada vez que um novo pivô é escolhido, os agrupamentos ficam cada vez mais ordenados.

Ou seja, são utilizadas chamadas recursivas para ordenar os subarrays de “cada lado” (os elementos maiores/menores em relação ao pivô), desse modo, ao final das partições, cada lado estará devidamente ordenado. Esse método checa se os subarrays contém um ou mais elementos, e este é o critério para as chamadas recursivas.

Dentre os métodos de ordenação, esse é considerado um dos mais rápidos, podendo ser aplicado para os mais variados tipos de situações, visto que além do método de partição conforme anteriormente citado, ele funciona em O(NlogN) para ordenações internas ou em memória, mas para ordenar informações em disco, são recomendados outros métodos.

Uma das suas desvantagens é que ele não é estável, podendo apresentar diversos erros.

1. **DESENVOLVIMENTO**

# **RESULTADOS E DISCUSSÃO**

# **CONSIDERAÇÕES FINAIS**

# **REFERÊNCIAS BIBLIOGRÁFICAS**

<https://www.embarcados.com.br/algoritmos-de-ordenacao-bubble-sort/> <acessado em 31/10/2018>

<https://pt.wikipedia.org/wiki/Bubble_sort> <data de acesso: 29/10/2018>

<https://www.devmedia.com.br/entendendo-o-algoritmo-bubble-sort-em-java/24812> <data de acesso: 01/11/2018>

<https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/> <data de acesso: 29/10/2018>

[www.forum.imasters.com.br](http://www.forum.imasters.com.br) <acessado em 21/10/2018>

[www.larback.com.br](http://www.larback.com.br) <acessado em 21/10/2018>

[www.facom.ufms.br](http://www.facom.ufms.br) <acessado em 21/10/2018>

[www.ic.unicamp.br](http://www.ic.unicamp.br) <acessado em 21/10/2018>

[www.stackoverflow.com](http://www.stackoverflow.com) <acessado em 21/10/2018>

<https://www.geeksforgeeks.org/heap-sort/> <acessado em 21/10/2018>

<https://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html> <acessado em 21/10/2018>

<https://www.cos.ufrj.br/~rfarias/cos121/aula_09.html> <acessado em 22/10/2018>

<https://www.ime.usp.br/~pf/algoritmos/aulas/bint.html> <acessado em 22/10/2018>

<https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/heapsort.pdf> <acessado em 22/10/2018>

<http://www.ufjf.br/jairo_souza/files/2009/12/2-Ordena%C3%A7%C3%A3o-HeapSort.pdf> <acessado em 22/10/2018>

<https://www.dcc.fc.up.pt/~pbv/aulas/progimp/teoricas/teorica19.html> <acessado em 13/11/2018>

<http://www.ufjf.br/jairo_souza/files/2009/12/2-Ordena%C3%A7%C3%A3o-QuickSort.pdf> <acessado em 13/11/2018>

<https://www.geeksforgeeks.org/quick-sort/> <acessado em 13/11/2018>

<https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/quicksort.pdf> <acessado em 13/11/2018>

# **CÓDIGO FONTE**

# **FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS**