



**UNIVERSIDADE FEDERAL DO CEARÁ
CAMPUS DE CRATEÚS
ESTRUTURA DE DADOS**

RELATÓRIO – TRABALHO FINAL

DISCENTES:

JHAMESON LUCAS PEREIRA FARIAS - 475827

VICTOR GABRIEL MARTINS DE OLIVEIRA LOIOLA - 473091

DOCENTE: ARNALDO BARRETO VILA NOVA

DEZEMBRO DE 2019

INTRODUÇÃO

Este relatório tem como objetivo mostrar como foi o desenvolvimento da atividade proposta pelo docente Arnaldo Barreto Vila Nova, que propôs que nós implementássemos algoritmos de ordenação: Insertion Sort, Selection Sort, MergeSort, Bubble Sort, QuickSort e um outro opcional, com listas duplamente encadeadas e exibíssemos os resultados de uma lista com elementos aleatórios, ordenados em ordem crescente e decrescente com o tempo de execução, quantidade de trocas e verificações. E cada algoritmo deveria ser testado com os mesmos arquivos 10 vezes com 10 listas diferentes nos casos de 10, 100 e 1000 valores aleatórios.

ALGORITMOS DE ORDENAÇÃO

insertion Sort

Insertion Sort, ou ordenação por inserção, é o algoritmo de ordenação que, dado uma estrutura (array, lista) constrói uma matriz final com um elemento de cada vez, uma inserção por vez. Assim como algoritmos de ordenação quadrática, é bastante eficiente para problemas com pequenas entradas, sendo o mais eficiente entre os algoritmos desta ordem de classificação. Podemos fazer uma comparação do Insertion Sort com o modo de como algumas pessoas organizam um baralho num jogo de cartas. Imagine que você está jogando cartas. Você está com as cartas na mão e elas estão ordenadas. Você recebe uma nova carta e deve colocá-la na posição correta da sua mão de cartas, de forma que as cartas obedecam a ordenação.

A cada nova carta adicionada à sua mão de cartas, a nova carta pode ser menor que algumas das cartas que você já tem na mão ou maior, e assim, você começa a comparar a nova carta com todas as cartas na sua mão até encontrar sua posição correta. Você insere a nova carta na posição correta, e, novamente, sua mão é composta de cartas totalmente ordenadas. Então, você recebe outra carta e repete o mesmo procedimento. Então outra carta, e outra, e assim por diante, até você não receber mais cartas. Esta é a ideia por trás da ordenação por inserção. Percorra as posições do array, começando com o índice 1 (um). Cada nova posição é como a nova carta que você recebeu, e você precisa inseri-la no lugar correto no subarray ordenado à esquerda daquela posição.

complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n^2)$
complexidade melhor caso	$O(n)$
complexidade de espaços pior caso	$O(n)$ total, $O(1)$ auxiliar
estabilidade	estável

Selection Sort

A ordenação por seleção (do inglês, selection sort) é um algoritmo de ordenação baseado em se passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida), depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os $n - 1$ elementos restantes, até os últimos dois elementos.

É composto por dois laços, um laço externo e outro interno. O laço externo serve para controlar o índice inicial e o interno percorre todo o vetor. Na primeira iteração do laço externo o índice começa de 0 e cada iteração ele soma uma unidade até o final do vetor e o laço mais interno

percorre o vetor começando desse índice externo + 1 até o final do vetor.

complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n^2)$
complexidade melhor caso	$O(n^2)$
complexidade de espaços pior caso	$O(n)$ total, $O(1)$ auxiliar

Bubbler Sort

O bubble sort, ou ordenação por flutuação (literalmente "por bolha"), é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vetor diversas vezes, e a cada passagem fazer flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

No melhor caso, o algoritmo executa n operações relevantes, onde n representa o número de elementos do vetor. No pior caso, são feitas n elevado 2 operações. A complexidade desse algoritmo é de ordem quadrática. Por isso, ele não é recomendado para programas que precisem de velocidade e operem com quantidade elevada de dados.

complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n^2)$
complexidade melhor caso	$O(n)$
complexidade de espaços pior caso	$O(1)$ auxiliar

Cocktail sort

Cocktail shaker sort, também conhecido como bubble sort bidirecional, cocktail sort, shaker sort (o qual também pode se referir a uma variação do insertion sort), ripple sort, shuffle sort,] ou shuttle sort, é uma variante do bubble sort, que é um algoritmo com não-estável e efetua Ordenação por comparação.

O algoritmo difere de um bubble sort no qual ordena em ambas as direções a cada iteração sobre a lista. Esse algoritmo de ordenação é levemente mais difícil de implementar que o bubble sort, e resolve o problema com os chamados coelhos e tartarugas no bubble sort.

Ele possui performance levemente superior ao bubble sort, mas não melhora a performance assintótica; assim como o bubble sort, não é muito usado na prática (insertion sort é escolhido para ordenação simples), embora seja usado para fins didáticos.

complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n^2)$
complexidade melhor caso	$O(n)$
otimo	Não
estabilidade	não-estável

Merge Sort

O merge sort, ou ordenação por mistura, 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.

estrutura de dados	Array, Listas ligadas
complexidade pior caso	$\Theta(n \log n)$
complexidade caso médio	$\Theta(n \log n)$
complexidade melhor caso	$\Theta(n \log n)$ típico, $\Theta(n)$ variante natural
complexidade de espaços pior caso	$\Theta(n \log n)$

Quick Sort

O algoritmo quicksort é um método de ordenação muito rápido e eficiente, inventado por C.A.R. Hoare em 1960, quando visitou a Universidade de Moscovo como estudante. Naquela época, Hoare trabalhou 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 mais fácil e rápido. Foi publicado em 1962 após uma série de refinamentos.

O quicksort é um algoritmo de ordenação por comparação não-estável.

complexidade pior caso	$O(n^2)$
complexidade caso médio	$O(n \log n)$
complexidade melhor caso	$O(n \log n)$
complexidade de espaços pior caso	$O(n)$
otimo	Não
estabilidade	não-estável

DESENVOLVIMENTO

Foram feitos uma série de arquivos e projetos até que encontramos uma melhor forma de organizar e padronizar o projeto. Dos arquivos .txt, foram feitos três tipos os “doc10.txt” que armazenavam 10 valores aleatórios, os “doc100.txt” com 100 valores aleatórios e os “doc1000.txt” com 1000 valores aleatórios. Para os valores em ordem crescente e decrescente não foi feito armazenamento nem uma série de testes diferentes, pois são sempre os mesmos valores, mesma quantidade de troca e verificações, portanto foi necessário apenas um teste com 10, 100 e 1000 elementos para cada algoritmo.

RESULTADOS

- 1 - Insertion Sort: O algoritmo em questão tem complexidade no melhor caso de $O(n)$, pior caso $O(n^2)$ e caso médio $O(n^2)$.
- 2 - Selection Sort: complexidade no melhor, pior e caso médio é de $O(n^2)$;
- 3 - Bubble Sort: complexidade no pior caso e médio é de $O(n^2)$ e no melhor caso $O(n)$;
- 4 - Merge Sort: melhor caso, pior e médio $O(n \log n)$;
- 5 - QuickSort: complexidade no pior caso é de $O(n^2)$, no melhor e caso médio $O(n \log n)$;
- 6 - Cocktail sort: complexidade no pior e médio caso $O(n^2)$ e no melhor caso $O(n)$.

Grupo de testes 1 (10 elementos)

1 - *Insertion Sort*

Elementos em ordem crescente e já ordenados:

Tempo: 0.016 segundos

Verificações: 18

Trocas: 0

Elementos em ordem decrescente não ordenados:

Tempo: 0.128 segundos

Verificações: 18

Trocas: 45

Elementos em ordem Aleatória:

INSERTION SORT			
DOCUMENTOS COM 10 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc10_1.txt"	0.117 s	18	15
"Doc10_2.txt"	0.129 s	18	17
"Doc10_3.txt"	0.125 s	18	13
"Doc10_4.txt"	0.117 s	18	22
"Doc10_5.txt"	0.093 s	18	22
"Doc10_6.txt"	0.128 s	18	15
"Doc10_7.txt"	0.095 s	18	20
"Doc10_8.txt"	0.105 s	18	27
"Doc10_9.txt"	0.098 s	18	20
"Doc10_10.txt"	0.1 s	18	28

Médias valores aleatórios:

Média de tempo: 0.112 segundos

Média de trocas: aproximadamente 20 trocas

Média de verificações: 18

2 - Selection Sort

Elementos em ordem crescente já ordenados:

Tempo: 0.122 segundos

Verificações: 10

Trocas: 0

Elementos em ordem decrescente não ordenados:

Tempo: 0.088 segundos

Verificações: 10

Trocas: 5

Elementos em ordem Aleatória:

SELECTION SORT				
DOCUMENTOS COM 10 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS	
"Doc10_1.txt"	0.114 s	44	6	
"Doc10_2.txt"	0.111 s	44	5	
"Doc10_3.txt"	0.131 s	44	3	
"Doc10_4.txt"	0.104 s	44	7	
"Doc10_5.txt"	0.133 s	44	6	
"Doc10_6.txt"	0.94 s	44	6	
"Doc10_7.txt"	0.98 s	44	6	
"Doc10_8.txt"	0.170 s	44	6	
"Doc10_9.txt"	0.233 s	44	6	
"Doc10_10.txt"	0.107 s	44	5	

Médias valores aleatórios:

Média de tempo: 0.133 segundos

Média de trocas: quase 6 trocas

Média de verificações: 44

3 - Bubble Sort

Elementos em ordem crescente:

Tempo: 0.144 segundos

Verificações: 45

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.105 segundos

Verificações: 45

Trocas: 45

Elementos em ordem Aleatória:

BUBBLE SORT				
DOCUMENTOS COM 10 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS	
"Doc10_1.txt"	0.103 s	45	15	
"Doc10_2.txt"	0.108 s	45	17	
"Doc10_3.txt"	0.111 s	45	13	
"Doc10_4.txt"	0.165 s	45	22	
"Doc10_5.txt"	0.108 s	45	22	
"Doc10_6.txt"	0.098 s	45	15	
"Doc10_7.txt"	0.122 s	45	20	
"Doc10_8.txt"	0.133 s	45	27	
"Doc10_9.txt"	0.099 s	45	20	
"Doc10_10.txt"	0.108 s	45	28	

Médias valores aleatórios:

Média de tempo: 0.116 segundos

Média de trocas: aproximadamente 20 trocas

Média de verificações: 45

4 - Merge Sort

Elementos em ordem crescente já ordenados:

Tempo: 0.144 segundos

Verificações: 20

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.105 segundos

Verificações: 20

Trocas: 0

Elementos em ordem Aleatória:

MERGE SORT			
DOCUMENTOS COM 10 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc10_1.txt"	0.088 s	20	0
"Doc10_2.txt"	0.08	20	0
"Doc10_3.txt"	0.127 s	20	0
"Doc10_4.txt"	0.079 s	20	0
"Doc10_5.txt"	0.07 s	20	0
"Doc10_6.txt"	0.10 s	20	0
"Doc10_7.txt"	0.098 s	20	0
"Doc10_8.txt"	0.067 s	20	0
"Doc10_9.txt"	0.09 s	20	0
"Doc10_10.txt"	0.10 s	20	0

Médias valores aleatórios:

Média de tempo: 0.0899 segundos

Média de trocas: 20

Média de verificações: 0

5 - *QuickSort*

6 - *Cocktail sort*

Elementos em ordem crescente:

Tempo: 0.140 s

Verificações: 10

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.170 segundos

Verificações: 100

Trocas: 45

Elementos em ordem Aleatória:

COCKTAIL SORT			
DOCUMENTOS COM 10 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc10_1.txt"	0.161 s	60	15
"Doc10_2.txt"	0.144 s	80	17
"Doc10_3.txt"	0.307 s	40	13
"Doc10_4.txt"	0.240 s	90	22
"Doc10_5.txt"	0.212 s	80	22
"Doc10_6.txt"	0.152 s	80	15
"Doc10_7.txt"	0.145 s	70	20
"Doc10_8.txt"	0.121 s	80	27
"Doc10_9.txt"	0.141 s	90	20
"Doc10_10.txt"	0.131 s	80	28

Médias valores aleatórios:

Média de tempo:0.175 segundos

Média de trocas: aproximadamente 20 trocas

Média de verificações:75

Grupo de testes 2 (100 Elementos)

1 - Insertion Sort

Elementos em ordem crescente:

Tempo: 0.016 s

Verificações: 198

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.118 s

Verificações: 198

Trocas: 4950

Elementos em ordem Aleatória:

INSERTION SORT			
DOCUMENTOS COM 100 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc100_1.txt"	0.159 s	198	2467
"Doc100_2.txt"	0.173 s	198	2425
"Doc100_3.txt"	0.154 s	198	2436
"Doc100_4.txt"	0.071 s	198	2596
"Doc100_5.txt"	0.301 s	198	2500
"Doc100_6.txt"	0.167 s	198	2375
"Doc100_7.txt"	0.153 s	198	2689
"Doc100_8.txt"	0.131 s	198	2480
"Doc100_9.txt"	0.164 s	198	2603
"Doc100_10.txt"	0.169 s	198	2758

Médias valores aleatórios:

Média de tempo: 0.164 segundos

Média de trocas: 2532,9

Média de verificações: 198

2 - Selection Sort

Elementos em ordem crescente:

Tempo: 0.104 segundos

Verificações: 100

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.114 segundos

Verificações: 100

Trocas: 50

Elementos em ordem Aleatória:

SELECTION SORT			
DOCUMENTOS COM 100 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc100_1.txt"	0.84 s	4949	82
"Doc100_2.txt"	0.232 s	4949	80
"Doc100_3.txt"	0.156 s	4949	82
"Doc100_4.txt"	0.142 s	4949	82
"Doc100_5.txt"	0.159 s	4949	78
"Doc100_6.txt"	0.197 s	4949	80
"Doc100_7.txt"	0.148 s	4949	77
"Doc100_8.txt"	0.145 s	4949	78
"Doc100_9.txt"	0.132 s	4949	80
"Doc100_10.txt"	0.161 s	4949	77

Médias valores aleatórios:

Média de tempo: 0,155 segundos

Média de trocas: 4949

Média de verificações: 79,6

3 - Bubble Sort

Elementos em ordem crescente:

Tempo: 0.116 segundos

Verificações: 4950

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.200 segundos

Verificações: 4950

Trocas: 4950

Elementos em ordem Aleatória:

BUBBLE SORT			
DOCUMENTOS COM 100 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc100_1.txt"	0.146 s	4950	2463
"Doc100_2.txt"	0.171 s	4950	2379
"Doc100_3.txt"	0.125 s	4950	2423
"Doc100_4.txt"	0.165 s	4950	2533
"Doc100_5.txt"	0.114 s	4950	2459
"Doc100_6.txt"	0.092 s	4950	2311
"Doc100_7.txt"	0.107 s	4950	2533
"Doc100_8.txt"	0.116 s	4950	2374
"Doc100_9.txt"	0.118 s	4950	2547
"Doc100_10.txt"	0.140 s	4950	2665

Médias valores aleatórios:

Média de tempo: 0,129 segundos

Média de trocas: 2469

Média de verificações: 4950

4 - Merge Sort

Elementos em ordem crescente:

Tempo: 0.126s

Verificações: 200

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.147

Verificações: 200

Trocas: 0

Elementos em ordem Aleatória:

MERGE SORT			
DOCUMENTOS COM 100 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc100_1.txt"	0.11 s	200	0
"Doc100_2.txt"	0.1 s	200	0
"Doc100_3.txt"	0.08 s	200	0
"Doc100_4.txt"	0.084 s	200	0
"Doc100_5.txt"	0.1 s	200	0
"Doc100_6.txt"	0.09 s	200	0
"Doc100_7.txt"	0.07 s	200	0
"Doc100_8.txt"	0.11 s	200	0
"Doc100_9.txt"	0.09 s	200	0
"Doc100_10.txt"	0.09 s	200	0

Médias valores aleatórios:

Média de tempo: 0.092 segundos

Média de trocas: 0

Média de verificações: 200

5 - QuickSort

6 - Cocktail sort

Elementos em ordem crescente já ordenados:

Tempo: 0.138s

Verificações: 100

Trocas: 0

Elementos em ordem decrescente não ordenados:

Tempo: 0.125s

Verificações: 100

Trocas: 45

Elementos em ordem Aleatória:

COCKTAIL			
DOCUMENTOS COM 100 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc100_1.txt"	0.458 s	9000	2467
"Doc100_2.txt"	0.432 s	7800	2425
"Doc100_3.txt"	0.281 s	8900	2436
"Doc100_4.txt"	0.339 s	9100	2596
"Doc100_5.txt"	0.359 s	8000	2500
"Doc100_6.txt"	0.454 s	9000	2375
"Doc100_7.txt"	0.318 s	8600	2689
"Doc100_8.txt"	0.233 s	9600	2480
"Doc100_9.txt"	0.311 s	8600	2603
"Doc100_10.txt"	0.311 s	9400	2758

Médias valores aleatórios:

Média de tempo:0.355 segundos

Média de trocas: aproximadamente 2533

Média de verificações:8800

Grupo de testes 3 (1000 elementos)

1 - Insertion Sort

Elementos em ordem crescente:

Tempo: 0.289 segundos

Verificações: 999

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.315 segundos

Verificações: 999

Trocas: 499500

Elementos em ordem Aleatória:

INSERTION SORT				
DOCUMENTOS COM 1000 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS	
"Doc1000_1.txt"	0.193 s	1998	253018	
"Doc1000_2.txt"	0.339 s	1998	253211	
"Doc1000_3.txt"	0.215 s	1998	248418	
"Doc1000_4.txt"	0.240 s	1998	256139	
"Doc1000_5.txt"	0.204 s	1998	251657	
"Doc1000_6.txt"	0.230 s	1998	248803	
"Doc1000_7.txt"	0.247 s	1998	252588	
"Doc1000_8.txt"	0.163 s	1998	251632	
"Doc1000_9.txt"	0.185 s	1998	254270	
"Doc1000_10.txt"	0.259 s	1998	255661	

Médias valores aleatórios:

Média de tempo: 0,227 segundos

Média de trocas: 227.280

Média de verificações: 1998

2 - Selection Sort

Elementos em ordem crescente:

Tempo: 0.293 segundos

Verificações: 1000

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.295 segundos

Verificações: 1000

Trocas: 500

Elementos em ordem Aleatória:

SELECTION SORT			
DOCUMENTOS COM 1000 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc1000_1.txt"	0.328 s	499499	918
"Doc1000_2.txt"	0.321 s	499499	916
"Doc1000_3.txt"	0.377 s	499499	907
"Doc1000_4.txt"	0.332 s	499499	915
"Doc1000_5.txt"	0.381 s	499499	914
"Doc1000_6.txt"	0.295 s	499499	916
"Doc1000_7.txt"	0.286 s	499499	912
"Doc1000_8.txt"	0.312 s	499499	922
"Doc1000_9.txt"	0.251 s	499499	905
"Doc1000_10.txt"	0.277 s	499499	918

Médias valores aleatórios:

Média de tempo: 0.316s

Média de trocas: 914,3

Média de verificações: 499499

4 - Merge Sort

Elementos em ordem crescente:

Tempo: 0.532s

Verificações: 2000

Trocas: 0

Elementos em ordem decrescente:

Tempo: 0.828

Verificações: 2000

Trocas: 0

Elementos em ordem Aleatória:

MERGE SORT			
DOCUMENTOS COM 1000 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc1000_1.txt"	0.09 s	2000	0
"Doc1000_2.txt"	0.17 s	2000	0
"Doc1000_3.txt"	0.18 s	2000	0
"Doc1000_4.txt"	0.19 s	2000	0
"Doc1000_5.txt"	0.18 s	2000	0
"Doc1000_6.txt"	0.17 s	2000	0
"Doc1000_7.txt"	0.19 s	2000	0
"Doc1000_8.txt"	0.15 s	2000	0
"Doc1000_9.txt"	0.17 s	2000	0
"Doc1000_10.txt"	0.16 s	2000	0

Médias valores aleatórios:

Média de tempo: 0.16 s

Média de trocas: 0

Média de verificações: 2000

5 - QuickSort

6 - Cocktail sort

Elementos em ordem crescente:

Tempo: 0.261s

Verificações: 1000

Trocas: 0

Elementos em ordem decrescente:

Tempo: 1.1915

Verificações: 1000000

Trocas: 499500

Elementos em ordem Aleatória:

COCKTAIL			
DOCUMENTOS COM 1000 ELEMENTOS ALEATÓRIOS	TEMPO	VERIFICAÇÕES	TROCAS
"Doc1000_1.txt"	2.297 s	947000	253018
"Doc1000_2.txt"	2.357 s	985000	253211
"Doc1000_3.txt"	2.107 s	982000	248418
"Doc1000_4.txt"	2.348 s	951000	256139
"Doc1000_5.txt"	2.615 s	971000	251657
"Doc1000_6.txt"	2.128 s	976000	248803
"Doc1000_7.txt"	1.441 s	976000	252588
"Doc1000_8.txt"	2.211 s	987000	251632
"Doc1000_9.txt"	2.320 s	929000	254270
"Doc1000_10.txt"	1.159 s	946000	255661

Médias valores aleatórios:

Média de tempo:2.098 segundos

Média de trocas: aproximadamente 252540 trocas

Média de verificações:965000

DIFICULDADES

Foram várias as dificuldades encontradas antes e durante o desenvolvimento do projeto. No início foi a interpretação sobre o que o trabalho realmente estava propondo, depois que tudo ficou claro os obstáculos foram implementar o Merge e Quick sort, pois o Select, insert e Bubble são algoritmos que ordenam tendo como base a troca de elementos. A implementação do Merge se deu de forma alternativa ao método utilizado em vetores, seguindo a lógica de dividindo a lista em partes menores e intercalar e posteriormente intercalar a toda a lista para retornar de maneira ordenada. Outra dificuldade foi no quesito manipulação de arquivos, mas ligeiramente foi sanada com auxílio que o docente deixou disponível no sistema acadêmico. É inegável dizer que não houve frustrações na implementação, e que o cansaço e ansiedade em entregar o trabalho no prazo e de maneira que atendesse os requisitos que esse desafio gerou. Mas a maioria das dificuldades foram “mortas” após aulas com tira dúvidas e os horários de atendimento que o docente ofereceu e outras maneiras de se resolver os problemas que surgiram a cada desenvolvimento dos algoritmos.

DIVISÃO DE TAREFAS

As tarefas foram divididas de maneira espontânea, mas aproveitando cada habilidade dos membros, ressaltando que as dificuldades que cada membro tem foram trabalhadas de tal maneira que algumas delas foram sanadas ao longo do desenvolvimento do trabalho.

É válido ressaltar que para algumas funções foi necessária ajudas externas para enxergar a lógica e estrutura da função.

CONCLUSÃO

Após diversos testes e pelo simples fato de sabermos os melhores e piores casos de cada algoritmo chegamos à seguinte conclusão: Merge Sort conseguiu se sair mais rápido, no entanto o seu custo computacional comparado aos outros algoritmos de ordenação é relevante pois o mesmo utilizar recursividade, o que ocasiona anteriormente já mencionado, maior custo de memória.

MATERIAIS UTILIZADOS

Sites:

https://pt.wikipedia.org/wiki/Insertion_sort
https://pt.wikipedia.org/wiki/Selection_sort
https://pt.wikipedia.org/wiki/Bubble_sort
https://pt.wikipedia.org/wiki/Cocktail_sort
<https://www.geeksforgeeks.org/merge-sort-for-linked-list/>

Plataforma de desenvolvimento utilizada foi o “FALCON C++” e “SUBLIME”