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

*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**

Em matemática e ciência da computação, um algoritmo é uma sequência finita de ações executáveis que visam obter uma solução para um determinado tipo de problema. Segundo Dasgupta, Papadimitriou e Vazirani; "Algoritmos são procedimentos precisos, não ambíguos, padronizados, eficientes e corretos.".

**1.2 Complexidade dos Algoritmos**

A complexidade de um algoritmo tem a ver com quanto tempo e memória esse algoritmo gasta de acordo com o tamanho de sua entrada. Por exemplo, queremos responder a perguntas como "se meu algoritmo gasta 1 minuto para processar uma entrada de 1000 bytes, quantos minutos ele gastará para processar uma entrada de 2000 bytes?"

Uma maneira muito direta de calcular a complexidade seria encontrando alguma fórmula que dê o número exato de operações feitas pelo algoritmo para chegar no resultado, em função do tamanho da entrada. Por exemplo, no algoritmo

**for(i=0; i<N; i++){**

**print(i);**

**}**

poderíamos dizer que o tempo gasto é

T(N) =

N\*(tempo gasto por uma comparação entre i e N) +

N\*(tempo gasto para incrementar i) +

N\*(tempo gasto por um print)

No entanto, dá muito trabalho fazer uma conta super precisa dessas e geralmente nem vale a pena. Por exemplo, suponha que tenhamos nos esforçado bastante e descoberto que um certo algoritmo gasta tempo

T(N) = 10\*N² + 137\*N + 15

Nesse caso o termo quadrático 10\*N² é mais importante que os outros pois para praticamente qualquer valor de N ele irá dominar o total da soma. A partir de N ≥ 14 o termo quadrático já é responsável pela maioria do tempo de execução e para N > 1000 ele já é responsável por mais de 99%. Para fins de estimativa poderíamos simplificar a fórmula para T(N) = 10\*N² sem perder muita coisa.

Outro ponto em que podemos simplificar a nossa fórmula é o fator constante multiplicando o N². Para prever o quão rápido o tempo de execução cresce dependendo da entrada não importa se T(N) = 10\*N ou T(N) = 1000\*N; em ambos os casos dobrar o N vai quadruplicar o tempo de execução.

Por isso tudo, a forma mais popular de se trabalhar com complexidade de tempo e espaço na análise de algoritmos é a *complexidade assintótica*, que ignora esses fatores constantes e os termos que crescem mais devagar. É agora que entra a história de O-grande, Θ-grande e Ω-grande: essas são notações que usamos para caracterizar uma complexidade assintótica.

Vamos começar pelo O-grande, que é uma maneira de dar um limite superior para o tempo gasto por um algoritmo. O(g) descreve a classe de funções que crescem no máximo tão rápido quanto a função g e quando falamos que f ∈ O(g) queremos dizer que g cresce pelo menos tão rápido quanto f. Formalmente:

Dadas duas funções f e g, dizemos que f ∈ O(g) se existem constantes x0 e c tal que para todo x > x0 vale f(x) < c\*g(x)

Nessa definição, a constante c nos dá margem para ignorar fatores constantes (o que nos permite dizer que 10\*N é O(N)) e a constante x0 diz que só nos importamos para o comportamento de f e g quando o N for grande e os termos que crescem mais rápido dominem o valor total.

Para um exemplo concreto, considere aquela função de tempo f(n) = 10\*N^2 + 137\*N + 15 de antes. Podemos dizer que o crescimento dela é quadrático:

Podemos dizer que f ∈ O(N²), já que para c = 11 e N > 137 vale

10\*N² + 137\*N + 15 < c \* N^2

Podemos escolher outros valores para c e x0, como por exemplo c = 1000 e N > 1000 (que deixam a conta bem óbvia). O valor exato desses pares não importa, o que importa é poder se convencer de que pelo menos um deles exista.

Na direção oposta do O-grande temos o Ω-grande, que é para limites inferiores. Quando falamos que f é Ω(g), estamos dizendo que f cresce pelo menos tão rápido quanto g. No nosso exemplo recorrente, também podemos dizer que f(n) cresce tão rápido quanto uma função quadrática:

Podemos dizer que f é Ω(N²), já que para c = 1 e N > 0 vale

10\*N² + 137\*N + 15 > c\*N^2

Finalmente, o Θ-grande tem a ver com aproximações justas, quando o f e o g crescem no mesmo ritmo (exceto por um possível fator constante). A diferença do Θ-grande para o O-grande e o Ω-grande é que estes admitem aproximações folgadas, (por exemplo, N² ∈ O(N³)) em que uma das funções cresce muito mais rápida que a outra.

Dentre essas três notações, a mais comum de se ver é a do O-grande. Normalmente as análises de complexidade se preocupam apenas com o tempo de execução no pior caso então o limite superior dado pelo O-grande é suficiente.

**1.3 Notação Big-O**

A notação big-O é usada Usamos a notação big-O para delimitar asintóticamente o crescimento do tempo de execução dentro dos fatores constantes superiores e inferiores. Algumas vezes queremos saber apenas o limite superior. Por exemplo, embora o tempo de execução no pior caso da busca binária é *Θ(lgn)*, estaria incorreto dizer que a busca binária executa em um tempo *Θ(lgn)* em *todos* os casos. E se encontrarmos o valor alvo na primeira tentativa? Então ele executa em um tempo *Θ(1)*. O tempo de execução da busca binária nunca é pior do que Θ(lgn), mas algumas vezes pode ser melhor. Seria conveniente ter uma forma de notação assintótica que diga "o tempo de execução cresce no máximo assim, mas poderia crescer mais devagar". A notação "O-grande" serve para tais ocasiões.

Se um tempo de execução é *O(f(n))*, então para um *n* suficientemente grande , o tempo de execução é no máximo *k⋅f(n)* para alguma constante *k*.

Dizemos que o tempo de execução é "big-O de *f(n)*” ou apenas "O de *f(n)*”*.* Usamos a notação big-O para **limites assintóticos superiores**, uma vez que ela limita o crescimento do tempo de execução superior para valores suficientemente grandes de entrada.

**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.  
 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**

Devem ser normalizadas de acordo com a NBR 6023:2002 da ABNT e apresentadas em sequência padronizada. São alinhadas à margem esquerda do texto, com espaçamento simples entre as linhas e separadas entre si por uma linha em branco. Abaixo estão destacados alguns exemplos. Demais exemplos disponíveis no manual do TG.

**Artigo de periódico**

AUTOR(es). Título do artigo. **Título do periódico**, local de publicação, v., n., p., ano.

**Artigo de periódico em meio eletronico**

AUTOR(es). Título do artigo. **Título do Periódico**, cidade, v., n., p., ano. Disponível em:<endereço eletrônico>. Acesso em: dia.mês.(abreviado).Ano.

AUTOR(es). Título do artigo. **Título do Periódico**, local de publicação, v., n. p., ano. CD-ROM.

**Livro**

AUTOR(es). **Título**: subtítulo. edição (abreviada). Local: Editora, ano. p. (total ou parcial).

**Capítulo de livro**

AUTOR. Título do capítulo. In: AUTOR do livro. **Título**: subtítulo. Edição (abreviada). Local: Editora, ano. páginas do capítulo.

**Livro em meio eletrônico**

AUTOR(es). **Título**. Edição (abreviada). Local: Editora, ano. p. (total ou parcial). Disponível em<endereço eletrônico>. Acesso em: dia.mês(abreviado).Ano.

AUTOR (es). **Título**. Edição (abreviada). Local: Editora, ano. p. CD-ROM.

**Dissertação, teses e trabalhos de graduação**

AUTOR. **Título**. ano. Número de folhas ou volumes. Categoria da Tese (Grau e área de concentração) - Nome da faculdade, Universidade, ano.

CODEPROJETS, **Visual representation of SQL joins,** 10/01/2015. Disponível em: <http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins>. Acesso em: 05.out.2015.

DATE, C J. **Introdução a sistemas de banco de dados**. 8 ed. Rio de Janeiro: Elsevier, 2003.

ELMASRI, Ramez; NAVATHE, Shamkant B. **Sistema de banco de dados**. 4 ed. São Paulo: Pearson Addison Wesley, 2005.

IBICT. INSTITUTO BRASILEIRO DE INFORMAÇÃO EM CIÊNCIA E TECNOLOGIA**. Bibliografia Brasileira de Ciência da Informação:** 2004/2006. Brasília: IBICT, 2007. 64pp.

**Uso De Siglas Em Referências:**

Apresentar primeiro a sigla, depois o nome completo.

Não usar transcrição de e-books no trabalho, só deverá ser feito como citação indireta.

**Referências da internet:**

chave, título, data ou *sd* quando nao tiver data. Disponível em <link>. Acesso em 12.jan.12.

No texto, quando for feita a citação da internet deverá constar:

chave, data ou *sd, online*.

**Referência:**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.

**Referência:**VERAS, Levi. **Notação Big-O**. 2018. 2 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://pt.khanacademy.org/computing/computer-science/algorithms/asymptotic-notation/a/big-o-notation. Acesso em: 21 nov. 2021.

**Referência:**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.

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