Trabalho Prático 2
A ordenação estratégia de dominação do Imperador

Jean Lucas Almeida Mota

Universidade Federal de Minas Gerais (UFMG)
Belo Horizonte - MG - Brasil

jeanlk@ufmg.br

1. Introdução

O problema a ser enfrentado possui natureza fictícia com base no universo cinematográfico paralelo da saga Star Wars. Neste problema deve ser implementado 4 sistemas independentes entre si, que ordenam, usando métodos distintos, um conjunto de civilizações. A ordenação dirá qual será invadida primeiro pelo Imperador. As civilizações possuem: distância e número de habitantes. E o número máximo de civilizações é 2000000 (2 milhões).

A ordenação deve obedecer aos seguintes critérios:

1. A civilização com a menor distância será invadida primeiro.
2. Se as distâncias forem equivalentes, a civilização com maior número de habitantes será invadida primeiro.

Dois dos sistemas serão usados pelo Imperador e devem usar métodos de ordenação simples. E os outros dois que serão usados pela aliança rebelde, devem usar métodos de ordenação eficientes. Após a implementação, deve ser feita uma análise experimental de todos os sistemas, comparando-os.

2. Implementação

O programa foi desenvolvido na linguagem C++, compilada pelo compilador G++.

2.1 Estrutura de dados

Os sistemas foram divididos em 4 pastas distintas:

src_ordenacao_imperador_algoritmo1: Representa o sistema que fará uso do método de ordenação simples Seleção.

src_ordenacao_imperador_algoritmo2: Representa o sistema que fará uso do método de ordenação simples Inserção.

src_ordenacao_alianca_rebelde_algoritmo1: Representa o sistema que fará uso do método de ordenação eficiente Quicksort.

src_ordenacao_alianca_rebelde_algoritmo2: Representa o sistema que fará uso do método de ordenação eficiente Heapsort.

A implementação usada em cada método de ordenação mencionado segue a implementação vista nas aulas da disciplina DCC205-Estrutura de Dados. Porém alterações foram feitas para os casos onde as distâncias se equivalem.

2.2 Classes

A fim de modularizar a implementação e facilitar a depuração de erros, os 4 sistemas foram modularizados de forma equivalente, se diferenciando apenas pelo método de ordenação.

Classe *Civilização* : Representa o objeto civilização. Possui os atributos: *planeta*, *distancia* e *população* que representam respectivamente, o nome da civilização, a distancia e o número de habitantes. O métodos implementados são: *GetPlaneta()*, *GetDistancia*, *GetPopulação*, que retornam seus respectivos atributos, e há também o método *imprime()* que faz a impressão dos atributos do objeto na tela.

Classe *Civilizacao* : É a classe de mais alto nível, ela que faz a manipulação das civilizações e as ordena. Possui os atributos: *num_civilizacoes* e *civilizações*, que representação respectivamente, o número de total de civilizações a se analisar e a lista de todas as civilizações a se ordenar. Para armazenar essa lista de civilizações foi utilizada a estrutura *<vector>*, sua escolha foi dada pela facilidade de implementação e diminuição de possíveis erros de memória. A classe *Civilizacao* possui os métodos: *AdicionaCivilizacao(planeta, distancia, população)* que recebe os atributos de uma civilização, cria o objeto *civilização* e o adiciona na lista de civilizações para posterior ordenação. O método *ImprimeCivilizacoes()*

faz a impressão de todos as civilizações adicionadas na lista. E por fim o método *Ordena()*, que é o método que de fato diferencia cada sistema, nele é implementado o método de ordenação escolhido para aquele sistema. Dependendo do método de ordenação envolvido, há métodos auxiliares, como por exemplo, para *src_ordenacao_alianca_rebelde_algoritmo2*, que utiliza o método Heapsort, há os métodos auxiliares *Constroi()* e *Refaz(esq,dir)*.

Observação: Foi desenvolvido no *main.cpp* uma forma de escrever os tempos de ordenação de cada um dos testes em um arquivo chamado *tempo.txt*. A cada execução o *main.cpp* adiciona um novo tempo nesse arquivo.

3. Análise de Complexidade

3.1 Tempo

Em todos os 4 sistemas propostos neste trabalho temos o mesmo custo de inicialização. Ao inicializar recebe-se o número de civilizações, custo O(1). E em seguida recebe-se todas as civilizações, que são inseridas em uma lista, custo O(n). Dessa forma o custo de inicialização é O(n+1).

Agora devemos analisar o custo de ordenação de cada um dos sistemas:

- **src_ordenacao_imperador_algoritmo1**: Usa o método de Seleção.
  - Comparações: $O(n^2)$
- **src_ordenacao_imperador_algoritmo2**: Usa o método de Inserção.
  - Comparações:
    - Melhor caso: O(n)
    - Pior caso: $O(n^2)$
- **src_ordenacao_alianca_rebelde_algoritmo1**: Usa o método Quicksort.
  - Comparações:
    - Melhor caso: O(n logn)
    - Caso médio: O(n logn)
    - Pior caso: $O(n^2)$
- **src_ordenacao_alianca_rebelde_algoritmo2**: Usa o método Heapsort.
  - Comparações: O(n logn)

3.2 Espaço

Em todos os sistemas a forma de guardar essas civilizações é a mesma, uma lista dinâmica (*<vector>)*. Ou seja, uma vez que não há a criação de estruturas

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

auxiliares para fazer alterações nos vetores em nenhum dos métodos de ordenação utilizados, temos que o custo de espaço é o mesmo para todos.

Vamos considerar, para nossa análise, que cada civilização ocupe uma unidade de espaço Dessa forma, temos que o custo de espaço depende apenas da quantidade de civilizações, ou seja, O(n).

4 Análise Experimental

A análise experimental do tempo de ordenação dos sistemas foi feita usando a biblioteca <*time*>[1], os testes utilizados foram os disponibilizados pelo professor e os gráficos foram feitos no canvas[2].

Número do teste	Número de civilizações
Teste 0	50
Teste 1	100
Teste 2	500
Teste 3	1000
Teste 4	10000
Teste 5	100000
Teste 6	250000
Teste 7	500000
Teste 8	1000000
Teste 9	2000000

4.1 Análise src_ordenacao_imperador_algoritmo1:

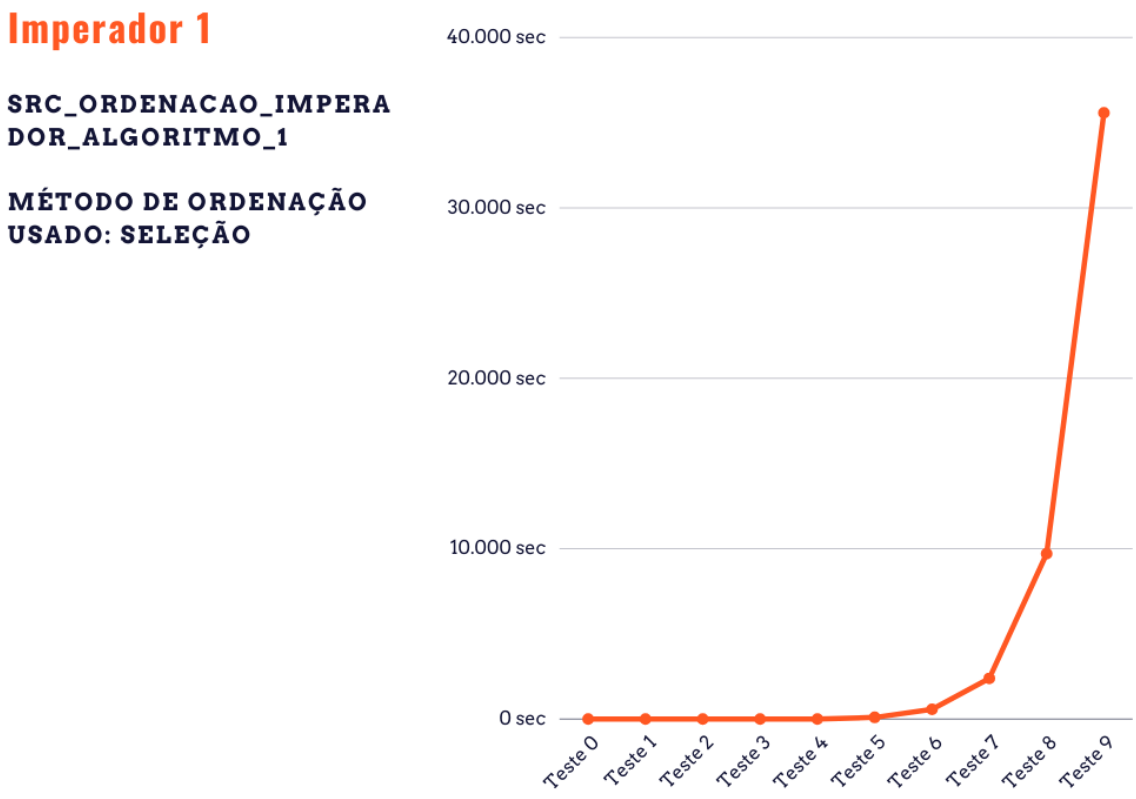
Número do teste	Tempo
Teste 0	0,000 segundos
Teste 1	0,000 segundos
Teste 2	0,002 segundos
Teste 3	0,009 segundos
Teste 4	0,907 segundos
Teste 5	91,56 segundos
Teste 6	570,76 segundos
Teste 7	2385,2 segundos

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

Teste 8	9699,5 segundos
Teste 9	35572.553 segundos

Tempo de ordenação máximo (2000000 civilizações): 35572.553 segundos ou 9 horas e 52 minutos.

No gráfico podemos verificar com mais facilidade a sua complexidade assintótica $O(n^2)$.



4.2 Análise src_ordenacao_imperador_algoritmo2:

Número do teste	Tempo
Teste 0	0.000 segundos
Teste 1	0.000 segundos
Teste 2	0.001 segundos
Teste 3	0.003 segundos
Teste 4	0.308 segundos
Teste 5	32.1 segundos
Teste 6	196.1 segundos

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

Teste 7	790.6 segundos
Teste 8	3125.2 segundos
Teste 9	12208.8 segundos

Tempo de ordenação máximo (2000000 civilizações): 12208,8 segundos ou 3 horas e 23 minutos.

Por mais que os testes mostrem que este método (Inserção) é mais rápido que o método anterior (Seleção), à medida que os testes se tornam assintóticos, o Imperador 2 se torna tão custoso em termos de tempo quanto o Imperador 1.

4.3 Análise src_ordenacao_alianca_rebelde_algoritmo1:

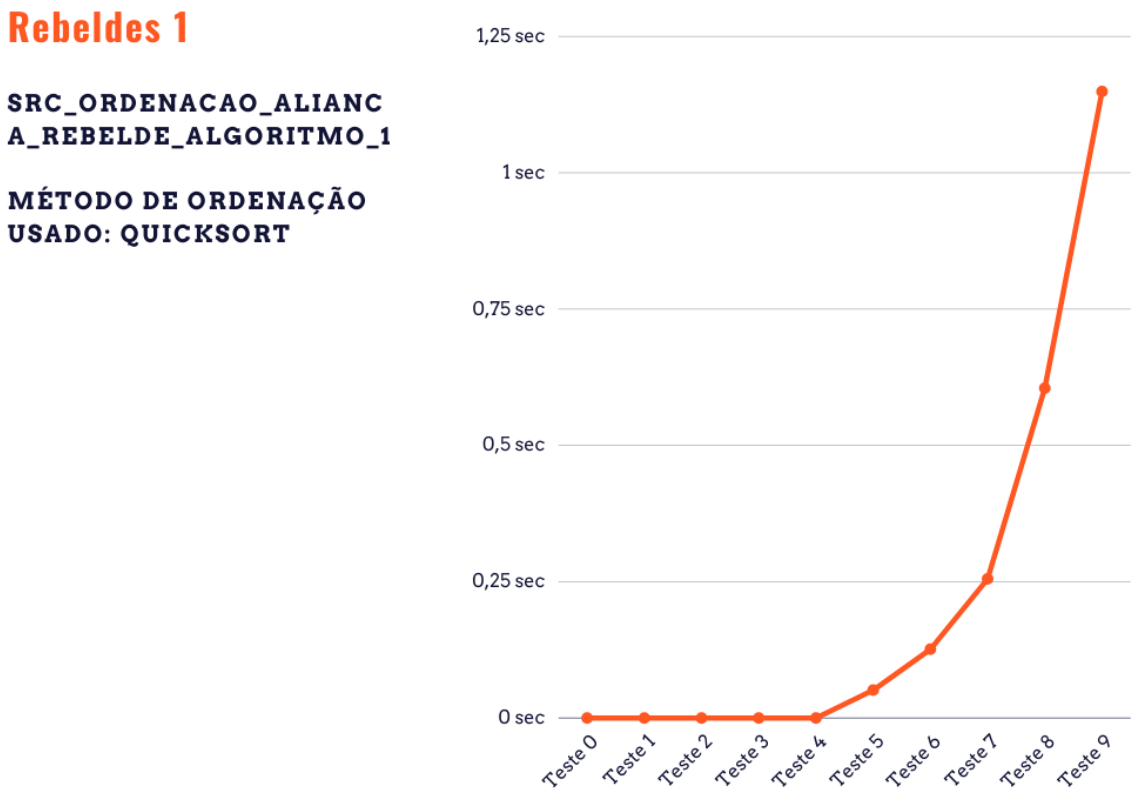
Número do teste	Tempo
Teste 0	0.000000 segundos
Teste 1	0.000000 segundos
Teste 2	0.000000 segundos
Teste 3	0.000000 segundos

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

Teste 4	0.000000 segundos
Teste 5	0.051000 segundos
Teste 6	0.126000 segundos
Teste 7	0.255000 segundos
Teste 8	0.605000 segundos
Teste 9	1.149000 segundos

Tempo de ordenação máximo (2000000 civilizações): 1.149 segundos.

Neste método observamos uma diferença enorme em relação aos métodos anteriores. Sem dúvida é um método eficiente. Dessa forma, fica evidente o ganho que a sua complexidade O(n logn) traz em relação aos métodos com complexidade O(n^2). E fica evidenciado que, por mais que o Quicksort tenha um pior caso O(n^2), ele realmente tem uma chance muito baixa que acontecer.



4.4 Análise src_ordenacao_alianca_rebelde_algoritmo2:

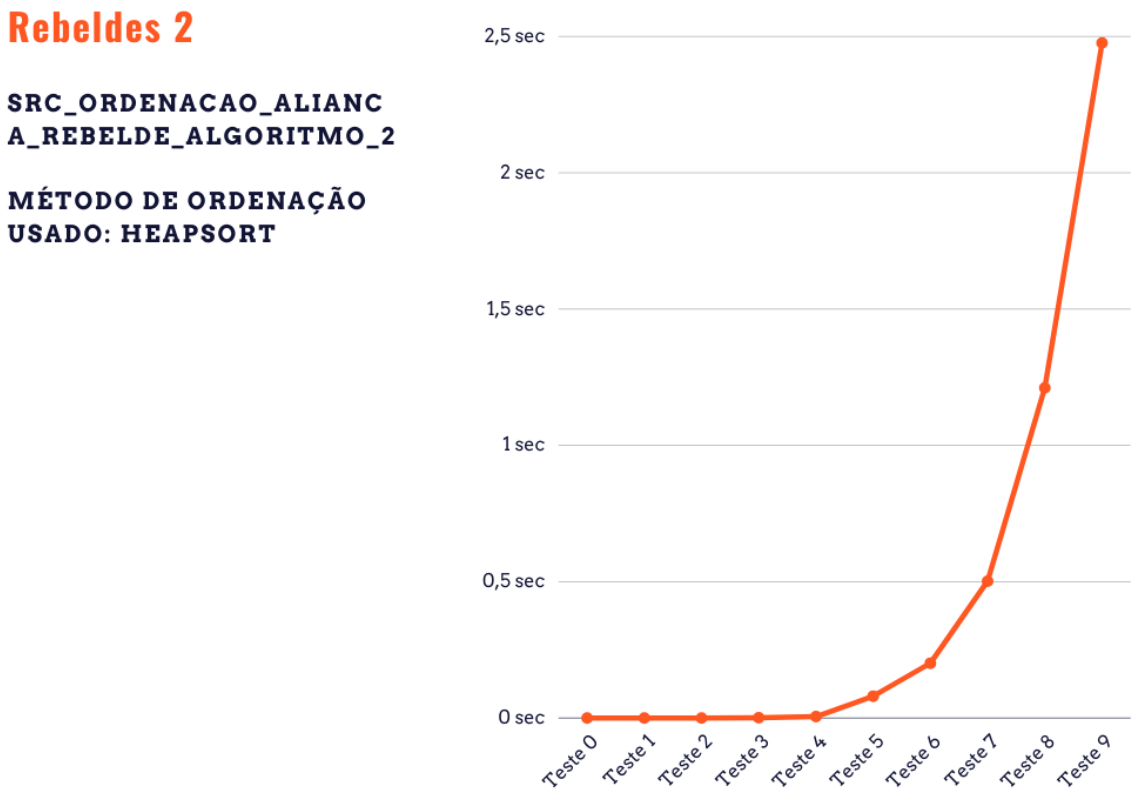
Número do teste	Tempo
Teste 0	0.000000 segundos

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

Teste 1	0.000000 segundos
Teste 2	0.000000 segundos
Teste 3	0.001000 segundos
Teste 4	0.005000 segundos
Teste 5	0.080000 segundos
Teste 6	0.201000 segundos
Teste 7	0.502000 segundos
Teste 8	1.211000 segundos
Teste 9	2.476000 segundos

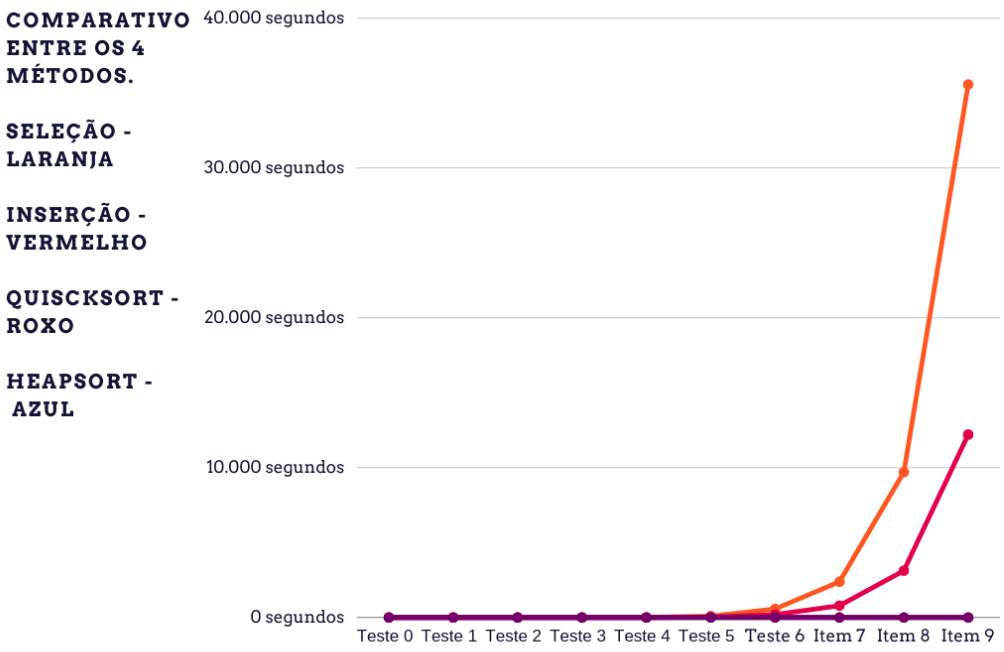
Tempo de ordenação máximo (2000000 civilizações): 2.476 segundos.

Neste método observamos uma diferença enorme em relação aos métodos simples (Seleção e Inserção). Sem dúvida é um método eficiente. No entanto, em relação ao Quicksort seu desempenho é um pouco inferior.



[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

4.5 Comparativo



É notável pelo gráfico que os métodos simples crescem tão rápido que a visualização dos métodos eficientes fica muito difícil. O método Heapsort fica tão semelhante ao Quicksort que sua visualização é impossível. Dessa forma, esse tipo de gráfico nos da uma ideia ainda melhor de qual métodos devemos escolher para a aliança rebelde e para o Imperador.

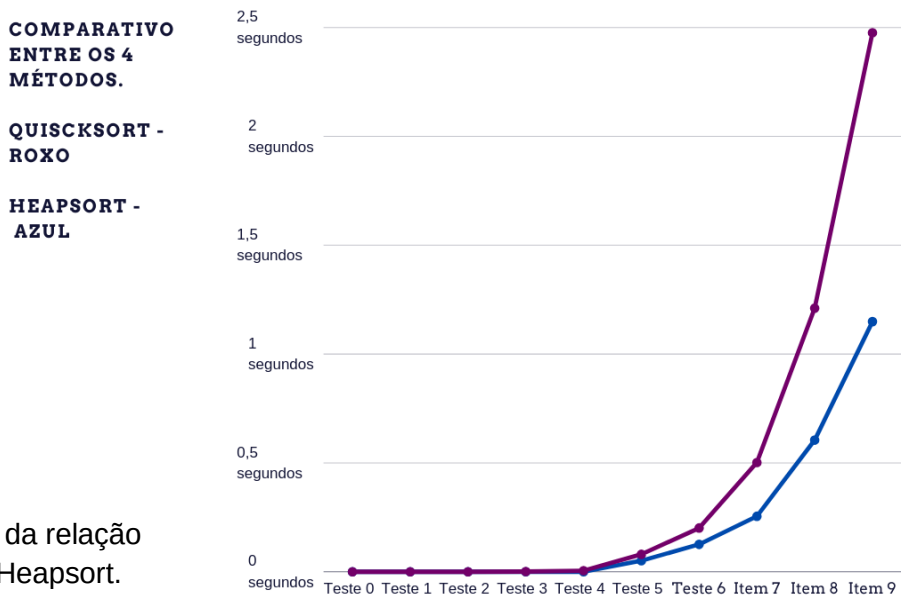


Imagem ampliada da relação entre Quicksort e Heapsort.

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

5. Conclusão

Após a leitura do problema proposto, nota-se que, os 4 sistemas propostos são quase iguais, se diferenciando apenas pela função ordenação. O grande esforço do problema proposto está, sobretudo, na implementação dos códigos de ordenação. Uma vez que alterações devem ser feitas para que o método de ordenação leve em conta o caso em que as distâncias se equivalem. Muitos erros lógicos podem aparecer durante essa implementação.

Ao finalizar a implementação e iniciar a análise experimental, podemos verificar na prática a teoria estudada em aula. A diferença entre os métodos simples e os eficientes é gritante quando o número de entradas aumenta muito. Dessa forma, a fim de maximizar as chances da aliança rebelde, para que ela tenha a ordem de invasão antes do Imperador, a aliança deve usar o *src_ordenacao_alianca_rebelde_algoritmo2* e o Imperador deve usar o *src_ordenacao_imperador_algoritmo1*.

6. Execução

O programa foi desenvolvido no Windows 10 e testado no Windows 10 e no Linux.

Para execução no Windows foi executado o seguinte comando:
mingw32-make
tp2
Para execução no Linux foi executado o seguinte comando:
make
./tp2
Observação: Os testes deixados pelo professor são executados somente no Linux, através do comando:
make test
Observação2: Os arquivos de testes não estão sendo enviados por causa da quantidade de memória que os mesmos utilizam, impossibilitando seu envio pelo moodle.

References

Chaimowicz, Luiz e Prates, Raquel. Slide sobre Seleção, Inserção, Quicksort e Heapsort (2020). Disciplina Estrutura de Dados DCC205 UFMG.

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/

Post sobre como adicionar informações em um .txt. Disponível em <https://pt.stackoverflow.com/questions/33032/quero-gravar-as-informa%C3%A7%C3%B5es-em-um-txt-em-c> Publicado em 18/09/2014. Acessado em 08/10/2020.

[1]https://www.geeksforgeeks.org/measure-execution-time-with-high-precision-in-c-c/
[2]https://www.canva.com/pt_br/graficos/grafico-barras/