**UNIVERSIDADE CIDADE DE SÃO PAULO**

**CIÊNCIA DA COMPUTAÇÃO**

**GIOVANNA QUERINO DA SILVA**

**GUILHERME GONÇALVES COLHADO VEIGA DE SOUZA**

**HELTON DIAS MARQUES**

**LEONARDO MIRANDA DE TOLEDO**

**PAULO VINÍCIUS DE SOUZA**

**STHEFANY STHER PEQUENO ALVES**

**RELATÓRIO DE PESQUISA**

**ALGORITMOS DE MÉTODOS DE ORDENAÇÃO**

**SÃO PAULO - SP**

**2021**

**Giovanna Querino da Silva**

**Guilherme Gonçalves Colhado Veiga de Souza**

**Helton Dias Marques**

**Leonardo Miranda de Toledo**

**Paulo Vinícius de Souza**

**Sthefany Sther Pequeno Alves**

**RELATÓRIO DE PESQUISA:**

**ALGORITMOS DE MÉTODOS DE ORDENAÇÃO**

Trabalho de Estrutura de Dados I apresentado ao professor Juliano Ratusznei do curso de Ciência da Computação da instituição Universidade Cidade de São Paulo

**SÃO PAULO - SP**

**2021**

**LISTA DE FIGURAS**

[Figura 1 - Bubble Sort em linguagem C 9](#_Toc70333291)

[Figura 2 - Execução do código da figura 1. 10](#_Toc70333292)

[Figura 3 - Execução do código da figura 1 com alteração do sinal da linha 28. 10](#_Toc70333293)

Figura 4 - Como é feito o método de ordenação.......................................................................11

Figura 5 - Função do Quicksort em C.......................................................................................12

Figura 6 - Código acima sendo executado................................................................................13

Figura 7 - Exemplo Visual do Método de Ordenação Merge funcionando..............................14

Figura 8 - Primeira Parte do Código de Merge Sort em Linguagem C....................................15

Figura 9 - Segunda Parte do Código de Merge sort em Linguagem C.....................................16

Figura 10 - Tela de Execução dos Códigos das Figuras 8 e 9..................................................16

Figura 11 - Vetor desorganizado...............................................................................................17

Figura 12 - Vetor sendo organizado..........................................................................................17

Figura 13 - Vetor sendo organizado..........................................................................................18

Figura 14 - Vetor organizado....................................................................................................18

Figura 15 - Código Shell sort em C++......................................................................................18

Figura 16 - Código sendo executado.........................................................................................19

Figura 17 - Exemplo de Código HeapSort em Linguagem C...................................................22

Figura 18 - Código da imagem acima sendo executado...........................................................22

Figura 19 - Exemplo de código Bucket Sort em Linguagem C++............................................24

Figura 20 - Código da Imagem acima sendo executado...........................................................25

[Figura 21 - Explicação das Iterações do Cocktail Sort 26](#_Toc70333294)

[Figura 22 - Exemplo de Método de Ordenação *Cocktail Sort* em linguagem C 28](#_Toc70333295)

[Figura 23 - Tela de Execução do código com o método *Cocktail Sort* 29](#_Toc70333296)

Figura 24 – Função do Radix Sort em código C.......................................................................31

Figura 25 – Código acima sendo executado.............................................................................31

**LISTA DE TABELAS**

[Tabela 1 - Decrescente 7](#_Toc70333355)

[Tabela 2 - Crescente 7](#_Toc70333356)

Tabela 3- Lista Desordenada.....................................................................................................11

Tabela 4 – Lista Ordenada pelo lado esquerdo do pivô............................................................11

Tabela 5 – Lista Ordenada pelo lado direito do pivô................................................................12

Tabela 6 – Vetor desorganizado................................................................................................30

Tabela 7 – Vetor sendo organizado...........................................................................................30

Tabela 8 – Vetor sendo organizado...........................................................................................30

Tabela 9 – Vetor organizado.....................................................................................................31

**SUMÁRIO**

**1 INTRODUÇÃO......................................................................................................................6**

**2 BUBBLE SORT......................................................................................................................7**

2.1 Aplicação em Código.....................................................................................................8

2.2 Vantagens e Desvantagens.............................................................................................9

**3 QUICK SORT......................................................................................................................11**

3.1 Aplicação em Código...................................................................................................12

3.2 Vantagens e Desvantagens...........................................................................................13

**4 MERGE SORT.....................................................................................................................14**

4.1 Vantagens e Desvantagens...........................................................................................14

4.2 Aplicação em Código...................................................................................................15

**5 SHELL SORT......................................................................................................................17**

5.1 Aplicação em Código...................................................................................................18

5.2 Vantagens e Desvantagens...........................................................................................19

**6 HEAP SORT.........................................................................................................................20**

6.1 Vantagens e Desvantagens...........................................................................................21

6.2 Aplicação em Código...................................................................................................21

**7 BUCKET SORT...................................................................................................................23**

7.1 Vantagens e Desvantagens...........................................................................................23

7.2 Aplicação em Código...................................................................................................24

**8 COCKTAIL SORT..............................................................................................................26**

8.1 Vantagens e Desvantagens...........................................................................................27

8.2 Aplicação em Código...................................................................................................27

**9 RADIX SORT.......................................................................................................................30**

9.1 Aplicação em Código...................................................................................................31

9.2 Vantagens e Desvantagens...........................................................................................32

**10 CONCLUSÃO....................................................................................................................33**

**REFERÊNCIAS......................................................................................................................34**

**1. INTRODUÇÃO**

Esta pesquisa foi produzida a partir de orientações dadas pelo professor Juliano Ratusznei, na matéria de Estrutura de Dados I. A pesquisa tem como tema principal, os métodos de ordenação, em linguagens de programação, e a sua importância em listas, arrays, vetores e matrizes.

Cada método de ordenação tem as suas vantagens e desvantagens, e a sua aplicação no código varia conforme a necessidade de cada programa, as principais características para se considerar na hora de escolher um método de ordenação, seriam: composição do vetor, estabilidade do código, complexidade de aplicação e memória exigida.

A utilização equivocada de um método em um determinado programa pode acarretar várias consequências, por exemplo: lentidão na execução e possível falta e gasto de memória desnecessária.

**2. BUBBLE SORT**

*Bubble Sort* é um algoritmo de ordenação (serve para colocar um conjunto de itens em uma ordem) mais simples, que compara elementos adjacentes, que poderá ser aplicado arrays e listas dinâmicas.

Por exemplo, se o seu objetivo é ordenar valores de forma crescente ou decrescente a posição atual é comparada com elementos dois a dois, ou seja, compara a primeira posição com a segunda, na segunda iteração, compara a segunda com a terceira e assim sucessivamente.

**Tabela 1 - Decrescente**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 30 | 100 | 5 | 422 | INÍCIO |
| 100 | 30 | 5 | 422 | PRIMEIRA TROCA |
| 100 | 30 | 422 | 5 | SEGUNDA TROCA |
| 100 | 422 | 30 | 5 | TERCEIRA TROCA |
| 422 | 100 | 30 | 5 | ÚLTIMA TROCA |

**Fonte: Autoria Própria**

**Tabela 2 - Crescente**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 30 | 100 | 5 | 422 | INÍCIO |
| 30 | 5 | 100 | 422 | PRIMIERA TROCA |
| 5 | 30 | 100 | 422 | SEGUNDA TROCA |
| 5 | 30 | 100 | 422 | TERCEIRA TROCA |
| 5 | 30 | 100 | 422 | ÚLTIMA TROCA |

**Fonte: Autoria Própria**

As tabelas exemplificam como o *bubble sort* funcionaria em um código, trocando de lugar os números a cada dois. Na tabela 1, o código será executado até cumprir a sua função, comparando com todos os números do vetor. Na tabela 2 o vetor fica ordenado de forma mais rápida, porém o código ainda será executado até que se compare com todos os números do vetor.

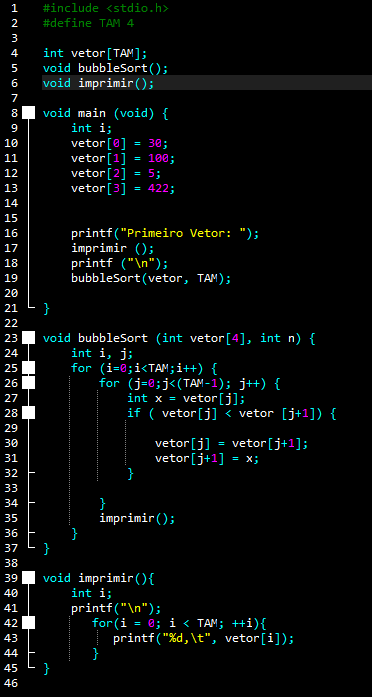
Na tabela 1 o código ordena o vetor para que ele fique de forma decrescente, significando que o primeiro número será sempre maior que os seguintes.

Já na tabela 2, o código deixa o vetor de forma crescente, indicando que o primeiro número deverá ser sempre menor que os demais, e assim por diante.

**2.1. Aplicação em Código**

Um exemplo de código em *bubble sort* na linguagem C pode ser visto nas figuras 1 e 2, ordenando o vetor de forma decrescente:

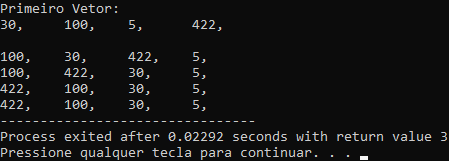
**Figura 1 -** Bubble Sort em linguagem C

****

**Fonte: Autoria Própria**

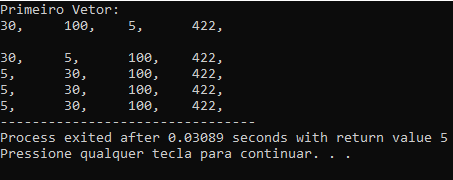
Na linha 28, se alterarmos o sinal de menor (<) para o maior (>), ele passará a ordenar a lista de forma crescente.

**Figura 2** - Execução do código da figura 1.

****

**Fonte: Autoria Própria**

**Figura 3 -** Execução do código da figura 1 com alteração do sinal da linha 28.

****

**Fonte: Autoria Própria**

**2.2. Vantagens e Desvantagens**

**Vantagens:**

- O algoritmo é simples e com isso facilita o aprendizado;

- Os elementos mudam de lugar sem utilizar armazenamento temporário (quando o seu computador fosse desligado estes dados seriam perdidos). Então precisando de menos espaço.

O melhor caso para que o *bubble sort* seja executado da melhor forma é enviar uma lista com itens já ordenados na entrada, pois o algoritmo irá percorrer a lista apenas uma vez, dado que após a primeira iteração os itens vão estar ordenados. Assim, seu tempo de execução no melhor caso é percorrer toda a lista apenas uma vez.

**Desvantagens:**

- Lentidão;

- Não apresenta bons resultados com listas de muitos itens.

O pior caso para *bubble sort*, é que caso tenha uma lista de 10 itens decrescentes, haverá 100 iterações para ordená-la de forma crescente, pois os menores valores estarão no final da lista, levando muitas iterações para levá-los para o começo.

**3. QUICK SORT**

O *QuickSort* é um método de ordenação, criado pelo cientista da computação Charles Antony Richard Hoare. Este método é muito utilizado, pois ele é muito rápido e eficiente.

Ele usa a estratégia de dividir e conquistar, ele vai dividindo os elementos e colocando em ordem a partir do seu pivô. Como é feito isso: é determinado qual será o pivô (pode ser feito de modo aleatório), depois ele irá separar os elementos menores que o pivô para a esquerda e maiores para a direita, determinado quais são maiores e menores, ele recursivamente irá colocar na ordem o lado esquerdo e direito.

**Figura 4 - Como é feito o método de ordenação.**



**Fonte: Disponível em: https://pt.wikipedia.org/wiki/Quicksort Acesso em: 22/04/2021.**

Vamos dar um exemplo de como funciona esse método com a seguinte sequência.

< 2 , 3 , 1 , 5 ,6 , 7 , 4 >

O pivô dessa sequência será o número 3, feito isso vamos determinar os números que são menores e colocar à esquerda do pivô, e os que são maiores à sua direita:

**Tabela 3 - Lista Desordenada**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 2 | 1 | 3 | 5 | 6 | 7 | 4 |

**Fonte: Autoria Própria**

Após feito isso, vamos reordenar o lado esquerdo do nosso vetor:

**Tabela 4 - Lista Ordenada pelo lado esquerdo do pivô**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 5 | 6 | 7 | 4 |

**Fonte: Autoria Própria**

Feito isso, vamos fazer o mesmo com o lado direito:

**Tabela 5 - Lista Ordenada pelo lado direito do pivô**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

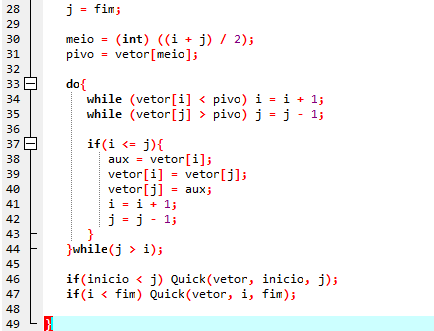
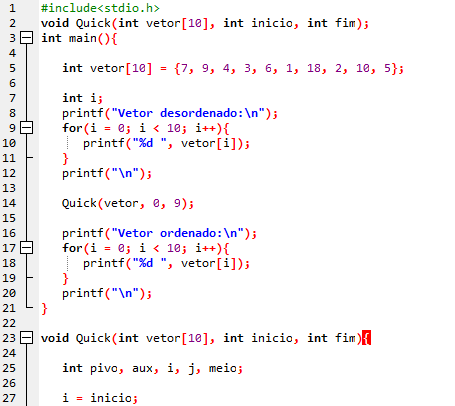
**Fonte: Autoria Própria**

Sempre comparando com o pivô, para ver como será a sua ordem.

**3.1. Aplicação em Código**

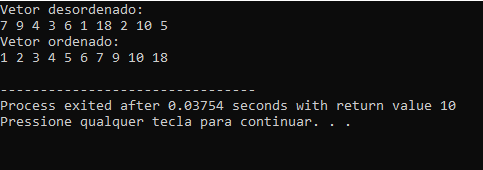
Agora vamos ver como é esse processo em algoritmo.

**Figura 5 - Função do *Quicksort* em C.**



**Fonte: https://www.vivaolinux.com.br/script/Ordenacao-QuickSort. Acesso em: 22/04/2021.**

**Figura 6 - Código acima sendo executado.**



**Fonte: Autoria Própria.**

**3.2. Vantagens e Desvantagens**

Vamos começar com as vantagens:

* Método rápido e muito eficaz.
* Utiliza o O(n log n) comparações em média.

Agora vamos com as desvantagens:

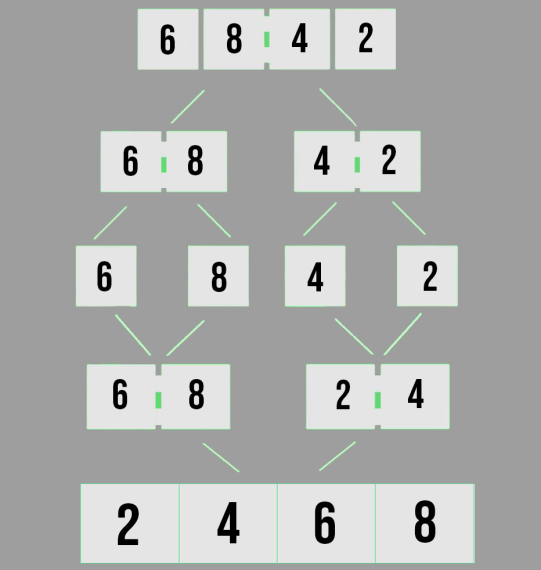
* O método não é estável.

**4. MERGE SORT**

O *Merge Sort* faz a sua ordenação através de fusão/intercalação e*,* assim como o *Quick Sort,* éum algoritmo com o método de “Dividir para Conquistar”, que consiste em dividir o problema em “subproblemas”; ir resolvendo os subproblemas de forma recursiva; assim, encontrando a solução para a situação final.

O algoritmo do *Merge Sort* vai dividindo a lista de valores até que sobrem apenas listas com um único valor, depois ele vai agrupando os números novamente, mas ele também os vai ordenando, deixando os menores números na frente, e os maiores no final, até que a sequência tenha o mesmo número de valores originalmente. Isso pode ser mais bem visualizado na Figura 7:

Figura 7 - Exemplo Visual do Método de Ordenação Merge funcionando



**Fonte: Autoria Própria**

Entre 1944 e 1946 um ​húngaro-americano chamado John von Neumann(1903 - 1957), ajudou a preparar um relatório sobre a capacidade dos computadores e teve uma participação ativa na criação do Merge Sort, que por sua vez foi um dos primeiros estilos de classificação propostos para computadores.

**4.1. Vantagens e Desvantagens**

**Vantagens**

• Cálculo do Merge Sort é O (n log n)

• Recomendado para aplicações que tem alguma restrição de tempo (executa sempre em um determinado tempo para n)

• Possibilita a transformação em estável

• Fácil Implementação

**Desvantagens**

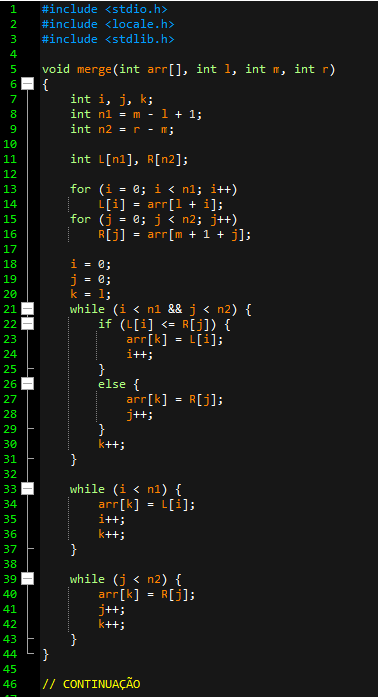
• Utiliza memória auxiliar

• Na prática é mais lento que *Quicksort* no caso médio

**4.2. Aplicação em Código**

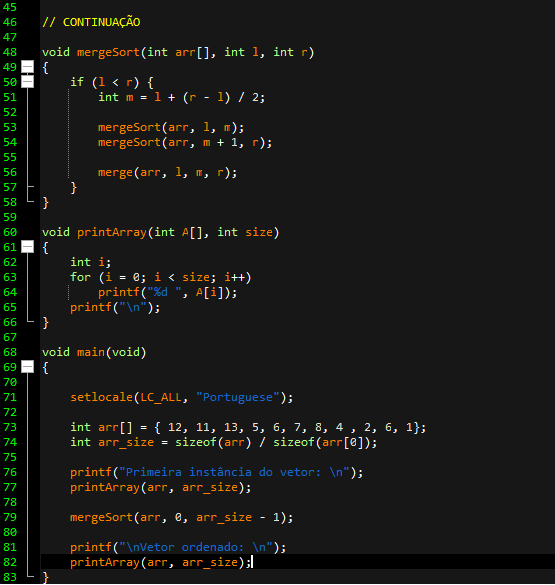
O código de aplicação onde o *Merge Sort* ordena um vetor desordenado pode ser visto nas Figuras 8 e 9, e o *output* do código na Figura 10:

Figura 8 - Primeira Parte do Código de Merge Sort em Linguagem C



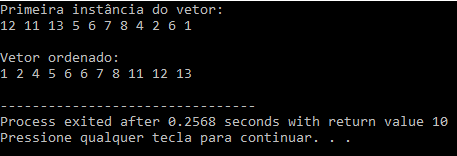
**Fonte: https://www.geeksforgeeks.org/merge-sort/. Acesso em: 25/04/2021**

Figura 9 - Segunda Parte do Código de Merge Sort em Linguagem C



**Fonte: https://www.geeksforgeeks.org/merge-sort/. Acesso em: 25/04/2021**

Figura 10 - Tela de Execução dos Códigos das Figuras 8 e 9



**Fonte: Autoria Própria**

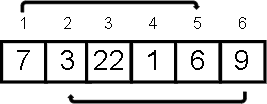
**5. SHELL SORT**

Donald L. Shell (1 de março de 1924 - 2 de novembro de 2015) um cientista da computação, projetou o Shell Sort (que teve como base o Insert Sort)em 1959 e foi publicado pela Universidade de Cincinnati. O algoritmo tem uma complexidade que é desconhecida, ninguém ainda foi capaz de encontrar uma fórmula fechada para sua função de complexidade e o método não é estável. Mesmo sendo considerado como um refinamento do Insert Sort, ele se difere pelo fato de ao invés de identificar o vetor a ser ordenado como um único segmento, ele considera outros vários segmentos.

O Shell Sort permite a troca de registros que estão distantes um do outro, diferente dos algoritmos de ordenação por inserção que possui a troca de itens adjacentes para determinar o ponto de inserção. Os itens que estão separados de h posições (itens distantes) são ordenados: o elemento na posição x é comparado e trocado (caso satisfaça a condição de ordenação) com o elemento na posição x-h. O processo se repete até h=1, quando estiver com a condição satisfeita o algoritmo se torna equivalente ao método de inserção.

Sua etapa final se torna bem similar ao Insert Sort, porém, sobre um vetor de dados quase completamente classificado.

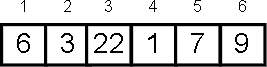
**Figura 11 -** Vetor desorganizado



**Fonte: Autoria Própria**

Na primeira passada (h=4), o item 7 é comparado com 6 (posições 1 e 5) e foram trocados. O item 3 é a seguir comparado e trocado com 9 (posições 2 e 6).

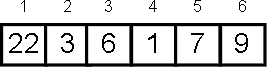
**Figura 12 -** Vetor sendo organizado



**Fonte: Autoria Própria**

Na segunda passada (h=2), 6, 22 e 7 (posições 1, 3 e 5) são rearrumados para resultar em 22, 6 e 7 nestas mesmas posições, da mesma forma 3, 1 e 9 (posições 2, 4 e 6) são comparados e mantidos nos seus lugares.

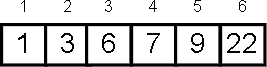
**Figura 13 -** Vetor sendo organizado



**Fonte: Autoria Própria**

A última passada (h=1) corresponde ao algoritmo de inserção, mas apenas trocas locais serão executadas.

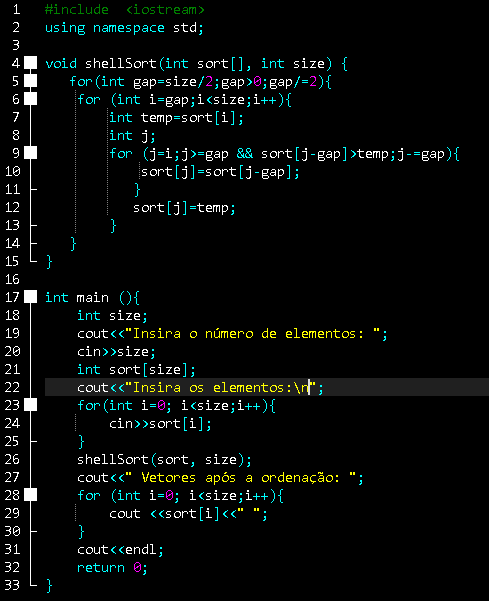
**Figura 14 -** Vetor organizado



**Fonte: Autoria Própria**

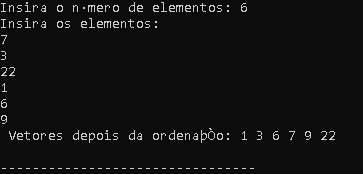
**5.1. Aplicação em Código**

**Figura 15 -** Código Shell Sort em C++



**Fonte: https:/**[**/ww**](http://www.youtube.com/watch?v=AP_C0amgXjM)**w**[**.youtube.com/watch?v=AP\_C0amgXjM**](http://www.youtube.com/watch?v=AP_C0amgXjM)**. Acesso em: 22/04/2021**

**Figura 16 -** Código sendo executado



**Fonte: Autoria Própria**

**5.2. Vantagens e Desvantagens**

**Vantagens:**

* *Shell Sort* é uma ótima opção para arquivos de tamanho moderado (médio);
* Sua implementação é simples e requer uma quantidade de código pequena;

**Desvantagens:**

* O tempo de execução do algoritmo é sensível à ordem inicial do arquivo;
* O método não é estável;

**6. HEAP SORT**

O algoritmo foi criado por John Williams e Robert W. Floyd (pioneiro da computação) em 1964.

"Antigamente, os programadores apenas mexiam nos programas até que parecessem funcionar", diz o professor emérito de The Art of Computer Programming Donald Knuth. "Floyd mostrou que havia uma maneira de provar que os programas funcionariam." Sua abordagem de casar a matemática com a ciência da computação foi "uma revelação para o campo", diz Knuth. John McCarthy, professor emérito de ciência da computação também elogia bastante Robert W.Floyd.

Método de ordenação com funcionamento parecido com o da ordenação por seleção.

Selecione o maior (ou menor) item do vetor e a seguir troque-o com o item que está na Heapsort a seguir troque-o com o item que está na última (ou primeira) posição do vetor; repita estas duas operações com os n - 1 itens restantes; depois com os n – 2 itens; e assim sucessivamente.

**Vetores e Árvores Binárias**

* O índice 1 é a raiz da árvore;
* O pai de qualquer índice f é f/2 (é claro que 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).

O algoritmo Heapsort possui uma estrutura de dados conhecida como heap, que possui como funcionalidade enxergar o vetor como uma árvore binária. Essa estrutura é dividida em: max-heap e min-heap; trataremos aqui apenas do primeiro, omitindo o prefixo max.

É fácil rearranjar os elementos um vetor de inteiros v[1..m] para que ele se torne um heap. Para isso deve-se repetir o processo enquanto o valor de um filho for maior que o valor do pai: trocar os valores de pai e filho e subir um passo em direção à raiz.

**Função Peneira**

O princípio de muitos algoritmos manipuladores de heaps é ter uma função que desce em direção à base da árvore. Essa função, comumente chamada de peneira, recebe um vetor qualquer v[1..m] e faz v[1] descer até sua posição correta, pulando de camada em camada.

**6.1. Vantagens e Desvantagens**

**Vantagens**

* Além de ser mais rápido no pior caso que o QuickSort, necessita de menos memória para executar.
* O comportamento do Heapsort é sempre O(n log n), qualquer que seja a entrada.

**Desvantagens**

* Não é estável.
* O anel interno do algoritmo é bastante complexo.

**6.2. Aplicação em Código**

* Rearranja os elementos de um vetor, deixando-os em ordem crescente (do menor para o maior).
* Deve-se utilizar "static" para indicar que a construção do heap é uma função auxiliar.
* No código, o índice pula de uma camada do vetor para a camada anterior.
* Cada iteração começa com um índice e escolhe o filho com maior valor.

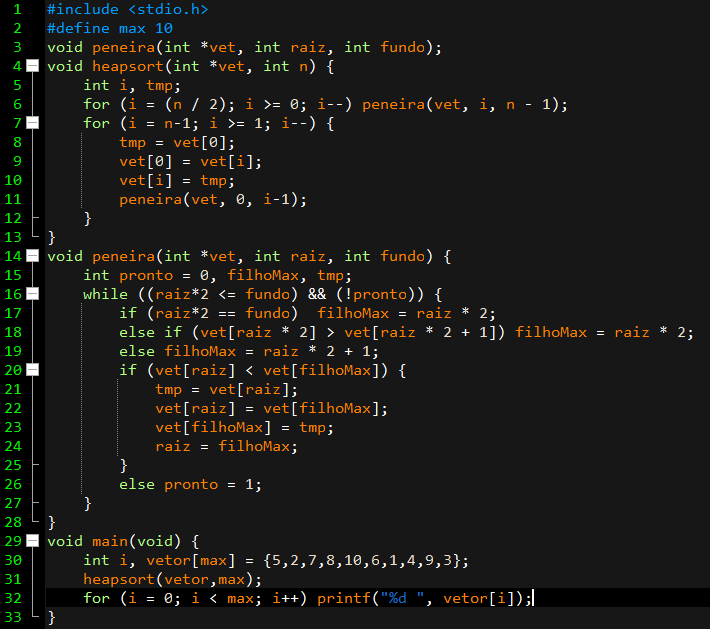


Figura 17 - Exemplo de Código HeapSort em Linguagem C

**Fonte:** [**https://www.portalgsti.com.br/2018/05/organizando-dados-em-heap-e-o-**](https://www.portalgsti.com.br/2018/05/organizando-dados-em-heap-e-o-heapsort-em-c-parte-2.html)[**heapsort-em-c-parte-2.html**](https://www.portalgsti.com.br/2018/05/organizando-dados-em-heap-e-o-heapsort-em-c-parte-2.html)**, 2021. Acesso em: 14/04/2021**

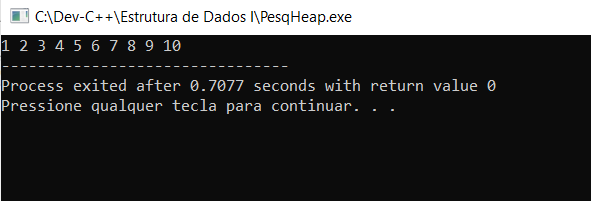


Figura 18 - Código da imagem acima sendo executado

**Fonte: Autoria Própria**

**7. BUCKET SORT**

O Bucket Sort se originou do método de Counting Sort, para o qual a classificação de intervalo se desdobra, se os intervalos possuírem tamanho 1. O Counting Sort ocorre ao separar cada número da matriz original e, em seguida, coloca-o novamente no final. O Bucket Sort faz a mesma coisa, exceto com vários números ao mesmo tempo.

**Definição**

* Elementos distribuídos uniformemente num intervalo [0,1);
* O Bucket Sort divide o intervalo [0,1) em n subintervalos com tamanhos iguais, conhecidos como baldes;
* Devem-se ordenar os números em cada balde, depois examinar os baldes em ordem e listar seus elementos;
* Também se devem concatenar as listas aos compartimentos diretamente associados, para obtermos a ordenação final.

**7.1. Vantagens e Desvantagens**

**Vantagens**

* Estável: Ele não modifica a ordem dos dados iguais (caso seja usado um algoritmo não estável nos baldes);
* Ele possui processamento simples que é muito parecido com o Counting Sort, mas com baldes mais sofisticados.

**Desvantagens**

* Todos os dados devem estar uniformemente distribuídos;
* Não é recomendado para grandes conjuntos;
* Ordena os valores inteiros que são positivos (Talvez altere para outros valores).

**7.2. Aplicação em Código**

Figura 19 - Exemplo de Código Bucket Sort em Linguagem C++

Texto

Descrição gerada automaticamente

**Fonte: https://www.thecrazyprogrammer.com/2017/02/bucket-sort-in-c.html. Acesso em: 24/04/2021**

Figura 20 - Código da Imagem acima sendo executado

Texto

Descrição gerada automaticamente

**Fonte: Autoria Própria**

**8. COCKTAIL SORT**

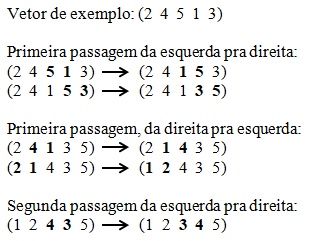
O *Cocktail Sort*, também conhecido como *Bubble Sort* bidirecional ou *Cocktail Shaker Sort*, é uma variação do *bubble sort*, enquanto o *bubble sort* sempre percorre os elementos da esquerda para a direita, e move os maiores valores para o final, o *cocktail sort* atravessa o vetor nas duas direções (da esquerda para a direita, e da direita para a esquerda) alternativamente.

Cada iteração do algoritmo pode ser dividida em duas etapas:

1. A primeira etapa faz o loop pelo vetor da esquerda para a direita, como o *bubble sort*, durante esse loop, itens adjacentes são comparados, e se o valor da esquerda for maior que o valor da direita, então os valores são trocados. No fim da primeira iteração, o maior valor ficará no final da lista.
2. Na segunda etapa, ele faz o loop pelo vetor, na direção contrária (da direita para a esquerda), começando pelo item anterior ao ordenado na primeira etapa, e se movendo de volta para o começo da lista. Nesta etapa, os valores também são comparados e trocados quando necessário, e no final o menor valor ficará no começo da lista.

Essa explicação pode ser mais bem visualizada de acordo com a Figura 21:

Figura 21 - Explicação das Iterações do Cocktail Sort



**Fonte: Autoria Própria**

Em relação ao desempenho do algoritmo, no melhor caso, a conta para calcular essa situação seria: O(n), que ocorreria quando o vetor já estivesse ordenado. E no pior caso, o cálculo seria de O(n²), onde o vetor estaria em ordem decrescente, fazendo com que o algoritmo percorresse o máximo número de vezes.

**8.1. Vantagens e Desvantagens**

**Vantagens**

Muitas vantagens do *cocktail sort* se assemelham às do *bubble sort*, porém umas das principais vantagens seriam:

* É um método de ordenação estável (onde a ordem relativa dos itens iguais não se altera durante a ordenação);
* Tem um desempenho melhor em relação ao *bubble sort*, aproximadamente, o *cocktail sort* consegue ser executado duas vezes mais rápido que o *bubble sort*;

**Desvantagens**

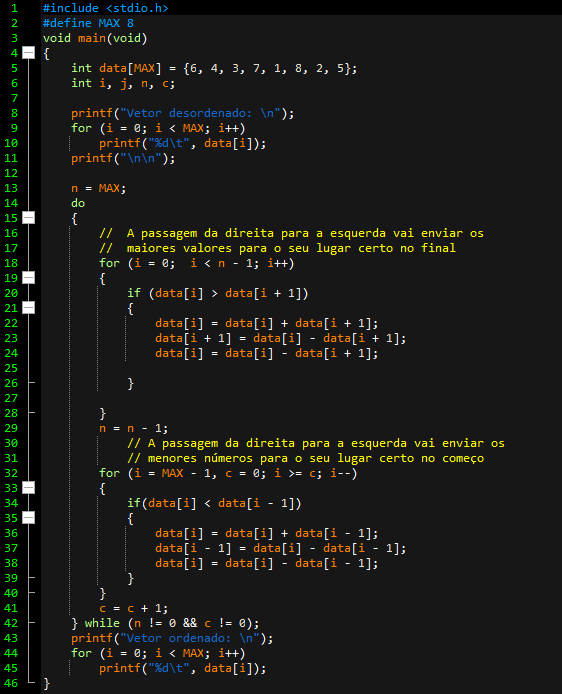
Algumas das desvantagens do *bubble sort* também são vistas presentes no *cocktail sort*, contudo, vale a pena destacar as seguintes:

* O *cocktail sort* é mais difícil de ser implementado no código em relação ao *bubble sort*;
* Em muitos casos, é mais aconselhável se utilizar de outros métodos de ordenação com desempenhos similares ao *cocktail sort*, como *Insertion sort* ou *Selection sort;*

**8.2. Aplicação em Código**

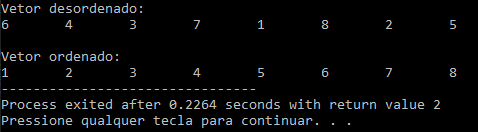
Como dito anteriormente, é mais complicado de aplicar o *cocktail* do que o *bubble sort,* podemos ver um exemplo de código com *cocktail sort* na Figura 22, com sua respectiva tela de resultado (output) na Figura 23:

Figura 22 - Exemplo de Método de Ordenação *Cocktail Sort* em linguagem C



**Fonte: https://www.sanfoundry.com/c-program-implement-cocktail-sort/. Acesso em: 22/04/2021**

Figura 23 - Tela de Execução do código com o método *Cocktail Sort*



**Fonte: Autoria Própria**

**9. RADIX SORT**

O Radix Sort é um método de ordenação utilizado em linguagens de programação, os algoritmos de ordenação são cada vez mais utilizados devido à quantidade de informações que são geradas a cada dia, sendo assim necessário que esta seja recuperada com uma eficiência satisfatória, que forneçam uma velocidade de processamento cada vez maior que os anteriormente propostos, até agora vimos métodos de ordenação que comparam chaves. O Radix Sort vem com uma abordagem alternativa para ordenação, cujo objetivo é processar as chaves por partes, que funciona separando as classes em unidades, dezenas e centenas e assim ordenando o vetor por partes da direita para a esquerda, ou seja, das unidades até as centenas. Por exemplo, 123 tem os dígitos 1, 2 e 3 na base 10, 123 tem os dígitos 1111011 na base 2, programação tem 11 caracteres na base 256.

**Tabela 6 - Vetor desorganizado**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 016 | 361 | 453 | 096 | 213 | 401 |

**Fonte: Autoria Própria**

Este vetor de 6 posições pega os últimos dígitos de cada coluna da direita para a

esquerda e ordenará. Ele vai trocar os valores pelos que tem as menores unidades:

Vetor[0]=6; Vetor[1]=1; Vetor[2]=3; Vetor[3]=6; Vetor[4]=3; Vetor[5]=1.

Existem 2 números 1; 2 números 3; 2 números 6.

**Tabela 7 - Vetor sendo organizado**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2º valor a ser  mudado | 6º valor a ser  mudado | 3º valor a ser  mudado | 5º valor a ser  mudado | 1º valor a ser  mudado | 4º valor a ser  mudado |
| 36**1** | 40**1** | 45**3** | 21**3** | 01**6** | 09**6** |

**Fonte: Autoria Própria**

Agora a função vai fazer a mesma coisa com o conjunto das dezenas:

Vetor[0]=6; Vetor[1]=0; Vetor[2]=5; Vetor[3]=1; Vetor[4]=1; Vetor[5]=9.

Existem 1 número 0; 2 números 1; 1 número 5; 1 número 6; 1 número 9.

**Tabela 8 - Vetor sendo organizado**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 2º valor a ser  mudado | 4º valor a ser  mudado | 5º valor a ser  mudado | 3º valor a ser  mudado | 1º valor a ser  mudado | 6º valor a ser  mudado |
| 4**0**1 | 2**1**3 | 0**1**6 | 4**5**3 | 3**6**1 | 0**9**6 |

**Fonte: Autoria Própria**

E para finalizar faremos de novo com as centenas:

Vetor[0]=4; Vetor[1]=2; Vetor[2]=0; Vetor[3]=4; Vetor[4]=3; Vetor[5]=0.

Existem 2 números 0; 1 número 2; 1 número 3; 2 números 4.

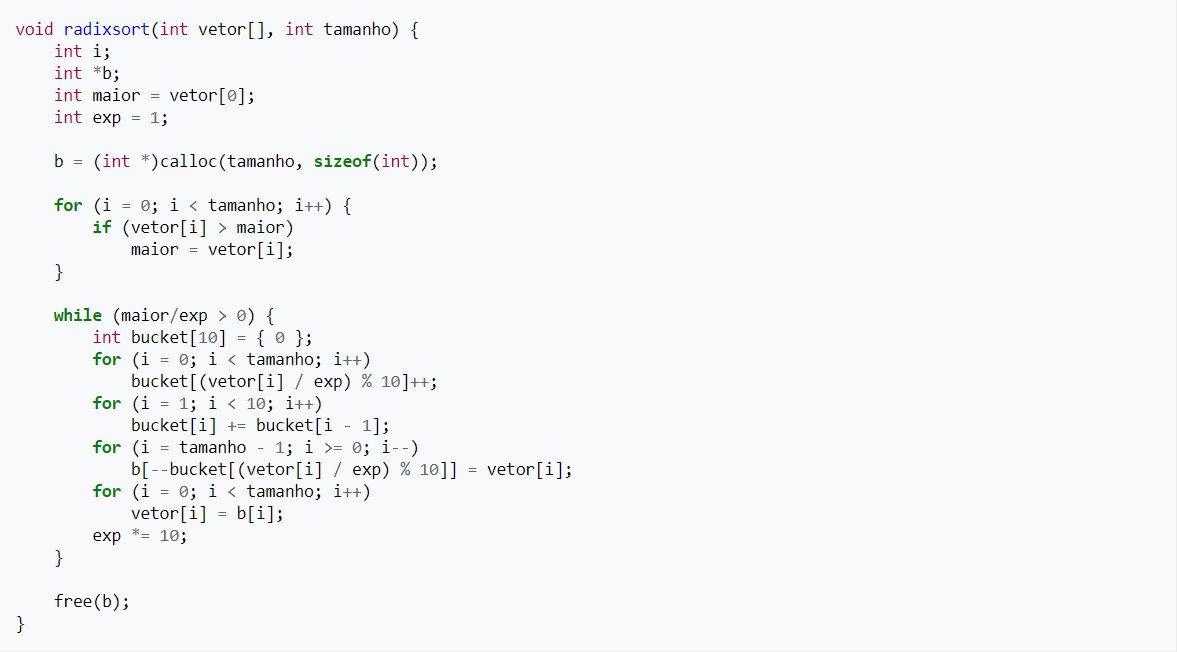
**Tabela 9 - Vetor organizado**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 3º valor a ser  mudado | 6º valor a ser  mudado | 2º valor a ser  mudado | 5º valor a ser  mudado | 1º valor a ser  mudado | 4º valor a ser  mudado |
| **0**16 | **0**96 | **2**13 | **3**61 | **4**01 | **4**53 |

**Fonte: Autoria Própria**

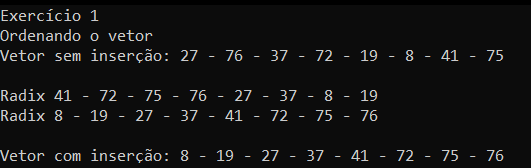
**9.1. Aplicação em Código**

**Figura 24 - Função do Radix Sort em código C**



**Fonte: https://pt.wikipedia.org/wiki/Radix\_sort#C%C3%B3digo\_em\_C. Acesso em: 18/04/ 2021.**

**Figura 25 - Código acima sendo executado**



**Fonte: Autoria Própria**

**9.2. Vantagens e Desvantagens**

Quais as vantagens e desvantagens de usar este método de ordenação?

* Vantagens:
* Estável;
* Não compara as chaves;
* *Radix sort* é bastante eficiente quando o número de registros é grande, mas o tamanho da chave é pequeno;
* Desvantagens:
* Nem sempre é fácil de otimizar a inspeção de dígitos;
* Depende do hardware
* Só é bom se o número de dígitos for pequeno

**10. Conclusão:**

Pode-se concluir que existem vários métodos de aplicação de ordenação, cada qual com a sua determinada função, e podemos afirmar que alguns métodos são melhores que outros, porém dependendo da situação, cada um é mais eficiente que os demais para determinada função, por exemplo: o *HeapSort* em comparação ao *QuickSort*, é mais rápido, e ocupa menos memória, também o *CocktailSort* consegue ser executado duas vezes mais rápido que o *BubbleSort*, tendo estruturas parecidas.

A importância de existirem métodos de ordenação para deixar uma lista em ordem crescente ou decrescente, seria a aplicação de métodos de busca de elementos onde uma lista/array necessitaria estar em um determinado padrão para que a busca fosse realizada de forma rápida e eficiente, pois a maioria dos métodos de busca só funcionam se a lista estiver em uma determinada ordem (crescente ou decrescente), por exemplo: a busca binária exige que o vetor esteja ordenado de forma crescente para que seu algoritmo funcione adequadamente.

Sendo assim concluímos que métodos de ordenação são muito importantes e aplicáveis em diversas áreas no nosso cotidiano, às vezes de forma implícita, como quando o celular ordena nossos contatos de forma alfabética, ou explícita, como numa tabela mostrando os pontos de cada time, a partir de quem fez a maior pontuação.

**REFERÊNCIAS**

Robson. Entendendo o Algoritmo Bubble Sort em Java, 2012. Disponível em: <https://www.devmedia.com.br/entendendo-o-algoritmo-bubble-sort-em-java/24812>. Acesso em 23 de abril de 2021.

BRAGA, Henrique. Algoritmo de Ordenação: Bubble Sort, 18 de fevereiro de 2018. Disponível em: <https://medium.com/@henriquebraga\_18075/algoritmos-de-ordenação-i-bubble-sort-c162a67261ef>. Acesso em 23 de abril de 2021.

GATTO, Eliane Cecília. Algoritmo de Ordenação: Bubble Sort, 16 de agosto de 2017. Disponível em: <https://www.embarcados.com.br/algoritmos-de-ordenacao-bubble-sort/>. Acesso em 23 de abril de 2021.

CHAVES, Thalita. Métodos de ordenação de estrutura de dados, 3 de julho de 2013. Disponível em: <https://pt.slideshare.net/thalitachaves98/metodos-de-ordenao-estrutura-de-dados>. Acesso em 23 de abril de 2021.

DIAS, Welinton Junior. Os dois tipos de armazenamento em memória. Disponível em: <https://meuartigo.brasilescola.uol.com.br/informatica/os-dois-tipos-armazenamento-memoria.htm>. Acesso em 23 de abril de 2021.

O ALGORITMO DE ORDENAÇÃO QUICKSORT. Disponível em:

<https://blog.pantuza.com/artigos/o-algoritmo-de-ordenacao-quicksort>. Acesso em: 22 de abril de 2021.

PROFº TÚLIO TUFFOLO. ORDENAÇÃO QUICKSORT. Disponível em:

<http://www3.decom.ufop.br/toffolo/site\_media/uploads/2013-1/bcc202/slides/15.\_quicksort.pdf>. Acesso em: 22 de abril de 2021.

PROFº ANDRÉ BACKES. ALGORITMO DE ORDENAÇÃO. Disponível em:

<http://www.facom.ufu.br/~backes/gsi011/Aula06-Ordenacao.pdf>.Acesso em: 22 de abril de 2021.

Canal do Código. (3 de Setembro de 2018). *1 Vídeo (12 min). Quicksort simples e sem complicações - Canal do Código*. Disponível em: <https://www.youtube.com/watch?v=PrR3nfq9wSY>. Publicado pelo canal Canal do Código. Acesso em 25 de Abril de 2021.

GeeksforGeeks. Divide and Conquer Algorithm | Introduction (2021). Disponível em: <https://www.geeksforgeeks.org/divide-and-conquer-algorithm-introduction/>. Acesso em: 25 de Abril. de 2021.

GeeksforGeeks. Merge Sort (2021). Disponível em: <https://www.geeksforgeeks.org/merge-sort/>. Acesso em: 25 de Abril de 2021.

BRUNET, João Arthur. Ordenação por Comparação: Merge Sort (2019). Disponível em: <https://joaoarthurbm.github.io/eda/posts/merge-sort/>. Acesso em 25 de Abril de 2021.

FARIAS, Ricardo. Estrutura de Dados e Algoritmos (2014). Disponível em: <https://www.cos.ufrj.br/~rfarias/cos121/aula\_07.html>. Acesso em: 25 de Abril de 2021.

Halisson. (19 de Julho de 2019). *1 Vídeo (23 min). MERGE SORT | Algoritmos #7*., Publicado pelo canal Programação Dinâmica. Disponível em: <https://www.youtube.com/watch?v=5prE6Mz8Vh0>. Acesso em 25 de Abril de 2021.

John von Neumann in Infopédia. Porto: Porto Editora, 2003-2021. Disponível em: <https://www.infopedia.pt/$john-von-neumann>. Acesso em: 25 de Abril de 2021.

TOFFOLO, Túlio. Ordenação: MergeSort (2013). Disponível em: <http://www3.decom.ufop.br/toffolo/site\_media/uploads/2013-1/bcc202/slides/14.\_mergesort.pdf>. Acesso em: 25 de Abril de 2021.

PAULO, João. Estrutura de Dados 1. (2016). Disponível em: <[https://precisoestudarsempre.blogspot.com/2016/08/a-criacao-de-donald-shell-o-](https://precisoestudarsempre.blogspot.com/2016/08/a-criacao-de-donald-shell-o-algoritmo.html) [algoritmo.html](https://precisoestudarsempre.blogspot.com/2016/08/a-criacao-de-donald-shell-o-algoritmo.html)>. Acesso em: 22 de Abril de 2021.

VIANA, Daniel. Estrutura de Dados 1. (2016). Disponível em: <https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/>. Acesso em: 22 de Abril de 2021.

Estrutura de Dados 1. Disponível em:<[https://cadernogeek.wordpress.com/tag/shell-](https://cadernogeek.wordpress.com/tag/shell-sort/) [sort/](https://cadernogeek.wordpress.com/tag/shell-sort/)>. Acesso em: 22 de Abril de 2021.

Estrutura de Dados 1. (2020). Disponível em:<https://pt.wikipedia.org/wiki/Shell\_sort>. Acesso em: 22 de Abril de 2021.

LINDER, Marcelo. aula17.pdf. Univasf, Petrolina. Disponível em:

<http://www.univasf.edu.br/~marcelo.linder/arquivos\_aed2/aulas/aula17.pdf>. Acesso em: 12, abril, 2021.

Professor Robert W. Floyd. Stanford, Califórnia, Setembro, 2001. Disponível em: <https://cs.stanford.edu/memoriam/professor-robert-w-floyd>. Acesso em: 13, abril, 2021.

Heapsort. USP, São Paulo, 25, setembro, 2018. Disponível em:

<https://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html>. Acesso em: 13, abril, 2021.

TOFFOLO, Túlio. BCC202 – Aula 17 Algoritmos e Estruturas de Dados I. UFOP, Ouro Preto, Janeiro, 2014. Disponível em:

<<http://www.decom.ufop.br/anascimento/site_media/uploads/bcc202/aula_16_-_fila_de_prioridade_e_heapsort.pdf>>. Acesso em: 14, abril, 2021.

ORDENAÇÃO EM TEMPO LINEAR BUCKETSORT. SlidePlayer. Disponível em: <https://slideplayer.com.br/slide/9494259/>. Acesso em: 24, abril, 2021.

HEBERT, James. Estrutura de Dados 1. São Paulo. Disponível em:

<https://prezi.com/2\_cf\_jf-pzv\_/bucket-sort/>. Acesso em: 24, abril, 2021.

Radix e Bucket Sort. Novembro, 2010. Disponível em:

<https://cavmelo.files.wordpress.com/2010/11/radixebucketsort.pdf>. Acesso em: 24, abril, 2021.

BACKES, André. Estrutura de Dados 1. 05, setembro, 2019 Disponível em:

<http://www.facom.ufu.br/~backes/gsi011/Aula06-Ordenacao.pdf>. Acesso em: 24, abril, 2021.

First, N. (12 de Dezembro de 2012). *1 Vídeo (4 min). Sorting Algorithms Redux 04: Cocktail Sort*., Publicado pelo canal 0612 TV w/ NERDfirst: Disponível em: <https://www.youtube.com/watch?v=Xmx\_6YRBaq8>. Acesso em 22, de abril de 2021.

COLARES, V. M. M.; OLIVEIRA, A. S.; GONÇALVES, J. J.; CORRÊA, M. M. UMA FERRAMENTA GRÁFICA PARA O ESTUDO E ANÁLISE COMPORTAMENTAL DE ALGORITMOS DE ORDENAÇÃO, (2015). Disponível em: <http://www2.bage.ifsul.edu.br/encif2015/pdf/20150930203219000000.pdf> Acesso em: 22 de abril de 2021.

KONISHI, Mari. Métodos de ordenação (2011). Disponível em: <https://marikonishi.wordpress.com/2011/02/28/metodos-de-ordenacao/>. Acesso em: 22 de abril de 2021.

WANDY, Joe. As vantagens e desvantagens dos algoritmos de ordenação (2017). Disponível em: < https://www.ehow.com.br/vantagens-desvantagens-algoritmos-ordenacao-info\_16277/>. Acesso em: 22 de abril de 2021.

VIANA, Daniel. Conheça os principais algoritmos de ordenação (2016). Disponível em: < https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/>. Acesso em: 22 de abril de 2021.

BHOJASIA, Manish. C Program to Implement CockTail Sort. Disponível em: < https://www.sanfoundry.com/c-program-implement-cocktail-sort/>. Acesso em: 22 de abril de 2021.

AGRAWAL, Rahul. Cocktail Sort (2020). Disponível em: <https://www.geeksforgeeks.org/cocktail-sort/>. Acesso em: 22 de abril de 2021.

ORDENAÇÃO: RADIXSORT: ALGORITMOS E ESTRUTURAS DE DADOS II. Disponível em: <https://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/radixsort.pdf>. Acesso em: 18 de abril de 2021.

ALGORITMO DE ORDENAÇÃO RADIX SORT. Disponível em:

<https://www.trabalhosfeitos.com/ensaios/Algoritmo-De-Ordena%C3%A7%C3%A3o-Radix-Sort/66964975.html>. Acesso em: 18 de abril de 2021.

RADIX SORT. Disponível em: <https://pt.wikipedia.org/wiki/Radix\_sort#C%C3%B3digo\_em\_C>. Acesso em: 18 de abril de 2021.

RADIX E BUCKET SORT. Disponível em: <https://cavmelo.files.wordpress.com/2010/11/radixebucketsort.pdf>. Acesso em: 18 de abril de 2021.