**Complexidade dos algoritmos e análise dos métodos de ordenação**

Gabriel Tonhatti Cardoso[[1]](#footnote-2)

Resumo

Digitar o resumo do trabalho em único parágrafo. Esse item deve conter entre 100 e 250 palavras, incluindo números, preposições, conjunções e artigos. Não deve conter citações bibliográficas nem abreviaturas. A expressão “Termos para indexação” (ou “Palavras-chave) deve ser seguida de dois pontos (:), deve ser grafada em letras minúsculas (exceto a letra inicial) e em negrito. Os termos devem vir logo à frente da expressão “Palavras-chave” ou “Termos para indexação” e ser separados por ponto e iniciados com letra maiúscula. Devem conter no mínimo três e no máximo seis palavras-chave, em ordem alfabética. Devem iniciar com letra maiúsculas e ser seguidas de ponto.

**Palavras-chave:** Análise de complexidade, métodos de ordenação.

*Abstract*

*Tradução para o inglês do texto contido no “Resumo”. Deve ser redigido em inglês científico, evitando-se sua tradução por meio de aplicativos comerciais. O texto deve ser justificado e digitado em espaço simples, começando por Abstract, em parágrafo único. Deve seguir os mesmos padrões do “Resumo” e ser todo em itálico.*

***Keywords:*** *Digitar. Em ordem alfabética. Palavras-Chave.*

1 Introdução

Será utilizada como diretriz a Associação Brasileira de Normas Técnicas (ABNT), norma NBR 14724:2011.

Os textos devem ser editados no Microsoft Word, formato A4, fonte Arial, tamanho 12. Excetuam-se as citações com mais de três linhas, notas de rodapé, paginação, legendas e fontes das ilustrações e das tabelas, que devem ser em tamanho menor e uniforme.

As páginas devem apresentar margem esquerda e superior de 3 cm e direita e inferior de 2 cm.

Todo o texto deve ser digitado com espaçamento entre linhas de 1,5 e apresentado na forma justificada. São exceções, neste caso, citações de mais de três linhas, notas de rodapé, referências, legendas das ilustrações e das tabelas, que devem ser digitadas em espaço simples.

O título do artigo será em letras maiúsculas e tamanho 14. Deve ter no máximo 15 palavras, incluindo-se os artigos, as preposições e as conjunções.

Os títulos de cada subseção (Resumo, Abstract, Introdução, Material e Métodos, entre outros), têm tamanho de letra 12 e somente a primeira letra em maiúsculo. São grafados em negritos e alinhados à esquerda. Devem ser precedidos de algarismo arábico e separados por um espaço de caractere, sem ponto ou travessão. Não deve ser colocado ponto final após os títulos e subtítulos. Excetuam-se os tópicos Considerações finais, Agradecimentos e Referências, que não possuem numeração e devem ter alinhamento centralizado.

Os títulos das seções devem ser separados do texto que os precede e que os sucede por uma linha em branco.

Optou-se, para a formatação dos trabalhos a serem publicados na revista EduFatec: educação, tecnologia e gestão, que os parágrafos iniciassem com recuo de 1,25cm na primeira linha.

Todas as páginas do artigo devem ser numeradas, a partir do número 1, em algarismos arábicos, no canto superior direito da folha.

As citações devem ser normalizadas de acordo com a NBR 10520 da ABNT e estão exemplificadas nas normas da revista.

As figuras e tabelas também estão exemplificadas nas normas da revista.

**1.1 Algoritmos**

Algoritmos são sequências de passos para se realizar uma tarefa, o exemplo mais famoso é uma receita de bolo, onde se tem o passo a passo de como preparar e fazer um bolo, da mesma forma na ciência da computação tem a sequência de passos que é escrito no código para se definir alguma ação a ser feita por um programa, podendo receber um valor de entrada e retornar um valor de saída.

Exemplos de problemas que envolvem algoritmos no dia a dia:

* Trocar uma lâmpada.
* Preparar um bolo.
* Organizar a lista de contatos por ordem alfabética.
* Calcular a rota mais curta entre duas ruas.
* Calcular a rota mais curta entre duas ruas

De forma geral, algoritmos são usados como ferramentas para decifrar um problema, as suas características são: finitas, bem definidas e efetivas.

As principais técnicas são:

* Algoritmo guloso
* Programação Dinâmica
* Dividir e Conquistar
* Busca e Ordenação
* Backtracking

Também são utilizadas as **Estrutura de Dados** juntamente com essas técnicas para ajudar numa melhor eficácia dos algoritmos. Como por exemplo: Grafos, Árvores, Heaps, Tabelas Hash, Pilhas e Filas.

**1.2 Eficiência dos Algoritmos**

Existe vários métodos para se analisar a performance e eficácia de algum programa, os famosos “Profilers”. Contudo, embora tenha uma boa eficácia, eles não são tão conveniente para analisar a complexidade de algoritmos. A complexidade dos algoritmos avaliam um algoritmo em “nível de idealização/definição”, ou seja, desconsiderando qualquer implementação de linguagens específicas ou hardware.

Comparar o tempo que um algoritmo leva para executar determinada tarefa em milissegundos em relação a outro, não é características para dizer que um algoritmo é melhor do que o outro.

Pode-se dizer que o melhor algoritmo para decifrar um problema é aquele que tem a menor complexidade de tempo e espaço. Ou seja, é aquele algoritmo em que, de acordo com que a entrada cresce visando o infinito, é aquele que mostra o menor tempo de variação e a menor memória utilizada para terminar.

Um algoritmo é capaz de ser melhor do que outro quando processa pouco dados, mas é capaz de ser muito pior de acordo com que os dados crescem. A**Análise de complexidade** nos permite medir o quão rápido um programa executa seus cálculos.

**1.3 Comportamento Assintótico**

Seria um trabalho muito cansativo ficar contando a quantidade de informação para cada trecho de código que fosse escrito. Além disso, a quantidade de informações varia muito de linguagem, compilador e até mesmo o hardware da máquina que está sendo utilizada.

Na **Análise de Complexidade** podemos apenas nos interessar pelo termo que mais cresce de acordo com a entrada. Para atingirmos esse termo, podemos remover todas as constantes e manter apenas o termo que mais cresce.

No método **f(n) = 6n + 4**, evidentemente, 4 continua com o mesmo valor independente da entrada, mas **6n** aumenta cada vez. Retirando ele, encontramo-nos com o método **f(n) = 6n**.

Visto que 6 é uma constante, conseguimos retirá-lo e obtermos o método **f(n) = n**. Dessa forma, facilita demais a análise da complexidade do algoritmo.

*Comportamento assintótico das seguintes funções:*

* *f(n) = 5n + 12 nos retorna f(n) = n  
  Pelo mesmo fato do exemplo anterior.*
* *f(n) = 915 nos retorna f(n) = 1  
  Estamos retirando o multiplicador****915****\* 1*
* *f(n) = n² + 2n + 300 nos retorna f(n) = n²  
  Aqui, o termo n² aumenta mais rápido do que 2n*
* *f(n) = n² + 2000n + 5929 nos retorna f(n) = n²  
  Mesmo que o fator antes de n é bem grande, podemos encontrar um valor para n onde n² se torna maior que 2000n.*

**1.4 Complexidade dos Algoritmos e Notação Big-O**

Para apresentar o comportamento assintótico de algum algoritmo, foi usado a **Notação Big-O** pelo cientistas da computação. Ela é usada para demarcar assintoticamente o aumento(tempo ou espaço) superior do algoritmo.

Usando o algoritmo para encontrar o maior elemento como forma de demonstração, somos capazes de encontrar caso de entrada que fará com que ele execute um número menor de cálculos. Não será para todo caso de entrada que seu método para a quantidade de instruções seja **f(n)**.

Usando a notação Big-O, conseguimos dizer que a complexidade do algoritmo é **“Big-O de O(n)**”, ou seja, no pior caso cresce em ordem de **n.**

Em algoritmos simples, é fácil identificar a complexidade do mesmo. Normalmente, se o algoritmo tem apenas 1 laço de repeitção, sua complexiade é **O(n)**, se possui 2 laços de repetição encadeados **O(n²)** e se não possuir nenhum laço **O(1)**.

**1.5 Complexidade de espaço**

Toda os estudos feitos até o momento foram em função da quantidade de operações que os algoritmos pedem, que é semelhante à **Complexidade de Tempo**.

A complexidade de espaço de um algoritmo não é muito distinto da complexidade de tempo em questão de análise, que também é utilizado a notação **Big-O**.

Para conseguir analisar a complexidade de espaço de algum algoritmo, deve-se identificar a quantidade de memória que o algoritmo precisa alocar para resolver o problema no pior dos casos.

**2 Métodos de Ordenação**

Um método de ordenação é estável se a ordem relativa dos itens iguais não se altera durante a ordenação. O funcionamento do algoritmo é bem simples: consiste em cada passo a partir do segundo elemento selecionar o próximo item da sequência e colocá-lo no local apropriado de acordo com o critério de ordenação.

**2.1 Bubble Sort**

Bubble Sort é um algoritmo de ordenação que pode ser aplicado em Arrays e Listas dinâmicas. Se o objetivo é ordenar os valores em forma decrescente, então, a posição atual é comparada com a próxima posição e, se a posição atual for maior que a posição posterior, é realizada a troca dos valores nessa posição. Caso contrário, não é realizada a troca, apenas passa-se para o próximo par de comparações.  
 Se o objetivo é ordenar os valores em forma crescente, então, a posição atual é comparada com a próxima posição e, se a posição atual for menor que a posição posterior, é realizada a troca. Caso contrário, a troca não é feita e passa-se para o próximo par de comparação.  
 Um array ou lista pode estar já ordenado no momento em que se solicita a ordenação, dessa forma, esta situação tem de ser considerada na implementação do algoritmo.

Segue abaixo o algoritmo de ordenação BUBBLE SORT, para exemplo:

**let pass, comps, trocas;**

**function bubbleSort(vetor) {**

**pass = 0, comps = 0, trocas = 0;**

**let trocou;**

**do {**

**pass++**

**trocou = false;**

**for (let i = 0; i < vetor.length - 1; i++) {**

**comps++;**

**if (vetor[i] > vetor[i + 1]) {**

**[vetor[i], vetor[i + 1]] = [vetor[i + 1], vetor[i]];**

**trocou = true;**

**trocas++;**

**}**

**}**

**} while (trocou)**

**}**

**let nums = [77, 44, 22, 33, 99, 55, 88, 0, 66, 11];**

**bubbleSort(nums);**

**console.log(nums);**

**2.2 Selection Sort**

A ordenação por seleção ou *selection sort* consiste em selecionar o menor item e colocar na primeira posição, selecionar o segundo menor item e colocar na segunda posição, segue estes passos até que reste um único elemento. Para todos os casos (melhor, médio e pior caso) possui complexidade C(n) = O(n²) e não é um algoritmo estável.

O Selection Sort é in-place e O(n2)O(n2), mas não é estável.

A estabilidade é uma propriedade relacionada à ordem relativa de valores iguais no array original. Por exemplo, se houver dois valores 97 no array antes da ordenação, após a execução do algoritmo, esses dois valores devem seguir a ordem relativa inicial. Ou seja, ao término da execução do algoritmo, a primeira ocorrência do 97 deve vir antes da segunda ocorrência do 97.

O Selection Sort não é estável porque dependendo das trocas, ele não mantém a ordem relativa dos valores iguais.

O Selection Sort é *in-place* porque a ordenação é feita rearranjando os elementos no próprio array, ao invés de usar arrays ou outras estruturas auxiliares.

Segue abaixo o algoritmo de ordenação SELECTION SORT, para exemplo:

**letpass,comps,trocas;**

**function *selectionSort*(vetor) {**

**pass = 0, comps = 0, trocas = 0;**

***for* (letposSel=0; posSel *<* vetor*.*length - 1; posSel++) {**

**pass++;**

**letposMenor=posSel+1;**

***for* (leti=posMenor+1; i *<* vetor*.*length; i++) {**

***if* (vetor[posMenor] *>* vetor[i]) {**

**posMenor = i;**

**}**

**comps++;**

**}**

**comps++;**

***if* (vetor[posSel] *>* vetor[posMenor]) {**

**[vetor[posSel], vetor[posMenor]] = [vetor[posMenor], vetor[posSel]];**

**trocas++;**

**}**

**}**

**}**

***let nums = [77, 44, 22, 33, 99, 55, 88, 0, 66, 11];***

***selectionSort*(nums);**

**console*.log*(nums);**

**2.2 Merge Sort**

Criado em 1945 pelo matemático americano *John Von Neumann* o Mergesort é um exemplo de algoritmo de ordenação que faz uso da estratégia “dividir para conquistar” para resolver problemas. É um método estável e possui complexidade C(n) = O(n log n) para todos os casos.

Esse algoritmo divide o problema em pedaços menores, resolve cada pedaço e depois junta (merge) os resultados. O vetor será dividido em duas partes iguais, que serão cada uma divididas em duas partes, e assim até ficar um ou dois elementos cuja ordenação é trivial.

Para juntar as partes ordenadas os dois elementos de cada parte são separados e o menor deles é selecionado e retirado de sua parte. Em seguida os menores entre os restantes são comparados e assim se prossegue até juntar as partes.

Segue abaixo o algoritmo de ordenação MERGE SORT, para exemplo:

**let comps = 0, divisoes = 0, juncoes = 0;**

**function mergeSort(vetor) {**

**if (vetor.length < 2) return vetor;**

**let meio = Math.floor(vetor.length / 2);**

**let vetEsq = vetor.slice(0, meio);**

**let vetDir = vetor.slice(meio);**

**divisoes++;**

**vetEsq = mergeSort(vetEsq);**

**vetDir = mergeSort(vetDir);**

**let posEsq = 0, posDir = 0, vetRes = [];**

**while (posEsq < vetEsq.length && posDir < vetDir.length) {**

**comps++;**

**if (vetEsq[posEsq] < vetDir[posDir]) {**

**vetRes.push(vetEsq[posEsq]);**

**posEsq++;**

**}**

**else {**

**vetRes.push(vetDir[posDir]);**

**posDir++;**

**}**

**}**

**let sobra;**

**if (posEsq < posDir) {**

**sobra = vetEsq.slice(posEsq);**

**}**

**else {**

**sobra = vetDir.slice(posDir);**

**}**

**juncoes++;**

**return [...vetRes, ...sobra];**

**}**

**let nums = [77, 44, 22, 33, 99, 55, 88, 0, 66, 11];**

**let numsOrd = mergeSort(nums);**

**console.log({ numsOrd });**

**2.2 Quick Sort**

O Algoritmo Quicksort, criado por *C. A. R. Hoare* em 1960, é o método de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.

Provavelmente é o mais utilizado. Possui complexidade C(n) = O(n²) no pior caso e C(n) = O(n log n) no melhor e médio caso e não é um algoritmo estável.

É um algoritmo de comparação que emprega a estratégia de *“divisão e conquista”*. A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores. Os problemas menores são ordenados independentemente e os resultados são combinados para produzir a solução final.

Basicamente a operação do algoritmo pode ser resumida na seguinte estratégia: divide sua lista de entrada em duas sub-listas a partir de um pivô, para em seguida realizar o mesmo procedimento nas duas listas menores até uma lista unitária.

Funcionamento do algoritmo:

* Escolhe um elemento da lista chamado pivô.
* Reorganiza a lista de forma que os elementos menores que o pivô fiquem de um lado, e os maiores fiquem de outro. Esta operação é chamada de “particionamento”.
* Recursivamente ordena a sub-lista abaixo e acima do pivô.

Segue abaixo o algoritmo de ordenação QUICK SORT, para exemplo:

**let pass = 0, comps = 0, trocas = 0;**

**function quickSort(vetor, ini = 0, fim = vetor.length - 1) {**

**if (fim <= ini) {**

**return; }**

**pass++;**

**const pivot = fim;**

**let div = ini - 1;**

**for (let i = ini; i < fim; i++) {**

**comps++;**

**if (vetor[pivot] > vetor[i] && i !== div) {**

**div++;**

**if (div !== 1) {**

**[vetor[i], vetor[div]] = [vetor[div], vetor[i]];**

**trocas++;**

**}**

**}**

**}**

**div++;**

**comps++;**

**if (vetor[div] > vetor[pivot] && div !== pivot) {**

**[vetor[div], vetor[pivot]] = [vetor[pivot], vetor[div]];**

**trocas++;**

**}**

**quickSort(vetor, ini, div - 1);**

**quickSort(vetor, div + 1, fim);**

**}**

**let nums = [77, 44, 22, 33, 99, 55, 88, 0, 66, 11];**

**quickSort(nums);**

**console.log(nums);**

**3 Materiais e métodos ou desenvolvimento**

Digite os materiais e métodos ou desenvolvimento.

**4 Resultados e discussão**

Apresente os resultados encontrados.

Considerações finais

Relembrar quais foram objetivos iniciais, o que foi de fato desenvolvido, quais foram os principais desafios e quais serão os projetos futuros que poderão ser realizados.

**Referências**

VIANA, Daniel. **Algoritmos de ordenação**. 2017. 7 f. TCC (Graduação) - Curso de Analise e Desenvolvimento de Sistemas, Fatec Franca - Faculdade de Tecnologia de Franca Dr Thomaz Novelino, São Paulo, 2021. Disponível em: https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao. Acesso em: 18 nov. 2021.

GATTO, Elaine Cecília. **Algoritmos de Ordenação: Bubble Sort**. 2017. 10 f. Dissertação (Mestrado) - Curso de Analise e Desenvolvimento de Sistemas, Fatec Franca - Faculdade de Tecnologia de Franca Dr Thomaz Novelino, Franca, 2021. Disponível em: https://www.embarcados.com.br/algoritmos-de-ordenacao-bubble-sort/. Acesso em: 21 nov. 2021.

PEREIRA, Wilder. **Introdução à Complexidade de Algoritmos**. 2019. 3 f. Dissertação (Mestrado) - Curso de Analise e Desenvolvimento de Sistemas, Fatec Franca - Faculdade de Tecnologia de Franca Dr Thomaz Novelino, Franca, 2021. Cap. 1. Disponível em: https://medium.com/nagoya-foundation/introdu%C3%A7%C3%A3o-%C3%A0-complexidade-de-algoritmos-4a9c237e4ecc. Acesso em: 24 nov. 2021.

BRUNET, João Arthur. **Ordenação por Comparação: Selection Sort**. 2019. 5 f. Dissertação (Mestrado) - Curso de Analise e Desenvolvimento de Sistemas, Fatec Franca - Faculdade de Tecnologia de Franca Dr Thomaz Novelino, Franca, 2021. Cap. 1. Disponível em: https://joaoarthurbm.github.io/eda/posts/selection-sort/. Acesso em: 24 nov. 2021.

1. Graduando em [...] pela Fatec Dr Thomaz Novelino – Franca/SP. Endereço eletrônico: [...]. [↑](#footnote-ref-2)