Desenvolvimento de sistema para análise de performance de algoritmos de ordenação de dados

Nome: Rodrigo Joaquim Sousa de Lima

RA: F32IEA0

Turma: CC4A30

Trabalho da APS

Índice

[1. Introdução 3](#_Toc87893986)

[2. Referencial teórico 5](#_Toc87893987)

[2.1. Bubble sort 5](#_Toc87893988)

[2.2. Selection sort 6](#_Toc87893989)

[2.3. Insertion sort 7](#_Toc87893990)

[2.4. Quick sort 8](#_Toc87893991)

[2.5. Merge sort 9](#_Toc87893992)

[2.6. Heap sort 10](#_Toc87893993)

[3. Desenvolvimento 11](#_Toc87893994)

[4. Resultados e discussão 11](#_Toc87893995)

[5. Considerações Finais 11](#_Toc87893996)

[6. Referências bibliográfico: 11](#_Toc87893997)

[7. Código Fonte 11](#_Toc87893998)

[8. Ficha de atividades práticas supervisionadas 11](#_Toc87893999)

# Introdução

Neste projeto abordaremos sobre algoritmos de ordenação, que são os mais próximos da estrutura de uma linguagem estruturada, que servem para organizar uma lista de números ou palavras de acordo com a necessidade do usuário.

Antes de abordar sobre algoritmos de ordenação, primeiro temos que entender o que é um algoritmo e de onde ele surgiu. A palavra algoritmo está relacionada ao nome do matemático Abu Abdullah Muhammad Bin Musa Al-Khwarizmique que viveu entre os séculos VIII e IX. O seu trabalho consistiu em preservar e difundir o conhecimento da Antiga Grécia e da Índia, os seus livros eram de fácil compreensão porque sua filosofia e principal valor não consistiam em criar teoremas e correntes de pensamento, mas de simplificar as matemáticas em um nível de compreensão acessível para todos.

O conceito de algoritmo existe há séculos e seu uso pode ser rastreado até os matemáticos russos, como a peneira de Eratóstenes e o algoritmo euclidiano.   
Os conceitos de algoritmos são frequentemente ilustrados por receitas de amostra, embora muitos algoritmos sejam mais complexos. Eles podem repetir etapas (iteração) ou solicitar decisões (como comparações ou lógica) até que a tarefa seja concluída. Um algoritmo implementado corretamente não resolverá um problema se for mal implementado ou se não corresponder ao problema de Jean Luc Chabert.

Um algoritmo não representa necessariamente um programa de computador, mas sim as etapas necessárias para realizar uma tarefa. Sua implementação pode ser feita por um computador, por outro tipo de autômato ou por um ser humano. Algoritmos diferentes podem realizar a mesma tarefa usando um conjunto diferente de instruções com mais ou menos tempo, espaço ou esforço do que outros algoritmos. Essa diferença pode refletir a complexidade computacional aplicada, que depende da estrutura de dados adequada ao algoritmo. Por exemplo, um algoritmo para se vestir pode especificar que você coloque meias e sapatos antes de colocar as calças, enquanto outro algoritmo especifica que você deve colocar primeiro as calças, depois as meias e os sapatos. É claro que o primeiro algoritmo é mais difícil de implementar do que o segundo, mesmo que ambos conduzam ao mesmo resultado.

Em matemática e ciência da computação, um algoritmo é uma série finita de ações possíveis que encontram uma solução para um determinado tipo de problema. De acordo com Dasgupta, Papadimitriou e Vazirani; "Algoritmos são procedimentos precisos, inequívocos, padronizados, eficientes e precisos."   
Suas características são: finitas, o algoritmo deve possivelmente resolver o problema; bem definidas: as etapas devem ser definidas de forma que possam ser compreendidas;

Para sermos eficazes, devemos sempre resolver os problemas que temos de resolver e antecipar as falhas. Assim, graças à especificação clara e concisa de como calcular sistematicamente, pode-se definir algoritmos em dispositivos mecânicos semelhantes ao ábaco.

Augusta Ada Byron King, Condessa de Lovelace, atualmente conhecida como Ada Lovelace, foi uma matemática e escritora inglesa. Hoje é reconhecida principalmente por ter escrito o primeiro algoritmo para ser processado por uma máquina, a máquina analítica de Charles Babbage, que em seus estudos detalhava o funcionamento da máquina analítica em 1842. Quando publicou seu artigo científico em 1843, ela fez comentários prescientes que incluíram suas previsões sobre uma máquina que poderia ser usada para compor música complexa, produzir gráficos e usada para uso prático e científico. Portanto, foi através do seu trabalho que os historiadores a reconhecem como a primeira programadora.

Um algoritmo é um mecanismo cego e sem vontade, que segue um conjunto de regras aplicadas sistematicamente a alguns dados de entrada apropriados e resolve os seus problemas em um número finito de passos elementares. Assim, um algoritmo é como uma "receita" para executarmos uma tarefa ou resolver algum problema. Se seguirmos uma receita de bolo corretamente, conseguiremos fazer o bolo e como toda receita, um algoritmo deve seguir determinadas instruções para se chegar na resolução do problema.

O desenvolvedor na construção de um algoritmo deve prever todos os tipos de situação que podem ocorrer no meio da execução do algoritmo e saber criar estratégias para fracionar problemas reais em instruções mais abstratas para que o computador consiga seguir na solução de um problema.

As linguagens de programação já possuem métodos de ordenação, mas é bom saber como funcionam os algoritmos, pois há casos de problemas em que o algoritmo de ordenação genérico não resolve, às vezes é necessário modificá-lo. Pode ser implementado com qualquer sequência de valores ou objetos que tenham uma lógica infinita (por exemplo, a língua portuguesa, a linguagem Pascal, a linguagem C, uma sequência numérica, um conjunto de objetos tais como lápis e borracha), ou seja, qualquer coisa que possa fornecer uma sequência lógica.

Os mais populares algoritmos de ordenação são: Insertion sort, Selection sort, Bubble sort, Comb sort, Quick sort, Merge sort, Heap sort e Shell sort. Neste projeto será apresentado os algoritmos como Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, Merge Sort e Heap Sort.

# Referencial teórico

O objetivo da ordenação é facilitar a recuperação dos dados de uma lista, o algoritmo de ordenação coloca os elementos de uma dada sequência em uma certaa ordem e cada tipo de algoritmo de ordenação tem o seu jeito de executar.

## Bubble sort

O Bubble Sort é o algoritmo mais simples, mas o menos eficientes. Neste algoritmo cada elemento da posição i será comparado com o elemento da posição i + 1, ou seja, um elemento da posição 2 será comparado com o elemento da posição 3. Caso o elemento da posição 2 for maior que o da posição 3, eles trocam de lugar e assim sucessivamente. Por causa dessa forma de execução, o vetor terá que ser percorrido quantas vezes que for necessária, tornando o algoritmo ineficiente para listas muito grandes.

Para entender o Bubble Sort, vamos considerar uma matriz não classificada [1, 23, 10, -2] e discutir cada etapa realizada para classificar a matriz em ordem crescente. Em cada passagem, dois elementos adjacentes são verificados e trocados se encontrados na ordem errada.

Primeira passagem: (1) e (23) são comparados e encontrados na ordem correta (ordem crescente neste caso). Depois disso (23) e (10) são comparados, uma vez que (23> 10), portanto, esses números são trocados. Então (23) e (-2) são comparados e trocados.

Segunda passagem: (1) e (10) são comparados e encontrados na ordem correta. Então (10) e (-2) são comparados, uma vez que (10> -2), portanto, esses números são trocados. Depois disso, (10) e (23) são comparados e encontrados na ordem correta.

Terceira passagem: (1) e (-2) são comparados, uma vez que (1> -2), portanto, esses números são trocados. Depois disso, (1,10) e (10,23) são verificados e encontrados na ordem correta.

## Selection sort

O Selection Sort é baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o segundo menor valor para a segunda posição e assim sucessivamente, até os últimos dois elementos.

Neste algoritmo de ordenação é escolhido um número a partir do primeiro, este número escolhido é comparado com os números a partir da sua direita, quando encontrado um número menor, o número escolhido ocupa a posição do menor número encontrado. Este número encontrado será o próximo número escolhido, caso não for encontrado nenhum número menor que este escolhido, ele é colocado na posição do primeiro número escolhido, e o próximo número à sua direita vai ser o escolhido para fazer as comparações. É repetido esse processo até que a lista esteja ordenada.

Para entender a classificação da seleção, vamos considerar uma matriz não classificada [1, 10, 23, -2] e discutir cada etapa realizada para classificar a matriz em ordem crescente. Em cada passagem, o menor elemento é encontrado no array não classificado e trocado pelo primeiro elemento do array não classificado.

Primeira passagem: A matriz inteira é uma matriz não classificada e (-2) é o menor número na matriz. Depois de encontrar (-2) como o menor número, ele é trocado pelo primeiro elemento da matriz.

Segunda passagem: neste exemplo, a matriz do lado esquerdo é a matriz classificada e o comprimento dessa matriz aumenta em um após cada iteração. Após a primeira passagem, o comprimento da matriz classificada é um. O resto da matriz é uma matriz não classificada. (1) é o menor número no array não classificado que é trocado pelo primeiro elemento do array não classificado. Após esta troca, o comprimento do array ordenado e do array não ordenado será dois.

Terceira passagem: (10) é o menor número no array não classificado e trocado pelo primeiro elemento do array não classificado.

Quarta passagem: (23) é o único número na matriz não classificada e está na posição correta

## Insertion sort

O Insertion Sort é baseado na ideia de consumir um elemento da matriz não classificada e inseri-lo na posição correta na matriz classificada. Isso resultará no aumento do comprimento da matriz classificada em um e na diminuição do comprimento da matriz não classificada em um após cada interação.

Para entender o Insertion Sort, vamos considerar um array não ordenado [23, 1, 10, 5, 2] e discutir cada passo dado para ordenar o array em ordem crescente. Em cada passagem, um elemento é retirado da matriz não classificada e inserido na posição correta na matriz classificada.

Primeira passagem: A matriz inteira é uma matriz não classificada e (23) é considerada como inserida na matriz classificada. Como (23) é o primeiro elemento da matriz classificada e não possui elementos para comparação, ele permanece em sua posição.

Segunda passagem: (1) é obtido da matriz não classificada para inserir na matriz classificada. Ele é comparado com todos os elementos do array ordenado e descobre que (23) é o único número maior que (1). (1) é inserido na matriz classificada e a posição de (23) é deslocada em um lugar na matriz. Isso resultou no aumento do comprimento da matriz classificada em um e na diminuição do comprimento da matriz não classificada em um.

Terceira passagem: (10) é obtido do array não classificado e comparado com todos os elementos do array ordenado. (23) é o único número na matriz classificada que é maior que (10). (10) é inserido na matriz classificada e a posição de (23) é deslocada em um lugar na matriz.

Quarta passagem: (5) é comparado com todos os elementos do array ordenado e descobre-se que (10) e (23) são os números que precisam ser deslocados em um lugar para inserir (5) no array ordenado.

Quinta passagem: para inserir (2) na matriz classificada, (5), (10) e (23) são deslocados em um lugar.

## Quick sort

O Quick Sort é o algoritmo mais eficiente na ordenação por comparação. Nele se escolhe um elemento chamado de pivô, a partir disto é organizada a lista para que todos os números anteriores a ele sejam menores que ele, e todos os números posteriores a ele sejam maiores que ele. Ao final desse processo o número pivô já está em sua posição final. Os dois grupos desordenados recursivamente sofreram o mesmo processo até que a lista esteja ordenada.

Para entender o Quick sort, vamos considerar uma matriz não classificada [-4, 1, 25, 50, 8, 10, 23] e discutir cada etapa realizada para classificar a matriz em ordem crescente.

Na implementação deste exemplo, o último elemento da matriz é escolhido como o elemento pivô. No início do processo de partição, as variáveis i e j são criadas, as quais são as mesmas inicialmente. À medida que a partição avança, os valores de i e j tornam-se diferentes. i significa o limite entre os elementos menores que o pivô e os elementos maiores que o pivô. j significa o limite entre os elementos maior do que o pivô e a matriz não particionada.

Após a partição, ele gera duas partições com a partição do lado esquerdo contendo elementos menores que o pivô e a partição do lado direito contém elementos maiores do que o pivô. Ambas as partições são novamente particionadas com a mesma lógica e este processo continua até que uma partição tenha zero ou um elemento. O resultado final desse processo será uma matriz classificada.

## Merge sort

O Merge Sort é um exemplo de algoritmo de ordenação por comparação do tipo dividir-para-conquistar.

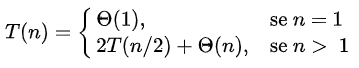
Sua ideia básica consiste em Dividir (o problema em vários subproblemas e resolver esses subproblemas através da recursividade) e Conquistar (após todos os subproblemas terem sido resolvidos ocorre a conquista que é a união das resoluções dos subproblemas). Como o algoritmo Merge Sort usa a recursividade, há um alto consumo de memória e tempo de execução, tornando esta técnica não muito eficiente em alguns problemas.

Os três passos úteis dos algoritmos de divisão e conquista, que se aplicam ao merge sort são:

Dividir: Calcula o ponto médio do sub-arranjo, o que demora um tempo constante 0(1);

Conquistar: Recursivamente resolve dois subproblemas, cada um de tamanho n/2, o que contribui com 2T(n/2) para o tempo de execução;

Combinar: Unir os sub-arranjos em um único conjunto ordenado, que leva o tempo 0(n)



Complexidade

Complexidade de tempo: 0(nlog2n)

Complexidade de espaço: 0(n)

## Heap sort

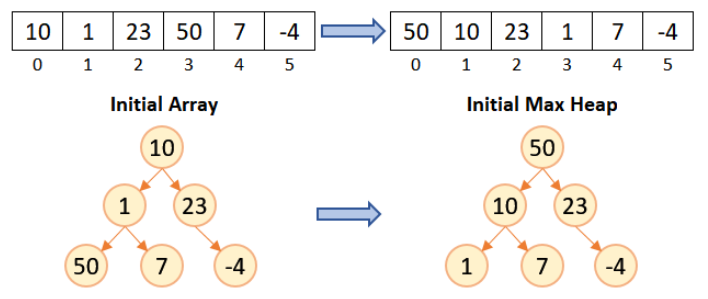
O Heap Sort usa a propriedade heap para classificar a matriz. Envolve a construção de um heap máximo para aumentar a classificação da ordem que contém o maior elemento do heap na raiz. Para ordenar por ordem decrescente, o min heap é usado, o qual contém o menor elemento do heap na raiz. A etapa do processo de classificação de ordem crescente de classificação de heap é resumida abaixo:

Etapa 1: construir um heap máximo que contenha o maior elemento do heap na raiz

Etapa 2: troque o último elemento do heap pelo elemento raiz e remova o último elemento do heap. Com os elementos restantes, repita a etapa 1.

Para entender o heap sort, vamos considerar uma matriz não classificada [10, 1, 23, 50, 7, -4] e discutir cada etapa realizada para classificar a matriz em ordem crescente.

Na figura abaixo, a estrutura de heap da matriz de entrada e heap máximo é mostrada. O número do índice do elemento raiz do heap é 0. No heap máximo, o maior elemento do heap sempre reside na raiz.



Depois de construir o heap máximo inicial, o último elemento do heap é trocado pelo elemento raiz e o último elemento que contém o maior número da matriz é removido do heap. Depois disso, a função heapify é usada nos elementos restantes do heap para torná-lo um heap máximo e o número de elementos será reduzido em um. Esse processo continua até que o número de elementos na pilha seja um. Neste ponto, a matriz estará classificada.

# Desenvolvimento

# Resultados e discussão

# Considerações Finais

# Referências bibliográfico:

<https://pt.wikipedia.org/wiki/Algoritmo>

<https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>

<https://dicasdeprogramacao.com.br/o-que-e-algoritmo/>

<https://brasil.elpais.com/brasil/2018/03/30/tecnologia/1522424604_741609.html>

<https://www.techtudo.com.br/listas/2020/05/o-que-e-algoritmo-entenda-como-funciona-em-apps-e-sites-da-internet.ghtml>

<http://www.testonline.com.br/historia-do-algoritmo-os-primeiros-passos-da-computacao/>

<https://www.alphacodingskills.com/php/pages/php-program-for-bubble-sort.php>

<https://www.alphacodingskills.com/php/pages/php-program-for-selection-sort.php>

<https://www.alphacodingskills.com/php/pages/php-program-for-insertion-sort.php>

<https://www.alphacodingskills.com/php/pages/php-program-for-quick-sort.php>

<https://www.alphacodingskills.com/php/pages/php-program-for-merge-sort.php>

<https://www.alphacodingskills.com/php/pages/php-program-for-heap-sort.php>

<https://pt.wikipedia.org/wiki/Merge_sort>

# Código Fonte

# Ficha de atividades práticas supervisionadas