FACULDADE DE TECNOLOGIA DE PRAIA GRANDE:

DESENVOLVIMENTO DE SOFTWARE MULTIPLATAFORMA – 2º CICLO

ESTRUTURA DE DADOS: QUICK SORT

Eduardo Henrique Silva de Amorim

Luiz Felipe Fernandes de Mello

Maria Eduarda Ferreira

Pedro Henrique Bomfim Wolski

Pedro Ricardo da Silva Tavares

Thiago Pereira de Jesus Souza

PRAIA GRANDE - SP

SETEMBRO / 2023

SUMÁRIO

1. ORIGEM DOS ALGORITMOS

Os algoritmos podem ser definidos como uma sequência de instruções ordenadas, cujo objetivo é resolver uma classe específica de problemas e, além disso, também sabemos que essa sequência está, fortemente, presente em nosso dia a dia. Entretanto, poucas pessoas conhecem sua verdadeira origem, essa história iniciou-se no século 9, por meio das contribuições de um matemático chamado Al-Khwarizmi.

O escritor persa foi o responsável por revolucionar as ciências ocidentais através da prática de resolver problemas complexos em partes menores, os tornando mais simples e gerando, assim, uma sequência lógica de etapas. Ademais, os livros do erudito, ao serem traduzidos ao Latim, levavam seu nome na capa, porém escrito como "algoritmi", estabelecendo a criação do substantivo algoritmo, que, no decorrer da história, devido às contribuições de Alan Turing, tivera seu significado moldado às definições que conhecemos hoje.

1.1. ORIGEM DO QUICKSORT

O QuickSort foi desenvolvido em 1960 pelo cientista da computação britânico Charles Antony Richard Hoare, também conhecido como Tony Hoare ou C. A. R. Hoare, que também desenvolveu a Lógica de Hoare e a linguagem formal CSP, usada para especificar interações entre processos concorrentes e que serviu de inspiração para a linguagem de programação Occam.

No período em que desenvolveu o algoritmo, Hoare estava participando em um projeto de tradução de máquina para o National Physical Laboratory. Ele criou o QuickSort ao tentar traduzir um dicionário de inglês para russo, ordenando as palavras, tendo como objetivo reduzir o problema original em subproblemas que possam ser resolvidos de forma mais fácil e rápida. Após uma série de refinamentos o algoritmo foi publicado em 1962.

2. PROCESSO E FUNCIONAMENTO

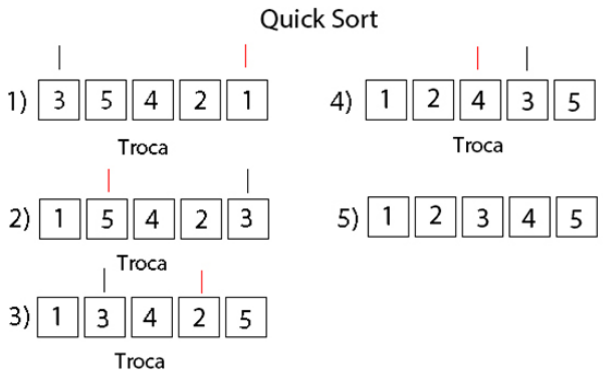
Trazendo esta ideia à área da programação, constata-se que o Quick Sort pode ser definido como um método recursivo que objetiva a ordenação dos elementos de um vetor, ou seja, uma função que exige sua própria execução para garantir a distribuição em ordem crescente dos dados.

A ideia básica do *Quick Sort* é encontrar, a cada passo, a posição de um determinado elemento dentro de um vetor, esse elemento é geralmente intitulado pivô. E a partir da definição dele, o *Quick Sort* busca garantir que todos os elementos a esquerda do pivô sejam menores que ele e os da direita maiores. Basicamente, essa técnica de particionamento ocorre por meio da troca de posições entre as duas variáveis de começo e fim, que serão as responsáveis por analisar todas as informações armazenadas no *array*.

Contudo, por mais que essa divisão seja feita, isso não nos certifica de que todos os dados estão ordenados, o que faz com que a recursividade seja colocada em prática neste método. Portanto, levando em conta o sucesso do particionamento, criam-se duas sub-listas, onde o *Quick Sort* executa-se várias vezes, até que o *array* esteja completamente ordenado.

Como pode-se perceber, o *Quick Sort* é o algoritmo mais eficiente para classificação comparativa. Abaixo, encontra-se um exemplo do funcionamento desse método e a descrição de cada uma das etapas que compõem o seu processo.

Figura 1 - *QuickSort*



Fonte: Stackoverflow.com

1. O número 3 é escolhido como eixo, nesta etapa um número menor que o número é pesquisado à direita para ser movido para a esquerda. O primeiro número menor encontrado foi 1, então eles trocaram de lugar.

2. Agora encontramos um número à esquerda que é maior que esse número. O primeiro número maior encontrado é 5, então eles trocam de lugar.

3. Acontece o mesmo processo do passo 1, o número 2 é o menor número encontrado, eles trocam de lugar.

4. O processo é semelhante ao passo 2, o número 4 é o maior número encontrado, eles trocam de lugar.

5. O vetor deste exemplo é um vetor pequeno, portanto já está ordenado, mas se fosse um vetor grande seria dividido e seguiria o mesmo processo de seleção do eixo de rotação e comparação.

3. MELHOR CENÁRIO

O melhor cenário de aplicação ao Quick Sort ocorre quando o número escolhido para ser o pivô se encontra no meio do vetor ou próximo a ele, originando, assim, a divisão de sub-vetores ideal: duas sub-listas proporcionais, facilitando a finalização do código e, automaticamente, aumentando a velocidade de ordenação do vetor. Ademais, por mais que o pivô não esteja exatamente centralizado, o tempo de execução do Quick Sort mantêm-se bastante efetivo se comparado aos demais métodos de ordenação.

3.1. PIOR CENÁRIO

Já o pior cenário acontece quando o pivô escolhido é na extrema direita ou na extrema esquerda, fazendo com que o Quick Sort realize um processo de particionamento desproporcional, logo, o resultado da partição será um sub-vetor vazio (0) e outro com todos os elementos do array (n – 1). Portanto, ao considerarmos que a cada iteração, o vetor será dividido de maneira desequilibrada, sabe-se que a execução do código perderá eficiência e velocidade.

Então, pensando em prevenções para esse tipo de situação, busca-se sempre planejar a seleção do pivô, seja através da aleatoriedade ou, também, por meio do cálculo da mediana do vetor.

4. APLICABILIDADE

Hoje, o QuickSort é um algoritmo de ordenação amplamente utilizado na área da ciência da computação, conhecido e preferido, principalmente, por sua eficiência acerca de ordenar grandes conjuntos de dados em aplicações gerais, independentemente de onde essas informações estiverem armazenadas, seja em listas, arrays ou até mesmo em arquivos.

Algumas das áreas onde o QuickSort é comumente utilizado são:

- **Sistemas de gerenciamento de banco de dados:** O DBMS faz uso de algoritmos de ordenação para ajeitar os resultados de consultas, auxiliar algoritmos de junção (join) e, além disso, organizar e classificar os dados armazenados em uma tabela de banco de dados. Tendo em vista a importância da ordenação dos dados nesse contexto, o QuickSort se tornou uma escolha muito eficaz na computação comercial e no ramo de engenharia.
- **Algoritmos de busca e pesquisa:** O QuickSort é frequentemente usado, também, como parte de algoritmos de busca e pesquisa, onde é necessário ordenar as informações antes de realizar operações de busca eficientes, neste caso, normalmente, está associado a busca binária.
- **Computação paralela:** O QuickSort também pode ser adaptado à execução paralela, o que o torna útil em ambientes onde várias tarefas são efetuadas simultaneamente em diferentes processadores.
- **Implementações de linguagens de programação:** Muitos códigos utilizam esta técnica para operações de ordenação, seja para ordenar arrays, matrizes ou listas.
- **Algoritmos de otimização:** Os otimizadores buscam encontrar a melhor saída em um espaço de soluções vasto, por isso, muitas vezes usam operações de ordenação e podem se beneficiar do desempenho do QuickSort.

Desse modo, é possível afirmar que o QuickSort é apreciado por sua eficiência média em relação a muitos outros algoritmos de ordenação, seu desempenho em situações reais e, além disso, pela facilidade de implementação, o que acabou tornando-o uma escolha popular em uma variedade de aplicações de diferentes áreas.

5. CÓDIGO

```
#include <iostream>

using namespace std;

int escolherPivot(int vetor[], int inicio, int fim, int valorPivot) {
    for (int i = inicio;i <= fim;i++) {
        if (vetor[i] == valorPivot) {
            swap(vetor[i], vetor[fim]);
            break;
        }
    }
    return fim;
}

void quicksort(int vetor[], int inicio, int fim) {
    if (inicio < fim) {
        int valorPivot = vetor[fim];
        int indiceVetor = escolherPivot(vetor, inicio, fim, valorPivot);

        int i = inicio;
        for (int j = inicio;j < fim;j++) {
            if (vetor[j] < valorPivot) {
                swap(vetor[i], vetor[j]);
                i++;
            }
        }
        swap(vetor[i], vetor[indiceVetor]);

        quicksort(vetor, inicio, i - 1);
        quicksort(vetor, i + 1, fim);
    }
}

int main()
{
    int tamanho = 0;

    cout << "Digite o tamanho do vetor: ";
    cin >> tamanho;

    int* vetor = new int[tamanho];
    cout << "Digite os elementos do array: ";
    for (int i = 0;i < tamanho;i++) {
        cin >> vetor[i];
    }

    cout << "\nVetor criado: ";
    for (int i = 0;i < tamanho;i++) {
        cout << vetor[i] << " ";
    }

    int valorPivot;
    cout << "\nDigite o valor do pivot: ";
    cin >> valorPivot;

    quicksort(vetor, 0, tamanho - 1);
```

```
cout << "\nVetor ordenado: ";
for (int i = 0;i < tamanho;i++) {
    cout << vetor[i] << " ";
}

return 0;
      }
```

5.1. DETALHAMENTO DA ESTRUTURA CÓDIGO

O presente capítulo está destinado ao detalhamento da estrutura do Algoritmo Quicksort, cujo código, em linguagem C++, fora confeccionado pelos próprios integrantes do grupo. Vale ressaltar que as únicas estruturas de condição e repetição que foram utilizadas no código foram: loop FOR e IF.

- **Método escolherPivot:**

O método escolherPivot será a responsável por garantir que o pivot selecionado pelo usuário torne-se o último elemento do vetor, para isto, o método passa a executar uma estrutura de repetição que analisará todos os elementos do nosso vetor. Dentro dessa estrutura de repetição, a condição (vetor[i] == valorPivot) será imposta, quando esta condição for alcançada, realiza-se uma troca entre o pivot selecionado e o último elemento do array, por meio da função swap(vetor[i], vetor[fim]), a responsável por fazer a troca de elementos entre duas variáveis.

Então, como não é necessário continuar verificando o resto dos dados, pois o pivot já foi definido, utiliza-se o comando break para interromper a estrutura de repetição e, por fim, o comando return retorna um valor inteiro armazenado na variável fim.

- **Método Quicksort:**

Nesta função serão alocados os parâmetros que demarcam o início e o fim do vetor, responsáveis por garantir a execução do particionamento do array até que, eventualmente, o início e fim tenham o mesmo valor. Além disso, a função atribuirá o valor do pivô ao último índice do vetor e, em seguida, exigirá a execução de outro método, o “escolherPivot”.

Posteriormente à execução do método “escolherPivot”, temos todos os dados necessários para executar a nossa técnica de ordenação. A partir de agora, o método utiliza outra estrutura de repetição para garantir o particionamento do array,

através da análise dos dados armazenados no vetor um a um, efetivada pela condição (vetor[j] < valorPivot) e, também, pelo comando swap. Ao fim do processo citado acima, será feita uma troca entre vetor[i] e vetor[indiceVetor] para garantir que o pivô seja colocado em sua posição final.

Por fim, o método quicksort chamará a si mesmo mais 2 vezes, comprovando sua recursividade. Uma das chamadas recursivas será feita a partir dos parâmetros: (inicio(0) até o i – 1) e outra a partir de: (i + 1 até o fim(tamanho – 1)), até que não haja mais a necessidade de execução.

- **Código Principal (main):**

Neste método iremos pedir para o usuário criar um vetor, definindo o seu tamanho e seus respectivos elementos. Após as devidas atribuições, mostraremos ao usuário o vetor criado alinhadamente, para facilitar a definição do pivot (lembrando: o pivô pode, ou não, estar alocado no array, porém, se não estiver, acarretará na perca de desempenho do Quick Sort) e, então, após a inserção, a função quicksort será consultada e o vetor ordenado exibido ao usuário, o que demarca o fim do código.

6. CONCLUSÃO

Conclui-se, portanto, que, por mais que o *Quick Sort* seja considerado um algoritmo instável, ele se mantém eficiente, principalmente, por possuir desempenho superior aos métodos simples tanto na situação de pior caso, como também nas situações de melhor e médio caso.

Foi constatado que o Quick Sort é capaz de reduzir consideravelmente seu tempo de execução e sua efetividade advém, principalmente, do uso de técnicas, como, por exemplo, os particionamentos de Lomuto e Hoare, a recursividade e o uso de constantes menores se comparadas às dos demais métodos, características como estas possibilitam a redução do vetor a cada iteração e diminuem o número de swaps em velocidade logarítmica, tornando o Quick Sort, pelo menos, três vezes mais rápido que as demais metodologias de ordenação em sua situação ideal, e é, justamente, por isso, que esse algoritmo de ordenação continua sendo fielmente empregado em diversas áreas.

REFERÊNCIAS BIBLIOGRÁFICAS

ALEXANDRE, Vanilo. **Lógica do algoritmo de ordenação quick sort**. Youtube, 2018. Disponível em: https://www.youtube.com/watch?v=WP7KDljG6IM. Acessado em: 21 de setembro de 2023.

APLICAÇÃO E USOS DO QUICKSORT. Acervolima, 2022. Disponível em: https://acervolima.com/aplicacao-e-usos-do-quicksort/. Acessado em: 21 de setembro de 2023.

BATALHA, Marco. **A Origem Dos Algoritmos**. Youtube, 2022. Disponível em: https://www.youtube.com/watch?v=6bOQQNRr2wI. Acessado em: 21 de setembro de 2023.

BRUNET, João Arthur. **Estrutura de Dados e Algoritmos: Ordenação por Comparação: Quick Sort**. Disponível em: https://joaoarthurbm.github.io/eda/posts/quick-sort/. Acessado em: 19 de setembro de 2023.

Bruno. **Algoritmos de ordenação: análise e comparação**. Disponível em: https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261#:~:text=Quick%20sort,-O%20Quicksort%20%C3%A9&text=Nele%20se%20escolhe%20um%20elemento,est%C3%A1%20em%20sua%20posi%C3%A7%C3%A3o%20final. Acessado em: 20 de setembro de 2023.

CIÊNCIA DA COMPUTAÇÃO: VISÃO GERAL DO QUICKSORT. Khanacademy, 2022. Disponível em: https://pt.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort. Acessado em: 21 de setembro de 2023.

CORMEN, Thomas. **Algoritmos – Teoria e Prática.** GEN LTC, 2012. (V.3).

FEOFILOFF, Paulo. **Ordenação: Quicksort**. Disponível em: https://www.ime.usp.br/~pf/analise_de_algoritmos/aulas/quick.html. Acessado em: 20 de setembro de 2023.

GUIMARÃES, Gleyser. **O Algoritmo QuickSort**. Disponível em: http://www.dsc.ufcg.edu.br/~pet/jornal/abril2013/materias/historia_da_computacao.html#:~:text=O%20QuickSort%20foi%20desenvolvido%20em,como%20Tony%20Hoare%20ou%20C.%20A.%20R. Acessado em: 20 de setembro de 2023.

ISIDRO, Professor Isidro. **Quicksort – o guia definitivo (espero eu) :)**. Youtube, 2018. Disponível em: https://www.youtube.com/watch?v=oGx4cXTVvQU. Acessado em: 21 de setembro de 2023.

MACCAFERRI, Leniel. **Quicksort and Binary Search algorithms in C++**. Disponível em: https://www.leniel.net/2008/05/quicksort-binary-search-algorithms-

c.html#:~:text=Quicksort%20is%20one%20of%20the,hence%20its%20name%20(binary). Acessado em: 19 de setembro de 2023.

MELHOR CASO, PIOR CASO E CASO MÉDIO. Wikipedia, 2023. Disponível em: https://pt.wikipedia.org/wiki/Melhor_caso,_pior_caso_e_caso_médioAcessado em: 20 de setembro de 2023.

NETO, Nelson Cruz Sampaio. **Análise de Algoritmos: Algoritmos de Ordenação**. Disponível em: https://www2.unifap.br/furtado/files/2016/11/Aula4.pdf. Acessado em: 19 de setembro de 2023.

PAZ, Hallison. **QUICKSORT | Algoritmos #8**. Youtube, 2019. Disponível em: https://www.youtube.com/watch?v=wx5juM9bbFo. Acessado em: 21 de setembro de 2023.

QUICKSORT ALGORITHM. Programiz, s.d. Disponível em: https://www.programiz.com/dsa/quick-sort. Acessado em: 20 de setembro de 2023.

QUICKSORT. Wikipedia, 2023. Disponível em: https://pt.wikipedia.org/wiki/Quicksort. Acessado em: 20 de setembro de 2023.

SAMBOL, Michael. **Classificação rápida em 4 minutos.** Youtube, 2016. Disponível em: https://www.youtube.com/watch?v=Hoixgm4-P4M. Acessado em: 21 de setembro de 2023.