INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DO PIAUÍ
CURSO SUPERIOR TECNÓLOGO EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS – ADS

DISCIPLINA DE ESTRUTURA DE DADOS

PROFESSOR: MARCEL RAIMUNDO DE SOUZA MOURA

DISCENTES:

OTÁVIO BRUNO SOUSA MARTINS

DAVID RUFINO FERREIRA

DÂMAYRA DOS REIS SANTOS

RELATÓRIO SOBRE COMPARAÇÃO DE DESEMPENHO: ORDENAÇÃO E ANÁLISE DE ALGORITMOS DE ORDENAÇÃO.

Parnaíba
21/12/2023

RELATÓRIO SOBRE COMPARAÇÃO DE DESEMPENHO: ORDENAÇÃO E ANÁLISE DE ALGORITMOS DE ORDENAÇÃO

1. IDENTIFICAÇÃO

Título: Comparação de Desempenho: ordenação e análise

Discente(s): Otávio Bruno Sousa Martins; David Rufino Ferreira; Damayra dos Reis Santos

Docente(s): Marcel Raimundo de Souza Moura

Unidade Curricular: Estrutura de dados.

Curso: Curso superior tecnólogo em análise e desenvolvimento de sistemas – ADS

Período: 21/12/2023

Objetivo: Para cada um dos algoritmos de ordenação discutidos em sala de aula será necessário realizar testes com conjuntos de dados específicos e analisar os resultados obtidos.

2. INTRODUÇÃO

Os algoritmos de ordenação em linguagem C desempenham um papel fundamental na organização eficiente de conjuntos de dados. Esses algoritmos são estruturas lógicas e procedimentos específicos desenvolvidos para arranjar elementos de uma lista, matriz ou qualquer estrutura de dados de acordo com uma sequência específica, seja em ordem crescente, decrescente ou baseada em algum critério definido. Os algoritmos de ordenação em C variam em termos de complexidade, eficiência e propósito, desde os métodos simples como Bubble Sort e Selection Sort até

abordagens mais sofisticadas, como Merge Sort, Quick Sort e Heap Sort (Cormen et al. 2009). Compreender e implementar esses algoritmos não apenas permite a organização eficiente de dados, mas também proporciona uma base sólida para otimização e resolução de problemas em uma ampla gama de aplicações. Os testes de desempenho desempenham um papel crucial na avaliação e comparação de algoritmos de ordenação. Quando se trata de selecionar o algoritmo mais adequado para uma determinada tarefa de ordenação, é fundamental entender como diferentes algoritmos se comportam em termos de eficiência e velocidade de execução (Sedgewick et al. 2019).

Os testes de desempenho em algoritmos de ordenação envolvem a análise e medição de diversos parâmetros, como o tempo de execução, o consumo de memória e como vai ser o comportamento em diferentes cenários de entrada, como isso é possível proporcionar insights valiosos sobre a eficácia de cada algoritmo.

Esses testes são essenciais para comparar o desempenho relativo de algoritmos de ordenação em situações do mundo real, ajudando os desenvolvedores a escolher o algoritmo mais apropriado para lidar com conjuntos de dados específicos. Ao avaliar e entender as características de desempenho de cada algoritmo, os testes proporcionam uma base sólida para a tomada de decisão considerando fatores como eficiência por tempo, utilização de recursos e adaptabilidade a diferentes tamanhos e tipos de conjuntos de dados (Cormen et al. 2009)

2.1 OBJETIVO

Para cada um dos algoritmos de ordenação discutidos em sala de aula, realizar testes com conjuntos de dados específicos e analisar os resultados obtidos.

2.2 OBJETIVOS ESPECÍFICOS

- Rodar os testes de ordenamento com os algoritmos Merge Sort, Quick Sort, Insertion Sort, Selection Sort, Heap Sort e Tim Sort.
- Coletar o tempo de execução de cada algoritmo com conjunto de dados determinados para os melhores, medio e pior caso para cada um.
- Com os tempos de execução observados montar uma análise estatística obtendo média e desvio padrão.
- Verificar se os resultados obtidos condizem com o esperado para os algoritmos

3. MATERIAIS E MÉTODOS

3.1 MATERIAIS UTILIZADOS

Foram utilizados os seguintes materiais para a prática realizada:

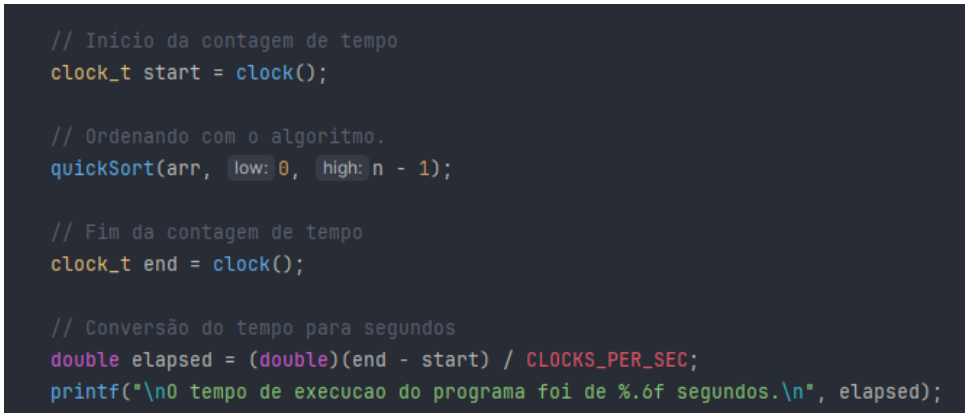
- Notebook Acer Nitro 5 com as seguintes especificações: Processador Ryzen 7 4800h, 24 GB de memória RAM.
- IDE utilizada: CLion

3.2 PROCEDIMENTO EXPERIMENTAL

Para a coleta do tempo de execução dos algoritmos de ordenação foi adotada a seguinte biblioteca <time.h>: que permite o acesso a funções relacionadas ao tempo, como clock() para medir a quantidade de tempo que uma seção de código leva para ser executada.

Conforme é possível observar na figura 1. abaixo, primeiro é criado uma variável "start" para coletar o tempo inicial da contagem, em sequência o algoritmo que será medido é inserido, e após isso a variável "end" para coletar o tempo final da contagem. Depois de todo esse processo é feito a subtração entre "start" e "end" para saber o tempo em clocks, logo em seguida esse tempo em clocks são convertidos para segundos com a função da biblioteca <time.h> chamada CLOCKS_PER_SEC.

Figura 1- Uso da biblioteca <time.h>



```
// Inicio da contagem de tempo
clock_t start = clock();

// Ordenando com o algoritmo.
quickSort(arr, low: 0, high: n - 1);

// Fim da contagem de tempo
clock_t end = clock();

// Conversão do tempo para segundos
double elapsed = (double)(end - start) / CLOCKS_PER_SEC;
printf("\nO tempo de execucao do programa foi de %.6f segundos.\n", elapsed);
```

Para a análise de comparação foram considerados 3 tamanhos de array (Menor, Médio e Grande): O primeiro com 100.000 elementos que em nossa análise representa o tamanho menor em quantidade de elementos, o segundo tamanho de 250.000 elementos considerado um tamanho intermediário e um terceiro conjunto com 500.000 elementos considerado um tamanho grande de dados para a análise, buscando assim testar os

algoritmos em diferentes situações que sejam realmente representativas em termos de resultados. Para os testes de melhor caso como demonstrado na fig-2, o array foi preenchido com valores de 1 até N em ordem crescente. Para medio caso o array é preenchido com números aleatórios de 1 até N (fig-3) e para o teste de pior caso (fig-4) o Array foi preenchido com valores de 1 até N em ordem decrescente.

Figura 2-Teste Melhor Caso.

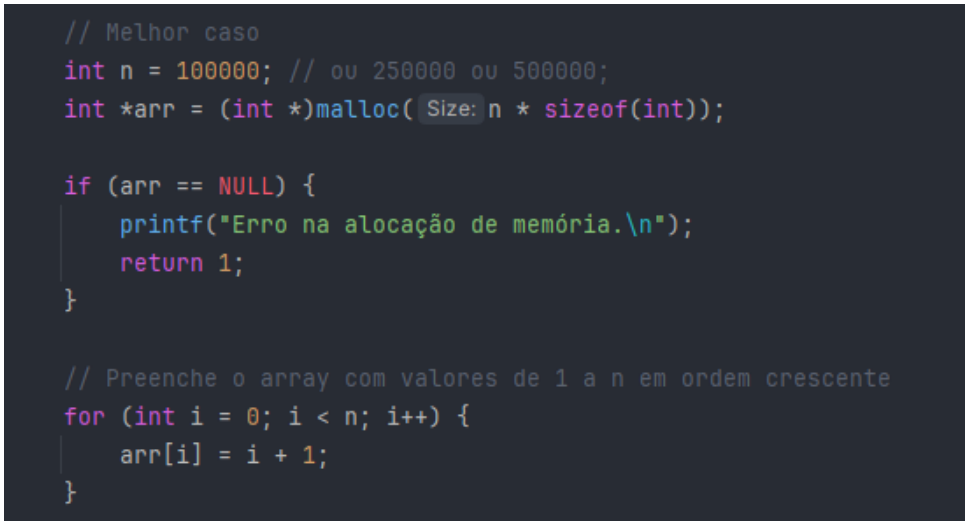


Figura 3- Teste Médio Caso.

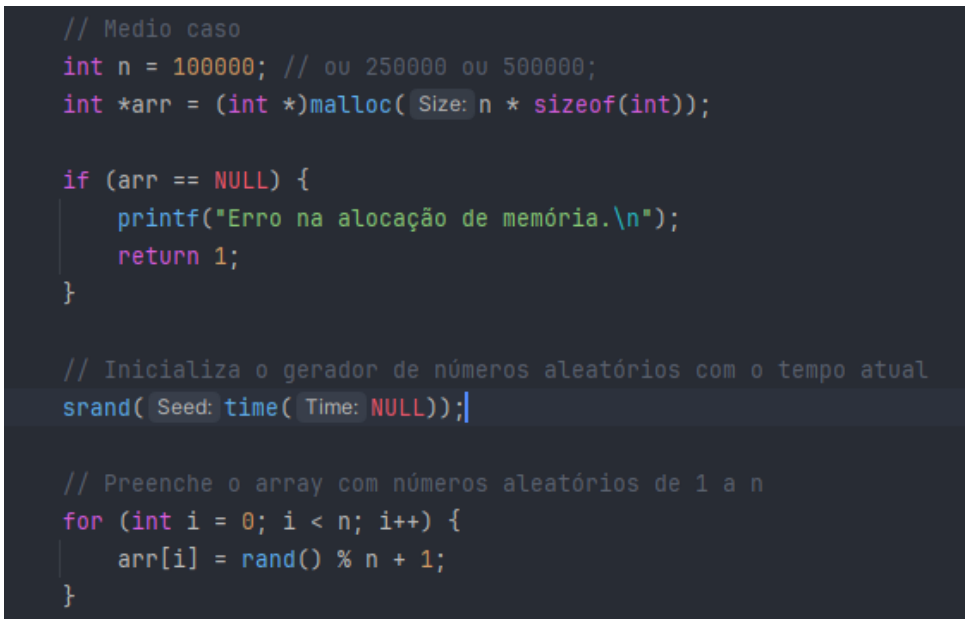
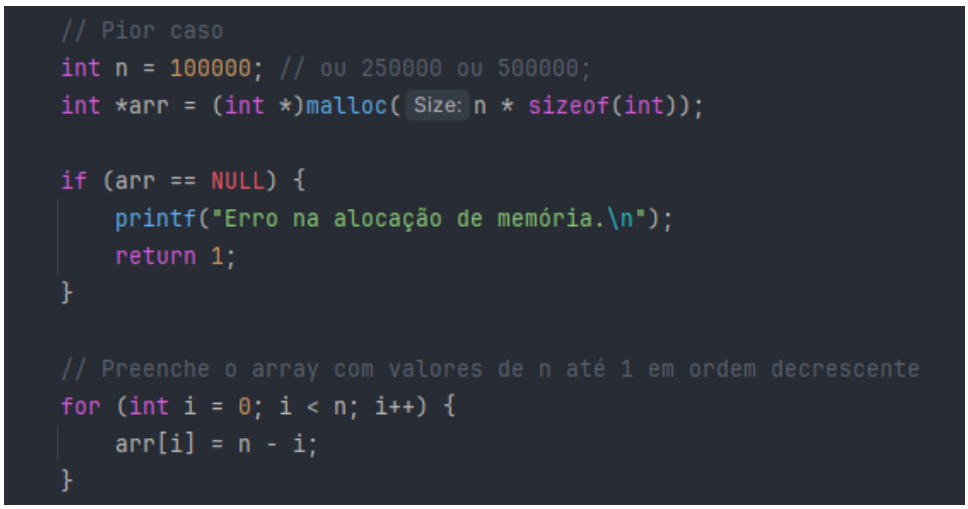


Figura 4-Teste Pior Caso.



```c
// Pior caso
int n = 100000; // ou 250000 ou 500000;
int *arr = (int *)malloc( Size: n * sizeof(int));

if (arr == NULL) {
    printf("Erro na alocação de memória.\n");
    return 1;
}

// Preenche o array com valores de n até 1 em ordem decrescente
for (int i = 0; i < n; i++) {
    arr[i] = n - i;
}
```

4. RESULTADOS E DISCUSSÃO

Os dados brutos dos testes que deram origem as tabelas que serão apresentadas a seguir estão no anexo I. Nas Tabelas 1, 2 e 3 abaixo, é possível visualizar os dados comparativos de cada algoritmo de ordenação para o array do tamanho de 100.000 elementos, analisando o melhor, médio e pior caso respectivamente. No melhor caso o algoritmo que teve o menor tempo de execução se mostrando assim mais eficiente foi o Insertion sort e o de pior desempenho o Selection sort.

Esse resultado corrobora a característica do Insertion Sort de ser mais eficiente com conjunto menores de dados e quando os dados no array estão ordenados. No caso o Tim Sort(Ω(n)) teve um tempo de execução próximo ao Insertion Sort (Ω(n)), nesse caso também poderia ser escolhido para organizar pequenas quantidades de dados como no caso apresentado.

Tabela 1

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Best Case in Arr[100000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,010333	0,012340	0,001600	9,515400	0,016600	0,007000
Desvio Padrão	0,004041	0,001604	0,000894	0,007925	0,004722	0,000707

Para o Médio caso (tabela-2) os resultados demonstram que o Quick Sort (O(n^2)) apresentou maior rapidez seguido pelo Tim Sort (O (n log(n)), nesse caso para o tamanho de dados apresentado o de pior desempenho foi do Selecion Sort (O(n^2)) com uma média de tempo de 9.518800 segundos de execução para o conjunto de dados testado.

Tabela-2

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Average Case in Arr[100000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,013800	0,012052	4,568000	9,518800	0,016400	0,012800
Desvio Padrão	0,000837	0,002292	0,081009	0,006181	0,001817	0,000837

Como resultado no Pior caso (tabela-3) para valores de array na ordem decrescente a análise mostra que o Tim Sort (θ (n log(n)) demonstrou maior rapidez (0.009400 segundos) seguido pelo Merge Sort (θ (n log(n)), nesse caso o Selection Sort (θ (n^2) foi o de pior desempenho com uma média de tempo de 9.881360 segundos de execução para o conjunto de dados seguido do Insertion Sort. Esses resultados vão de encontro com a característica do Insertion e do Selection de apresentar baixo desempenho quando os dados estão em ordem decrescente.

Tabela-3

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Worst Case in Arr[100000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,009500	0,012760	9,068600	9,881360	0,013800	0,009400
Desvio Padrão	0,003697	0,001031	0,123292	0,012048	0,000837	0,000548

Nas Tabelas 4, 5 e 6 abaixo, é possível visualizar os dados comparativos de cada algoritmo de ordenação para o array do tamanho de 250.000 elementos, analisando o melhor, médio e pior caso respectivamente. Para esse tamanho de array e considerando o melhor caso o algoritmo que apresentou melhor desempenho em tempo de execução foi o Insertion Sort (Ω(n)) seguido do Tim Sort (Ω(n)), nesse caso o de pior desempenho foi o Selection Sort (Ω (n^2) levando um tempo consideravelmente alto.

Tabela-4

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Average Case in Arr[250000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,038400	0,029800	29,573980	59,394386	0,050400	0,033200
Desvio Padrão	0,004278	0,004438	0,369217	0,347106	0,008264	0,004970

Ainda para o array de 250.000 elementos, porém para o médio caso (tabela 5) o algoritmo que teve o melhor desempenho foi o Quick Sort (O(n^2)) (0,029800 segundos) corroborando sua característica de ser muito rápido e eficiente, enquanto o de pior desempenho levando em torno de 59,394386 segundos foi o Selection Sort (O(n^2)), resultando em um tempo muito acima da média dos demais, nesse teste o Insertion Sort (O(n^2)) também apresentou baixo de desempenho levando um tempo de 29,573980 segundos para concluir a ordenação dos dados.

Tabela-5

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Average Case in Arr[250000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,038400	0,029800	29,573980	59,394386	0,050400	0,033200
Desvio Padrão	0,004278	0,004438	0,369217	0,347106	0,008264	0,004970

No pior caso para o array em discussão na tabela 6, pode ser observado que houve pouca diferença no tempo de execução entre os algoritmos que se saíram melhor, no caso Merge Sort, Tim Sort e Quick Sort apresentando 0.019600 no Caso do Merge (θ n log(n)) que foi o mais rápido. Novamente para esse caso o Selection (θ (n^2)) e o Insertion (θ (n^2)) tiveram os piores desempenhos com um tempo de execução muito elevado em comparação com os demais.

Tabela –6

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Worst Case in Arr[250000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,019600	0,029800	60,498880	61,155198	0,040200	0,022200
Desvio Padrão	0,002702	0,003004	0,413955	0,620158	0,004764	0,004087

Nas Tabelas 7, 8 e 9 abaixo, é possível visualizar os dados comparativos de cada algoritmo de ordenação para o array do tamanho de 500.000 elementos, analisando o melhor, médio e pior caso respectivamente. Na tabela 7, considerando o melhor caso, o algoritmo que apresentou melhor desempenho em tempo de execução foi o Insertion Sort (Ω(n)) seguido do Tim Sort (Ω(n)), nesse caso o de pior desempenho foi o Selection Sort (θ (n^2)) levando um tempo consideravelmente alto em relação aos demais batendo 237,844840 segundos.

Tabela-7

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Best Case in Arr[500000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,049000	0,061000	0,002480	237,844840	0,077000	0,040200
Desvio Padrão	0,004183	0,002739	0,000672	0,586627	0,001000	0,004207

No **médio caso** (tabela 8) é notório a diferença do tempo de execução entre os algoritmos com melhor e pior desempenho. O Merge Sort (O n log(n)) foi o algoritmo com melhor desempenho executando 500.000 elementos em 0,044200 segundos, já o Insertion Sort (O (n^2)) com 124,949400 segundos e o Selection Sort (O (n^2)) com 237,366600 segundos tiveram o pior desempenho.

Tabela-8

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Average Case in Arr[500000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,044200	0,062000	124,949400	237,366600	0,117200	0,073400
Desvio Padrão	0,005263	0,005523	1,497923	1,392686	0,009985	0,006348

No **pior caso** (tabela 9), assim como no médio caso, o Merge Sort (θ n log(n)) teve o melhor desempenho com 0,046800 segundos, o Insertion Sort (θ (n^2)) e Selection Sort (θ (n^2)) os piores desempenhos 253,788600 segundos e 246,570480 segundos respectivamente.

Tabela-9

Compilador		Máquina(CPU)			Mem RAM	
CLion		Ryzen 7 4800h			24 GB	
Sorting Algorithms						
Tabela de comparação dos tempos em segundos						
Best Case in Arr[250000]						
Stats	Merge Sort	Quick Sort	Insertion Sort	Selection Sort	Heap Sort	Tim Sort
Média	0,019200	0,029300	0,000940	59,325760	0,039800	0,014800
Desvio Padrão	0,003421	0,002898	0,000096	0,156456	0,004438	0,002168

5. CONCLUSÃO

Após analisar os resultados de cada teste, fica claro que cada algoritmo de ordenação de dados possui suas próprias características e particularidades. Essas diferenças podem variar de um computador para outro. Durante os testes, optou-se por criar um array dinâmico na memória heap, evitando assim erros comuns como o "exit code -1073741571 (0xC00000FD)" conhecido como "stack overflow". Cada método de organização de dados demonstra sua eficácia e complexidade em diferentes conjuntos de dados, variando também com o tamanho desses conjuntos, sejam pequenos, médios ou grandes. Isso implica em diferentes cenários de desempenho, seja no melhor, médio ou pior caso, que podem variar conforme o método utilizado e o tamanho do conjunto de dados. É importante ressaltar que este trabalho é de cunho acadêmico, não se tratando de uma pesquisa definitiva, mas sim de uma análise baseada em dados e testes realizados para essa finalidade.

6. REFERENCIAS

Tutorialspoint. C Library. Disponível Programiz. Merge Sort Algorithm. Disponível em: <https://www.programiz.com/dsa/merge-sort\>.

Programiz. Quick Sort Algorithm. Disponível em: <https://www.programiz.com/dsa/quick-sort\>.

Programiz. Insertion Sort Algorithm. Disponível em: <https://www.programiz.com/dsa/insertion-sort\>.

Programiz. Selection Sort Algorithm. Disponível em: <https://www.programiz.com/dsa/selection-sort\>.

Programiz. Heap Sort Algorithm. Disponível em:

<https://www.programiz.com/dsa/heap-sort\>.

GeeksforGeeks. Tim Sort Algorithm. Disponível em:
<https://www.geeksforgeeks.org/timsort/\>.

Programiz. Sorting Algorithm. Disponível em:
<https://www.programiz.com/dsa/sorting-algorithm\>.

CORMEN, Thomas H. et al. Introduction to Algorithms. [S.l.]: MIT Press, 2009.

SEDGEWICK, Robert; WAYNE, Kevin. Algorithms. [S.l.]: Addison-Wesley, 2011.

ANEXO I

TABELA DETALHANDO A COMPLEXIDADE DE CADA ALGORITMO DE ORDENAÇÃO

Sorting Algorithm	Time Complexity			Space Complexity	Stability
	Best	Worst	Average		
Merge Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)	Yes
Quick Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)	No
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)	Yes
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)	No
Heap Sort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)	No
Tim Sort	Ω(n)	θ(n log(n))	O(n log (n))	O(n)	Yes

ANEXO II

TABELAS COM OS DADOS COMPLETOS DAS MEDIÇÕES DE TEMPO DE EXECUÇÃO DOS ALGORITMOS.

A. Array de 100000 elementos.

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Tim Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[100000]	0,007000	0,013000	0,009000
2	Arr[100000]	0,008000	0,014000	0,010000
3	Arr[100000]	0,007000	0,012000	0,009000
4	Arr[100000]	0,006000	0,013000	0,009000
5	Arr[100000]	0,007000	0,012000	0,010000
Média	Arr[100000]	0,007000	0,012800	0,009400
Desvio Padrão	Arr[100000]	0,000707	0,000837	0,000548

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Merge Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[100000]	0,015000	0,013000	0,007000
2	Arr[100000]	0,008000	0,014000	0,008000
3	Arr[100000]	0.009000	0,015000	0.009000
4	Arr[100000]	0,008000	0,013000	0,015000
5	Arr[100000]	0.009000	0,014000	0,008000
Média	Arr[100000]	0,010333	0,013800	0,009500
Desvio Padrão	Arr[100000]	0,004041	0,000837	0,003697

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Quick Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[100000]	0,012500	0,011000	0,012500
2	Arr[100000]	0,013700	0,015000	0,014200
3	Arr[100000]	0,010200	0,014000	0,011700
4	Arr[100000]	0,011300	0,010000	0,012000
5	Arr[100000]	0,014000	0,010260	0,013400
Média	Arr[100000]	0,012340	0,012052	0,012760
Desvio Padrão	Arr[100000]	0,001604	0,002292	0,001031

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Insertion Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[100000]	0,001000	4,481000	8,981000
2	Arr[100000]	0,003000	4,647000	9,193000
3	Arr[100000]	0,002000	4,484000	8,899000
4	Arr[100000]	0,001000	4,590000	9,141000
5	Arr[100000]	0,001000	4,638000	9,129000
Média	Arr[100000]	0,001600	4,568000	9,068600
Desvio Padrão	Arr[100000]	0,000894	0,081009	0,123292

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Selection Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[100000]	9,518000	9,522000	9,870000
2	Arr[100000]	9,518000	9,517000	9,877000
3	Arr[100000]	9,502000	9,521000	9,872000
4	Arr[100000]	9,516000	9,509000	9,890000
5	Arr[100000]	9,523000	9,525000	9,897800
Média	Arr[100000]	9,515400	9,518800	9,881360
Desvio Padrão	Arr[100000]	0,007925	0,006181	0,012048

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Heap Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[100000]	0,014000	0,018000	0,013000
2	Arr[100000]	0,025000	0,014000	0,014000
3	Arr[100000]	0,015000	0,015000	0,014000
4	Arr[100000]	0,014000	0,018000	0,015000
5	Arr[100000]	0,015000	0,017000	0,013000
Média	Arr[100000]	0,016600	0,016400	0,013800
Desvio Padrão	Arr[100000]	0,004722	0,001817	0,000837

B. Array de 250000 elementos.

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Tim Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[250000]	0,014000	0,036000	0,024000
2	Arr[250000]	0,012000	0,031000	0,015000
3	Arr[250000]	0,018000	0,026000	0,023000
4	Arr[250000]	0,015000	0,034000	0,025000
5	Arr[250000]	0,015000	0,039000	0,024000
Média	Arr[250000]	0,014800	0,033200	0,022200
Desvio Padrão	Arr[250000]	0,002168	0,004970	0,004087

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Merge Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[250000]	0,016000	0,046000	0,020000
2	Arr[250000]	0,022000	0,036000	0,020000
3	Arr[250000]	0,022000	0,037000	0,022000
4	Arr[250000]	0,021000	0,036000	0,015000
5	Arr[250000]	0,015000	0,037000	0,021000
Média	Arr[250000]	0,019200	0,038400	0,019600
Desvio Padrão	Arr[250000]	0,003421	0,004278	0,002702

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Quick Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[250000]	0,033500	0,037000	0,033500
2	Arr[250000]	0,025800	0,026000	0,029400
3	Arr[250000]	0,029400	0,031000	0,025700
4	Arr[250000]	0,030200	0,028000	0,031800
5	Arr[250000]	0,027600	0,027000	0,028600
Média	Arr[250000]	0,029300	0,029800	0,029800
Desvio Padrão	Arr[250000]	0,002898	0,004438	0,003004

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Insertion Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[250000]	0,001000	29,744000	60,892000
2	Arr[250000]	0,001000	29,726500	60,003400
3	Arr[250000]	0,000780	29,000400	60,650000
4	Arr[250000]	0,000920	29,957000	60,836000
5	Arr[250000]	0,001000	29,442000	60,113000
Média	Arr[250000]	0,000940	29,573980	60,498880
Desvio Padrão	Arr[250000]	0,000096	0,369217	0,413955

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Selection Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[250000]	59,454000	59,496000	61,769000
2	Arr[250000]	59,356600	59,001900	61,727000
3	Arr[250000]	59,446540	59,205400	60,266000
4	Arr[250000]	59,070140	59,345600	61,013000
5	Arr[250000]	59,301520	59,923030	61,000990
Média	Arr[250000]	59,325760	59,394386	61,155198
Desvio Padrão	Arr[250000]	0,156456	0,347106	0,620158

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Heap Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[250000]	0,041000	0,051000	0,048000
2	Arr[250000]	0,038000	0,063000	0,039000
3	Arr[250000]	0,037000	0,044000	0,041000
4	Arr[250000]	0,047000	0,052000	0,036000
5	Arr[250000]	0,036000	0,042000	0,037000
Média	Arr[250000]	0,039800	0,050400	0,040200
Desvio Padrão	Arr[250000]	0,004438	0,008264	0,004764

C. Array de 500000 elementos.

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Heap Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[500000]	0,078000	0,127000	0,078000
2	Arr[500000]	0,077000	0,112000	0,077000
3	Arr[500000]	0,087000	0,105000	0,076000
4	Arr[500000]	0,078000	0,128000	0,076000
5	Arr[500000]	0,086000	0,114000	0,078000
Média	Arr[500000]	0,077000	0,117200	0,077000
Desvio Padrão	Arr[500000]	0,001000	0,009985	0,001000

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Tim Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[500000]	0,039000	0,073000	0,048000
2	Arr[500000]	0,038000	0,063000	0,058000
3	Arr[500000]	0,041000	0,074000	0,048000
4	Arr[500000]	0,047000	0,079000	0,049000
5	Arr[500000]	0,036000	0,078000	0,053000
Média	Arr[500000]	0,040200	0,073400	0,051200
Desvio Padrão	Arr[500000]	0,004207	0,006348	0,004324

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Merge Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[500000]	0,054000	0,047000	0,045000
2	Arr[500000]	0,047000	0,050000	0,044000
3	Arr[500000]	0,046000	0,045000	0,052000
4	Arr[500000]	0,045000	0,036000	0,046000
5	Arr[500000]	0,053000	0,043000	0,047000
Média	Arr[500000]	0,049000	0,044200	0,046800
Desvio Padrão	Arr[500000]	0,004183	0,005263	0,003114

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Quick Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[500000]	0,058000	0,053000	0,058200
2	Arr[500000]	0,061000	0,062000	0,060500
3	Arr[500000]	0,065000	0,068000	0,065300
4	Arr[500000]	0,059000	0,063000	0,056800
5	Arr[500000]	0,062000	0,064000	0,062700
Média	Arr[500000]	0,061000	0,062000	0,060700
Desvio Padrão	Arr[500000]	0,002739	0,005523	0,003415

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Insertion Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[500000]	0,002000	126,874000	253,040000
2	Arr[500000]	0,002000	123,412000	254,650000
3	Arr[500000]	0,003000	124,561000	253,934000
4	Arr[500000]	0,002000	123,776000	252,331000
5	Arr[500000]	0,003400	126,124000	254,988000
Média	Arr[500000]	0,002480	124,949400	253,788600
Desvio Padrão	Arr[500000]	0,000672	1,497923	1,105295

Compilador		Máquina(CPU)		Mem RAM
CLion		Ryzen 7 4800h		24 GB
Selection Sort				
Tabela de comparação dos tempos em segundos				
Teste	Array Range	Best	Average	Worst
1	Arr[500000]	237,251000	237,803000	247,015000
2	Arr[500000]	237,560100	238,760000	247,004700
3	Arr[500000]	237,871100	237,431000	247,200700
4	Arr[500000]	237,733000	235,036000	246,358000
5	Arr[500000]	238,809000	237,803000	245,274000
Média	Arr[500000]	237,844840	237,366600	246,570480
Desvio Padrão	Arr[500000]	0,586627	1,392686	0,792043