SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho 3 - AEDS 1

Algoritmos de ordenação - Quick Sort

João Victor Graciano Belfort de Andrade
Mateus Henrique Vieira Figueiredo
Vitor Ribeiro Lacerda

Florestal - MG
2022

Sumário

1. Introdução

Esta documentação se refere ao projeto feito no Trabalho Prático III, da disciplina ALGORITMOS E ESTRUTURA DE DADOS I, de código CCF 211. No trabalho, foi sugerido a implementação de um código para análise de algoritmos de QuickSort, e a montagem de gráficos para analisar seu tempo de execução, a fim de aprender sobre a efetividade dos tipos de algoritmos de ordenação. O projeto realizado descrito neste documento está disponível na página do GitHub: QuickSorts.

Algumas bibliotecas externas foram utilizadas para o programa, elas são “time.h”, “stdbool.h”, “ctype.h”, ”stddef.h” e “string.h”. “time.h” foi utilizada para pegar o horário da máquina e calcular o tempo de execução do código, a biblioteca “ctype.h” foi usada para diferenciar letras de dígitos, utilizando a função “isdigit()”. A biblioteca “stdbool.h” foi utilizada para inserção do tipo bool e para lógica booleana, ”stddef.h” foi utilizada para a inserção do macro “NULL”, a biblioteca “string.h” foi utilizada para o tratamento de strings. O programa foi dividido em 2 pastas, sendo uma para armazenamento de testes utilizados para a montagem dos gráficos e outra para o algoritmo solicitado no trabalho.

2. Organização

Na pasta principal do arquivo podem ser encontrados 7 arquivos, cada um destes com os seguintes nomes: “arquivos” (Pasta de arquivos), “src” (Pasta de arquivos), “.gitignore.txt”, “README.md”, “CMakeLists.txt”, “.idea”, “cmake-build-debug”. Os arquivos “cmake-build-debug” e “CMakeLists.txt” são arquivos gerados pela IDE Clin, na compilação do código, não sendo componentes essenciais do trabalho, já os arquivos “README.md” e “.gitignore.txt” são referentes ao repositório do GitHub que contém o trabalho, também não sendo essencial. Os arquivos mencionados do tipo “pasta de arquivos” ainda contém outros, em “arquivos”: “teste_1(2312545543).txt”, “teste_2(5476878798).txt”, “teste_3(3243434554).txt”, “teste_4(8978734545).txt”, “teste_5(1324546457).”txt. Já em src temos: “quickSorts.c”, “quickSorts.h”, “main.c”.

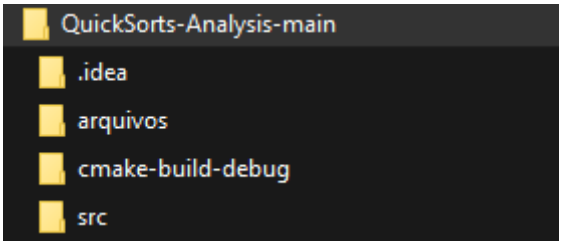

Figura 1- Repositório do projeto

Sendo assim, a pasta principal do projeto contém arquivos referentes ao repositório do github, além de arquivos referentes ao projeto e arquivos criados pela ide CLion na

compilação do código. Em “arquivos” tem-se os testes criados pelo grupo, variando de "teste_1(2312545543).txt" até “teste_5(1324546457).txt”. Já a pasta “src” contém os códigos tanto das tads como do código principal do trabalho.

3. Desenvolvimento

3.1 “quickSort.c” e “quickSort.h”

Nestes arquivos foi descrita a “TAD Tupla”, uma estrutura abstrata para armazenamento das informações referentes aos testes, essas informações são: tempo(segundos), comparações, movimentações. Além da estrutura neste arquivo também são declaradas as funções às funções de quicksort: “quickSortRecursivo”, “quickSortMediana”, “quickSortInsercao”, “quickSortEmpilha”, “quickSortIterativo”. Para as funções de ordenação(quicksort), foram criadas as seguintes funções:

3.1.1 - “criaParticao”: Função utilizada para criar partições no array. Esta função é utilizada para a ordenação dos arrays.

3.1.2 - “criaParticaoMediana”: Função utilizada para criar partições no array. Esta função é utilizada na função “quickSortMediana”, para a ordenação dos arrays.

3.1.3 - “insercaoSort”: Função que organiza o array utilizando do estilo do algoritmo de insertion sort.

3.1.4 - “calculaMediana”: Função criada para realizar o cálculo da mediana de 3 ou 5 valores aleatórios do array. Tal função calcula a mediana para que possa ser usada em outras funções.

3.1.5 - “troca”: Função utilizada para trocar a posição dos elementos no array. Essa função é utilizada pelas funções de ordenação.

3.1.6 - “inicializaTupla”: Função utilizada para a inicialização de tuplas (estrutura).

3.1.7 - “quickSortRecursivo”: Função que utiliza o método básico do algoritmo de quicksort, para a ordenação de um vetor.

3.1.8 - “quickSortMediana”: Função que utiliza da variação do Quicksort recursivo que escolhe o pivô para partição como sendo a mediana de k elementos do vetor, aleatoriamente escolhidos. Utilize k = 3 e k = 5.

3.1.9 - “quickSortInsercao”: Função que utiliza da variação do Quicksort Recursivo, que utiliza o algoritmo de inserção para ordenar partições, com tamanho menor ou igual a m.

3.1.10 - “quickSortEmpilha”: Função que utiliza da variação otimizada do QuicksortRecursivo, que processa primeiro o lado menor da partição.

3.1.11 - "quickSortIterativo": Esta função utiliza a variação que escolhe o pivô como o elemento do meio, mas não é recursiva. Ou seja, esta é uma versão iterativa do Quicksort recursivo.

3.2 "main.c"

Neste arquivo foi implementado o código principal do programa que utiliza dos demais executáveis e headers, recebe a semente do gerador de números aleatórios bem como o nome do arquivo de saída. Estes parâmetros são passados pela linha de comando. Por exemplo: "quicksort saida_semente1.txt", gerando mensagens de erro caso valores inválidos sejam inseridos pelo usuário.

Para as ordenações, foram criadas as seguintes funções:

3.2.1 - "inicializaClock": Função criada para inicializar o clock. A função retorna um double que será utilizado no cálculo do tempo necessário para a ordenação.

3.2.2 - "calculaTempo": Utiliza o double retornado pela função "inicializaClock" para calcular o tempo em que foram realizadas as ordenações.

3.2.3 - "escreveNoArquivo": Função criada para escrever os outputs gerados pelo código principal no arquivo de teste.

3.2.4 - "quickSortsSemK": Função criada para que possa ser feita a chamada de funções quicksort que não possuem constante para ser passada como parâmetro, através de do conceito de "high order functions". Essa forma foi utilizada para uma melhor organização do código. Ao mesmo tempo, essa função salva detalhes da função executada.

3.2.5 - "quickSortsComK": Função criada para que possa ser feita a chamada de funções quicksort que possuem constante para ser passada como parâmetro, através de do conceito de "high order functions". Essa forma foi utilizada para uma melhor organização do código. Ao mesmo tempo, essa função salva detalhes da função executada.

3.2.6 - "geraNumerosAleatorios": Função com o objetivo de gerar números aleatórios com base na seed específica, após o número ser gerado, o mesmo é inserido no array.

3.2.7 - "checaOrdenacao": Função utilizada para chegar a ordenação do array. Esta função percorre o array e checa sua ordenação para saber se a ordenação do array está correta. Caso não esteja uma mensagem de erro aparece no console.

4. Resultados

Após rodar os algoritmos de QuickSort descritos acima, foi montado gráficos com os resultados obtidos utilizando a plataforma google planilhas, com o eixo X representando o número de variáveis no array ordenado em todos os gráficos, e o eixo Y representando respectivamente em cada um dos 3 gráficos o tempo(segundos) gasto para cada ordenação, as movimentações de cada ordenação e as comparações de cada ordenação. Abaixo, estão representados estes gráficos:

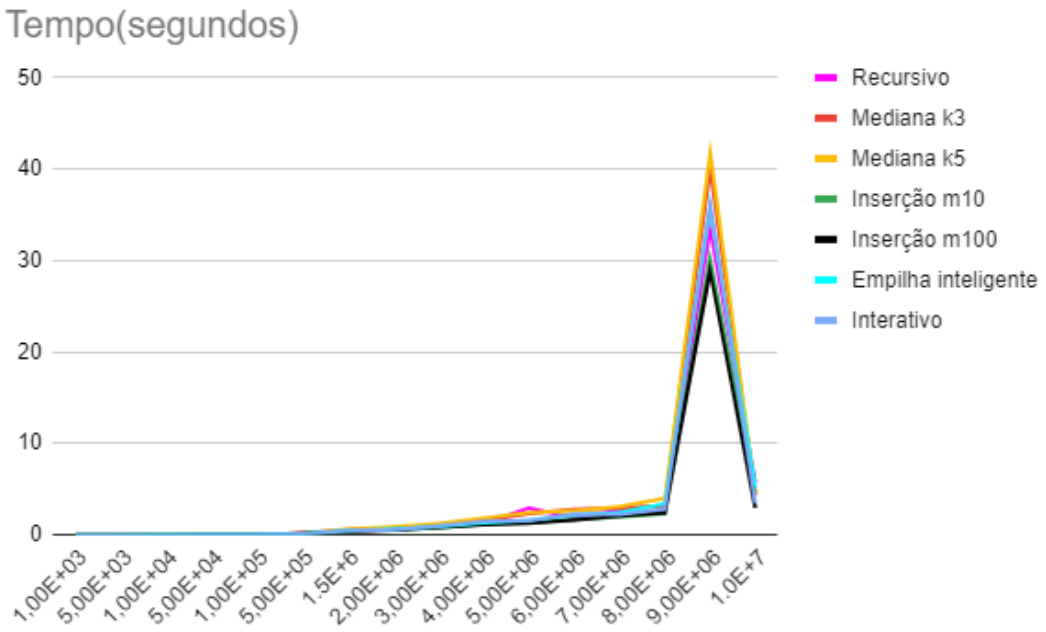

Figura 2- Gráfico de tempo

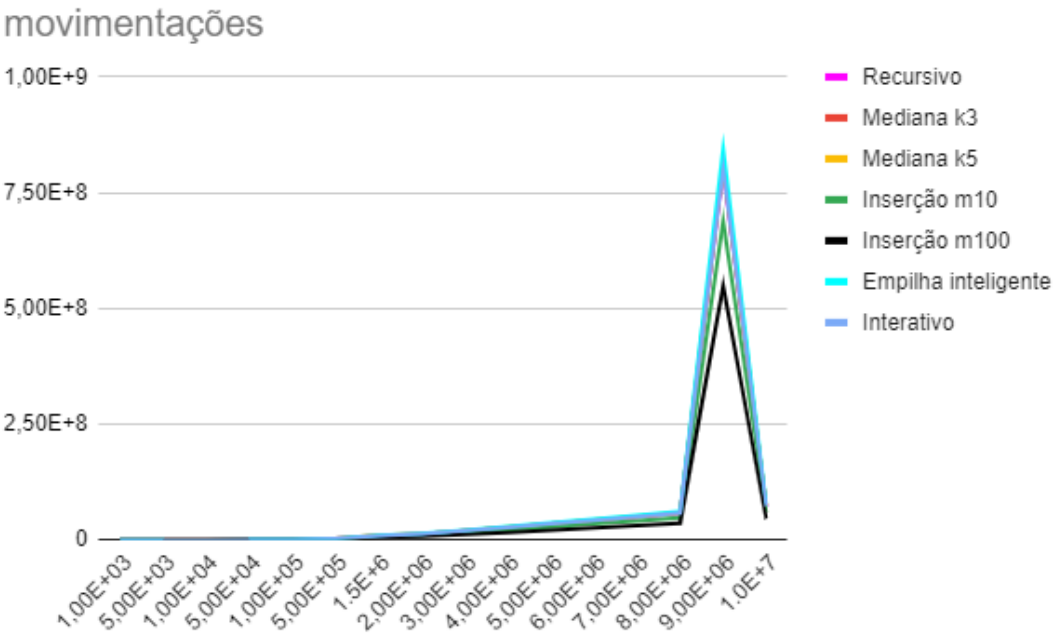

Figura 3 - Gráfico de movimentações

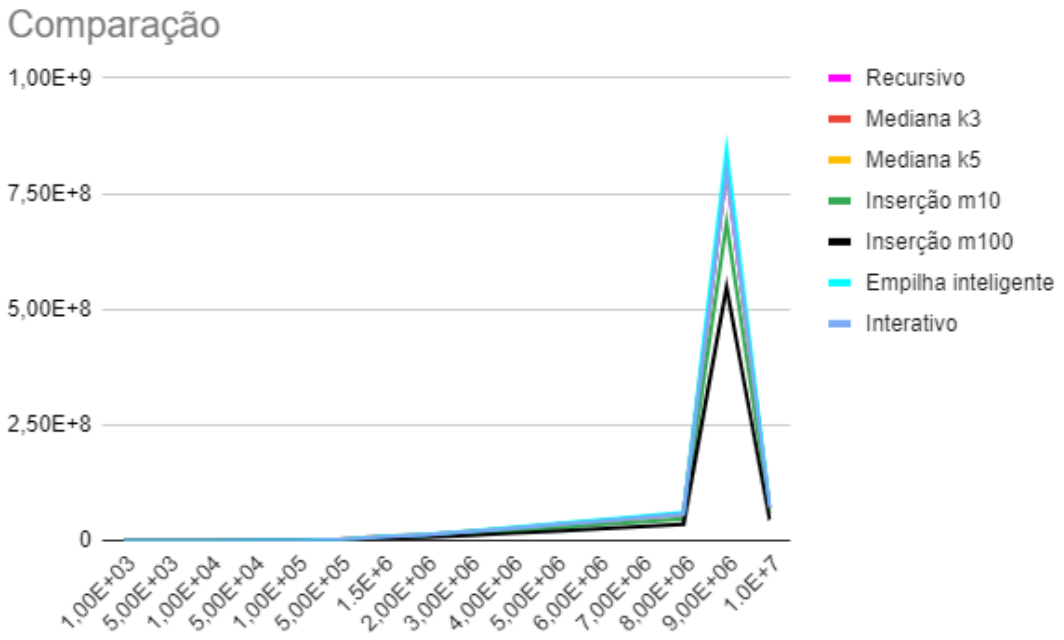

Figura 4 - Gráfico de comparação

Após rodar o código com seeds diferentes, pode-se obter os arquivos de output que contém as informações dos tipos de ordenação, das quantias de elementos, tempo(segundos), movimentações e comparações. Como exemplificado de forma resumida na figura abaixo, em que temos o output para os diversos tipos de QuickSort para o valor de 10 milhões (pelo tamanho dos arquivos de output escolhemos colocar apenas um trecho), para ver os demais outputs, ou trechos, acesse a pasta de "arquivos".

```
Quick Sort Recursivo - 10000000 elementos
5.7120 73418525 73336518

Quick Sort Mediana (k = 3) - 10000000 elementos
4.3310 74203322 74119274

Quick Sort Mediana (k = 5) - 10000000 elementos
4.5850 74630018 74547425

Quick Sort Inserção (m = 10) - 10000000 elementos
4.6490 60679069 60607133

Quick Sort Inserção (m = 100) - 10000000 elementos
2.9870 45813653 45772220

Quick Sort Empilha Inteligente - 10000000 elementos
5.2630 78102570 78015333

Quick Sort Iterativo - 10000000 elementos
3.4680 73418525 73336518
```

Figura 5 - Output (Tempo, comparações, etc.)

Tempo																
Elementos	1,00E+03	5,00E+03	1,00E+04	5,00E+04	1,00E+05	5,00E+05	1.5E+6	2,00E+06	3,00E+06	4,00E+06	5,00E+06	6,00E+06	7,00E+06	8,00E+06	9,00E+06	1.0E+7
Recursivo	0	0,001	0,002	0,012	0,022	0,198	0,485	0,535	0,958	1,215	2,924	1,887	2,72	2,491	33,496	5,712
Mediana k3	0	0,001	0,003	0,016	0,036	0,156	0,557	0,699	1,095	1,561	2,33	2,77	2,959	3,196	40,136	4,331
Mediana k5	0	0,002	0,005	0,039	0,037	0,2	0,609	0,868	1,239	1,861	2,44	2,658	3,094	4,026	41,779	4,585
Inserção m10	0	0,001	0,002	0,022	0,021	0,109	0,394	0,556	0,775	1,174	1,408	1,692	2,008	2,324	30,195	4,649
Inserção m100	0	0,001	0,002	0,027	0,023	0,161	0,356	0,518	0,816	1,158	1,302	1,619	2,072	2,443	28,827	2,987
Empilha inteligente	0	0	0,003	0,029	0,025	0,129	0,489	0,648	0,948	1,332	1,593	2,166	2,439	3,42	35,552	5,263
Interativo	0	0,002	0,002	0,014	0,025	0,124	0,504	0,571	0,98	1,488	1,589	2,27	2,224	2,847	36,11	3,468

Figura 6 - Tabela de tempo (resultados)

Comparações																
Elementos	1,00E+03	5,00E+03	1,00E+04	5,00E+04	1,00E+05	5,00E+05	1.5E+6	2,00E+06	3,00E+06	4,00E+06	5,00E+06	6,00E+06	7,00E+06	8,00E+06	9,00E+06	1.0E+7
Recursivo	2828	17245	36571	213210	447925	2673276	9061173	12460532	19382536	26830091	34340063	41920281	49519537	57663998	801166406	73418525
Mediana k3	2956	17360	37322	217124	462036	2732782	9226829	12615867	19842899	27174145	34760620	42581613	50322371	58255910	808065952	74203322
Mediana k5	2997	17776	37594	219513	465182	2749727	9287990	12737906	19944420	27290018	34937304	42759710	50690675	58703621	812409751	74630018

Inserção m10	1787	11917	25841	157966	334905	2063866	7184006	9925658	15619187	21741232	27981477	34383134	40691087	47464609	687590047	60679069
Inserção m100	962	7675	17133	113746	243993	1525890	5286077	7326679	11476014	16136677	20902540	25728069	30464766	35665271	54853864 9	45813653
Empilha inteligente	2953	17840	37775	221586	469200	2848715	9680244	133011 34	20701120	28586830	36618831	44649648	52669193	61266840	844370306	78102570
Interativo	2828	17245	36571	213210	447925	2673276	9061173	12460532	19382536	26830091	34340063	41920281	49519537	57663998	801166406	73418525

Figura 7 - Tabela de comparações (resultados)

	Movimentações														
Elementos	1,00E+03	5,00E+03	1,00E+04	5,00E+04	1,00E+05	5,00E+05	1.5E+6	2,00E+06	3,00E+06	4,00E+06	5,00E+06	6,00E+06	7,00E+06	8,00E+06	9,00E+06	1.0E+7
Recursivo	2564	15875	33869	202423	431099	2636243	9007586	12402756	19318604	26761924	34268363	41845767	49442714	57585555	801051366	73336518
Mediana k3	2712	16149	34881	207485	446198	2695185	9171807	12556694	19777496	27103864	34687329	42505400	50243212	58174866	807947861	74119274
Mediana k5	2760	16539	35275	210271	450217	2713930	9234862	12680529	19880777	27221847	34865282	42685219	50613584	58624386	812292642	74547425
Inserção m10	1663	11294	24590	152401	325192	2036830	7140441	9877917	15565200	21683084	27919854	34318560	40624327	47396186	687485091	60607133
Inserção m100	944	7594	16974	112932	242429	1519137	5270222	7307671	11452456	16109045	20871301	25693869	30428480	35627477	548464234	45772220
Empilha int.	2685	16463	SS	210483	451482	2808011	9622271	13238480	20632148	28513401	36542022	44570195	52587199	61183059	844249943	78015333
Interativo	2564	15875	33869	202423	431099	2636243	9007586	12402756	19318604	26761924	34268363	41845767	49442714	57585555	801051366	73336518

Figura 8 - Tabela de movimentações (resultados)

Foi sugerido pelo responsável da disciplina que fossem respondidas as seguintes questões com base nas observações feitas, durante o processo de criação, e análise do trabalho:

1- Qual variação tem melhor desempenho, considerando as diferentes métricas. Por quê?

R: Após diversas rodadas de teste, utilizando diferentes seeds, a variação que obteve o melhor desempenho segundo a avaliação gráfica, foi a de inserção m = 100, em todas as três diferentes métricas (Tempo, movimentações e comparações), obteve o melhor desempenho, ou seja, teve o menor número dentre todas. Isto ocorre pois, o algoritmo de Quicksort gasta muito processamento para ordenação de listas pequenas, se comparado com o algoritmo de inserção, e pelo mesmo começar com o "m" não muito grande e não muito pequeno ("m=100" e não "m=10"), o algoritmo de inserção pode obter um desempenho satisfatório.

2- Qual o impacto das variações nos valores de k e de m nas versões Quicksort Mediana(k) e Quicksort Inserção(m)?

R: Em todas as três métricas, o algoritmo de ordenação (Quicksort Mediana) que obteve o melhor desempenho foi o "quickSortMediana" com o "k" = 5, pois o algoritmo consegue escolher um número que seja mais próximo da verdadeira mediana do array, evidenciando assim um grande impacto do valor de "k" nos resultados dos algoritmos de ordenação. Já para o algoritmo de Quicksort Inserção, o que obteve o melhor desempenho para as três métricas, foi o algoritmo de "m" = 100, pelos motivos supracitados na primeira questão.

5. Conclusão

De forma geral, foram implementados TADs e códigos, de forma organizada, comentada e de fácil entendimento para o problema solicitado pelo trabalho, foi decidida uma abordagem prática, que ao fim foi uma ótima opção abstraindo o código, e o deixando mais fácil de ser visualizado e compreendido, e utilizado. Os resultados saíram assim como o esperado, sendo que algumas implementações feitas, apesar de deixarem o código mais extenso, facilitam seu uso e entendimento, assim como as tabelas e gráficos criados para a análise, sendo assim o trabalho, de forma geral atende as exigências passadas pelo Professor do curso de AEDES-2021/2.

6. Referências

Para versionar o projeto foi utilizado o Github [1].
Código “Interative Quicksort" [2].
Código “QuickSort Tail Call Optimization” [3].

[1] Github. Disponível em: <https://github.com/Mateus-Henr/QuickSorts-Analysis> Último acesso em: 21 de março de 2022.

[2] Geeks for Geeks, **Iterative Quick Sort,** 6 setembro de 2022. Disponivel em: <https://www.geeksforgeeks.org/iterative-quick-sort/> Último acesso em: 18 de março de 2022.

[3] Geeks for Geeks, **QuickSort Tail Call Optimization,** 21 de Jan. de 2022. Disponivel em : <https://www.geeksforgeeks.org/quicksort-tail-call-optimization-reducing-worst-case-space-log-n//> Último acesso em: 17 de março de 2022.