A Fintech

André Luiz Rodrigues,* Bruno Ramos†

Escola Politécnica — PUCRS

15 de setembro de 2021

Resumo

Este trabalho tem como objetivo descrever as soluções encontradas para o problema proposto pelo Prof. Marcelo Cohen na disciplina de Algoritmos e Estrutura de Dados II. O objetivo do problema apresentado consiste em encontrar a melhor combinação das ordens de compra e venda das ações da empresa HAL Corporation, de modo que a fintech, responsável por intermediar essas operações, possa lucrar o máximo possível com o spread dos valores das ordens enviadas. Para a resolução do problema, trabalhou-se com a linguagem Java.

Introdução

Com o intuito de compreender as soluções retratadas neste artigo, serão descritas as informações e especificações disponibilizadas no enunciado do problema.

1. Uma ordem de compra é dada por *C <quant> <preço>* onde a quantidade e o preço são inteiros e o preço significa que o comprador deseja comprar cada uma das ações pelo preço informado ou menor;
2. Uma ordem de venda é dada por *V <quant> <preço>* onde a quantidade e o preço são inteiros e o preço significa que o comprador deseja vender cada uma das ações pelo preço informado ou maior;[1]
3. A *fintech* lucra com a diferença entre preços (*spread*). Por exemplo, quando um vendedor quer vender três ações por $10 ou mais cada uma e um comprador quer comprar duas ações por $12 ou menos cada uma, temos um *match* entre as ordens e a *fintech* realiza a transação e fica com a diferença de valor, lucrando $2 em cada uma das duas ações que são vendidas, ou seja, $4 no total. Note que ainda ficou pendente a ordem de venda de uma ação por $10 ou mais, que ficará esperando

[1]*andre.rodrigues99@edu.pucrs.br
†bruno.ramos93@edu.pucrs.br

novas ordens de compra compatíveis com o preço pedido;

4. A cada nova operação que chega, todos os negócios que podem ser feitos naquele momento são realizados.

Como o objetivo do programa é encontrar a combinação de ordens de compra e venda que gere mais lucros à *fintech*, a diferença entre preços de compra e venda sempre deve ser a maior possível. Serão utilizados os arquivos de entrada disponibilizados para fazer os testes com o algoritmo desenvolvido, e ao final de cada simulação, será informado:

1. O lucro da empresa na simulação;
2. A quantidade de ações negociadas;
3. Quantas ordens de compra ficaram pendentes; e
4. Quantas ordens de venda ficaram pendentes;

```
10
V 104 1674
C 130 424
V 69 1972
V 159 1452
C 150 343
C 174 160
V 135 1929
V 191 722
V 94 735
C 141 1506
```

Figura 1: **Exemplo de arquivo de entrada.**

Solução

A solução proposta para o problema consiste, inicialmente, em interpretar o arquivo de entrada. A primeira linha informa o número de ordens que serão enviadas ao programa, já as demais linhas, correspondem às ordens de compra e venda, com a primeira coluna correspondendo ao tipo de ordem, a segunda coluna informando a quantidade de ações que serão vendidas ou compradas. Já a terceira coluna representa a que preço será executada a ordem, podendo ser maior ou menor que o preço de referência, dependendo do tipo de ordem a ser executada (compra ou venda).

Para armazenar e tornar útil as informações do arquivo, bem como estabelecer os atributos e métodos que serão armazenados dentro de cada Ordem, foi criada uma classe *Ordem*. A classe *Ordem* possui um método *compareTo*, oriundo da interface *Comparable*, responsável por organizar as ordens de acordo com seu preço em ordem decrescente. Além disso, a classe possui os seguintes

atributos:

- Tipo de ordem;
- Quantidade de ações da ordem;
- Preço de cada ação;
- Quantidade de ações executadas / negociadas na ordem;

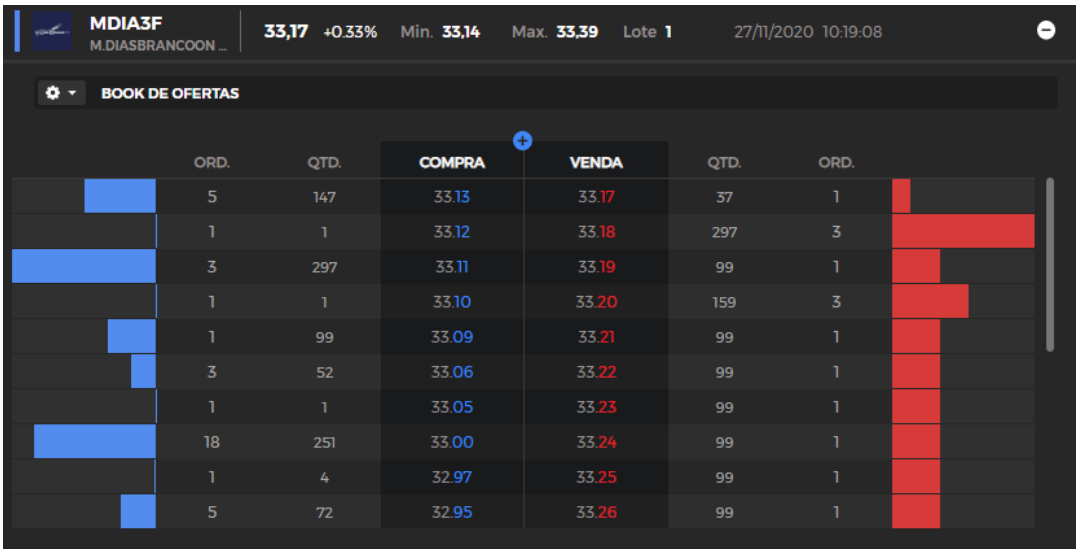
Após a interpretação das ordens presentes no arquivo processado, o próximo passo consiste em instanciar as ordens e armazená-las em uma lista inicial que terá todas as ordens do arquivo. Para que as ordens da lista sejam executadas, deve haver um *match* entre uma ordem de compra e uma ordem de venda, ou seja, a ordem de compra deve possuir o preço de negociação maior que o da ordem de venda.

Como o objetivo é maximizar os lucros da *fintech*, deve-se realizar o *match* apenas entre as ordens que possuem *spread* de preço, onde de preferência, ele seja o maior possível, ou seja, a diferença entre o preço de compra e venda deve ser a maior possível. Para isso, deve-se filtrar e ordenar a ordens instanciadas, criando uma nova lista de ordens de venda, e uma nova lista de ordens de compra. As ordens armazenadas nessas listas serão ordenadas de acordo com o preço de cada ordem, conforme abaixo:

- Ordens de venda: Ordenada em ordem crescente(do menor preço para o maior preço);
- Ordens de compra: Ordenada em ordem decrescente(do maior preço para o menor preço);

```
1.   procedimento OrganizarOrdens()
2.       Para cada elemento do ArrayList de ordens
3.           SE tipo de Ordem → V
4.               adiciona a ordem no ArrayList listaOrdensVenda
5.           Senao tipo de Ordem = C
6.               adiciona a ordem no ArrayList listaOrdensCompra
7.       Fim
8.           Ordena em ordem decrescente listaOrdensCompra
9.           Ordena em ordem crescente  listaOrdensVenda
10.      Fim
11.  Fim
```

Com as duas listas criadas e ordenadas, tem-se uma espécie de *book de ofertas* (figura 2), onde as ordens de compra foram ordenadas em ordem decrescente, conforme ilustrado na figura 3. O *book de ofertas* é o local onde o investidor consegue visualizar as ordens de compra e venda que estão em negociação no mercado de ações.



ORD.	QTD.	COMPRA	VENDA	QTD.	ORD.
5	147	33.13	33.17	37	1
1	1	33.12	33.18	297	3
3	297	33.11	33.19	99	1
1	1	33.10	33.20	159	3
1	99	33.09	33.21	99	1
3	52	33.06	33.22	99	1
1	1	33.05	33.23	99	1
18	251	33.00	33.24	99	1
1	4	32.97	33.25	99	1
5	72	32.95	33.26	99	1

Figura 2: **Book de ofertas da corretora Clear para ações da MDias Branco (MDIA3F) no mercado fracionário. Fonte: https://www.diariodeinvestimentos.com.br/book-de-ofertas/**

ORDEM DE COMPRA	
Qtd	Preço
199	1948
51	1924
55	1516
141	1506
161	1283
100	985
126	939
182	909
105	889
112	731
130	424
150	343
178	304
174	160

ORDEM DE VENDA	
Qtd	Preço
109	225
136	526
191	722
94	735
96	943
173	977
128	1073
161	1208
159	1452
81	1527
95	1532
57	1573
104	1674
171	1754

Figura 3: **Book de ofertas simplificado**

Com o *book de ofertas* formado, basta realizar o *match* entre as ordens que possuem o maior *spread* entre os preços até que se esgote as todas ordens de compra ou venda, ou até que não seja mais possível executar um *match* entre as ordens restantes, ou seja, quando as todas as ordens de compra possuem um preço de negociação menor do que os preços das ordens de venda.

Para que o *match* seja efetuado, seleciona-se a primeira ordem de compra(ordem que está disposta a pagar mais pela ação) através de uma estrutura de repetição *for each* na lista de ordens de compra, e tentamos dar *match* com a primeira ordem de venda(ordem que deseja vender as ações pelo menor valor), também através de uma estrutura de repetição *for each*, porém dentro da lista de ordens de venda. Para certificar que o *match* é válido, verifica-se se o preço da ordem de compra é maior que o preço da ordem de venda através da estrutura condicional *if else*, e caso a condição for verdadeira, a ordem é executada.

```
1.      procedimento Simular()
2.          Para cada elemento do ArrayList de ordens de Compra
3.              Verifica cada elemento do ArrayList de ordens de Venda
4.                  Se PrecoOrdemCompra >= PrecoOrdemVenda
5.                      lucroFintech ← lucroFintech + executaCompraVenda(ordemCompra ,
ordemVenda);
6.                  Fim
7.              Fim
8.          Fim
```

Durante a execução de cada ordem, será calculado o *spread* da negociação de cada ação, a fim de contabilizar o lucro da *fintech* ao final da simulação, bem como a quantidade de ações que foram negociadas em cada ordem.

```
1.      procedimento executaCompraVenda(Compra, Venda)
2.          Se QuantidadeOrdemCompra  >= QuantidadeOrdemVenda
3.              Faca
4.                  QuantidadeAcoesExecutada = QuantidadeOrdemVenda
5.                  spreadOperacao = (PrecoCompraAcao - PrecoVendaAcao) x
QuantidadeAcoesExecutada
6.                  QuantidadeOrdemVendaRestante = 0
7.                  QuantidadeOrdemCompraRestante = QuantidadeOrdemCompra -
QuantidadeAcoesExecutada
8.              Fim
9.          Senao Se(QuantidadeOrdemVenda >= QuantidadeOrdemCompra)
10.             Faca
11.                 QuantidadeAcoesExecutada = QuantidadeOrdemCompra;
12.                 spreadOperacao = (PrecoCompraAcao - PrecoVendaAcao) *
```

QuantidadeAcoesExecutada;

13. *QuantidadeOrdemCompraRestante = 0;*

14. *QuantidadeOrdemVendaRestante = QuantidadeOrdemVenda - QuantidadeAcoesExecutada;*

15. ***Fim***

//atualiza as quantidades de ações das ordens envolvidas na negociação

16. *QuantidadeAcoesCompra = QuantidadeOrdemCompraRestante*

17. *QuantidadeAcoesVenda = QuantidadeOrdemVendaRestante*

18. *QuantidadeAcoesJaExecutadasCompra = (QuantidadeCompraJaExecutadaCompra + QuantidadeAcoesExecutada)*

19. *QuantidadeAcoesJaExecutadasVenda = (QuantidadeAcoesJaExecutadasVenda + QuantidadeAcoesExecutada)*

20. ***Retorna*** *spreadOperacao*

21. ***Fim***

Caso haja ações remanescentes a serem executadas na ordem, o *match* das quantidades que ficaram pendentes serão executadas em conjunto com a próxima ordem da lista de compra ou venda a ser chamada pela estrutura de repetição.

Ao final da execução de todas as negociações possíveis, serão realizadas as seguintes operações:

- Contagem do total de ações que foram negociadas, através da contagem da quantidade de ações que foram executadas de um tipo de ordem(compra ou venda) utilizando a estrutura de repetição *for each*;
- Obtenção da quantidade de ordens de compra e venda que ficaram pendentes, verificando quais ordens, dentro de um laço *for each*, não estão com suas quantidades de ações zeradas.

Após isso, é informado o lucro total da fintech, o número de ações negociadas, e o número de ordens de compra e venda restantes.

1. ***procedimento*** *totalAcoesNegociadas()*

2. *QuantidadeNegociada = 0*

3. ***Para*** *cada elemento do ArrayList de ordens de Compra*

4. *QuantidadeNegociada = QuantidadeNegociada + getQtdAcoesExecutadas(ordem)*

5. ***Fim***

6. ***Retorna*** *QuantidadeNegociada*

Análise de Complexidade do Código

Com o algoritmo finalizado, podemos então analisar sua complexidade para sabermos o quão eficiente ele é. Analisando nosso algoritmo podemos afirmar que:

1. Se trata de um algoritmo iterativo;
2. Possui uma estrutura de repetição dentro de outra estrutura de repetição;
3. O método *sort* empregado para organizar as listas de ordens utiliza o algoritmo de ordenação *TimSort*, que possui notação assintótica Big-O do tipo O(n log n);
4. Possui um total de sete condicionais de verificação e duas situações de saída possíveis, uma quando todas as ordens de compra foram executadas, e outra quando o preço das ordens de venda restantes é maior que o preço das ordens de compra remanescentes;
5. O algoritmo percorre todas as ordens de ambas as listas de compra e venda a fim de verificar o total de ações negociadas e quantas ordens tiveram ações pendentes de serem executadas;
6. O pior caso ocorre quando pelo menos uma das listas de compra ou venda é totalmente executada, ou seja, o programa percorreu uma das listas de ordens completamente durante a etapa de negociação;

Observando as afirmações expostas, especialmente no que diz respeito ao item nº 2, podemos inferir que nosso algoritmo, até o momento, apresenta uma forte tendência de pertencer à classe de complexidade polinomial, pelo fato de uma estrutura de repetição dentro de outra ser uma característica comum nos algoritmos polinomiais.

Oportunidades de aperfeiçoamento da solução apresentada

Em meio a revisão e desenvolvimento da solução, foi verificado que a contagem de ações negociadas e a contagem das ordens de Compra e Venda que ficaram pendentes ao fim do programa, estavam sendo realizadas em laços de repetição separados, ou seja, percorria-se ambas as listas por completo duas vezes. Com isso, a fim de otimizar e simplificar o algoritmo, a função de contagem das ordens de Compra que ficaram pendentes, e função da contagem de ações negociadas, foram unificadas dentro de apenas uma função com laço de repetição único, armazenando assim as informações obtidas em dentro de um vetor.

Uma particularidade percebida durante a revisão do algoritmo, diz respeito ao fato de que quando uma ordem de venda tem suas ações totalmente negociadas, ela não é retirada da lista de ordens de

venda. Pelo fato dessa lista ser percorrida desde seu início a cada nova ordem de compra que é chamada pela estrutura de repetição, o algoritmo perde desempenho ao ter que analisar cada ordem de venda inúmeras vezes a fim de verificar se ela ainda possui ações a serem negociadas. Ao remover as ordens de venda da lista, o programa percorrerá um número menor de ordens, e não perderá tempo verificando se uma ordem ainda possui ações a serem negociadas.

Outro detalhe referente a não exclusão da ordem de venda da sua lista, diz respeito a execução da estrutura de repetição que percorre as ordens de venda mesmo que todas tenham sido já executadas por completo. Por não haver a sua retirada da lista de vendas, a cada nova ordem de compra chamada, será executado o laço que percorre todas as ordens de venda, e isso ocorre mesmo na hipótese de que todas as ordens de vendas já tenham sido executadas, causando perda de performance do programa.

Resultados obtidos

Com a implementação do algoritmo desenvolvido em linguagem Java, após sua execução para os casos de teste disponibilizados, obteve-se os seguintes resultados:

Tabela 1 - Resultados de execução

Arquivo teste	Nº de ordens	Nº ações negociadas	Nº de ordens de compras pendentes	Nº de ordens de vendas pendentes	Lucro (R$)	Tempo de execução (segundos)
trinta_enunciado.txt	30	848	6	9	74.005	0,000384
trinta.txt	30	707	8	11	613.847	0,00036
cem.txt	100	2.024	23	43	2.081.062	0,000663
trezentos.txt	300	6.376	97	102	7.242.603	0,00165
mil.txt	1.000	22.829	338	293	24.350.119	0,00416
tres_mil.txt	3.000	70.922	935	949	723.450.321	0,01
cinco_mil.txt	5.000	117.832	1.605	1.533	116.117.990	0,02
dez_mil.txt	10.000	236.112	3.127	3.107	235.293.389	0,04
cinquenta_mil.txt	50.000	1.177.041	15.512	15.624	1.182.651.970	0,35
cem_mil.txt	100.000	2.363.398	31.356	30.993	2.356.457.766	1,32
milhao.txt	1.000.000	23.548.489	312.124	312.870	23.539.430.037	491,84

Análise de desempenho

A fim de aferir melhor a performance do algoritmo bem como sua classe de complexidade, foram realizados três casos de testes adicionais. Compilando todos os desempenhos obtidos, temos os seguintes resultados:

Tabela 2 - Relação Nº de Ordens x Tempo Execução

Nº de ordens	**Tempo de execução (segundos)**
30	0,000384
30	0,00036
100	0,000663
300	0,00165
1.000	0,00416
3.000	0,01
5.000	0,02
10.000	0,04
50.000	0,35
100.000	1,32
300.000	27,27
500.000	65,32
700.000	210,09
1.000.000	491,84

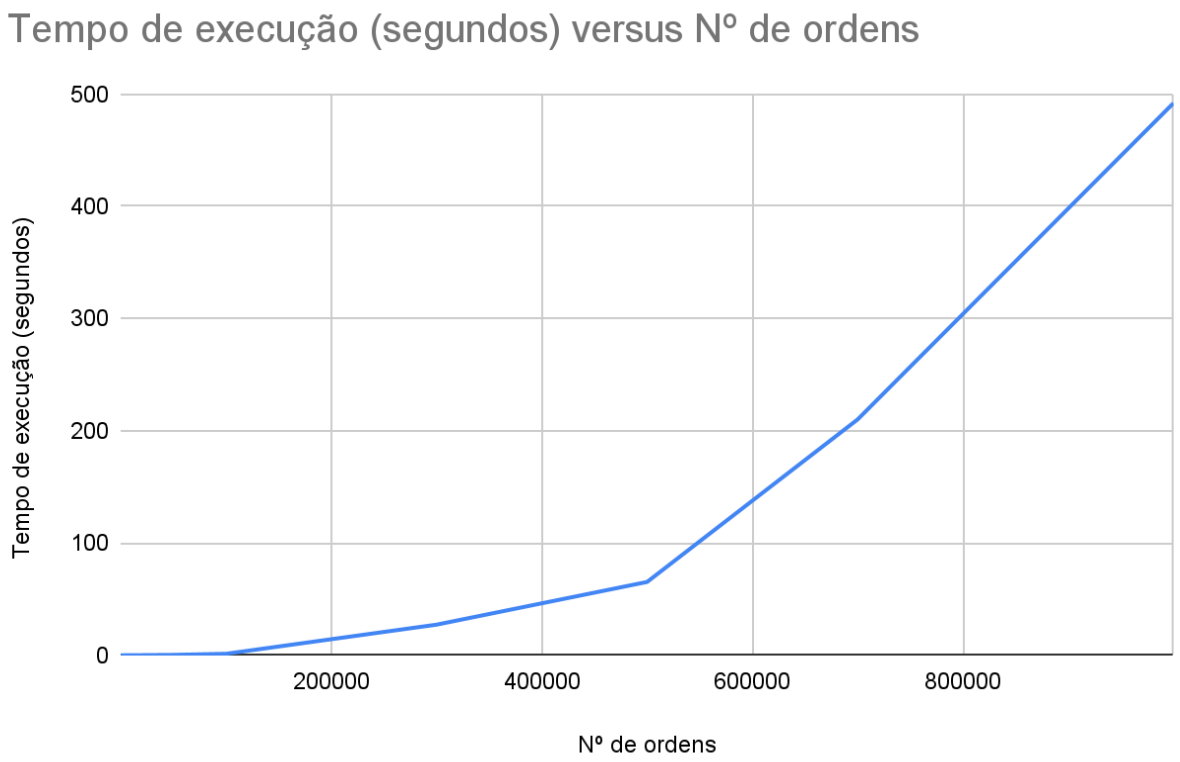


Figura 4: **Gráfico comparando o tempo de execução do programa em relação à quantidade de ações a serem negociadas por cada ordem enviada.**

Olhando para o gráfico não é possível obter nenhuma conclusão precisa. A função cresce de forma mais rápida a partir de uma entrada de 300.000 ordens, tendo uma maior inclinação a partir das 700.000 ordens, mas não suficientemente para ser considerada uma função exponencial. Levando isso em consideração, pode-se imaginar que realmente se trata de um algoritmo de classe polinomial, mas para ter certeza, aplica-se escala logarítmica no eixo Y, obtendo assim o seguinte resultado:

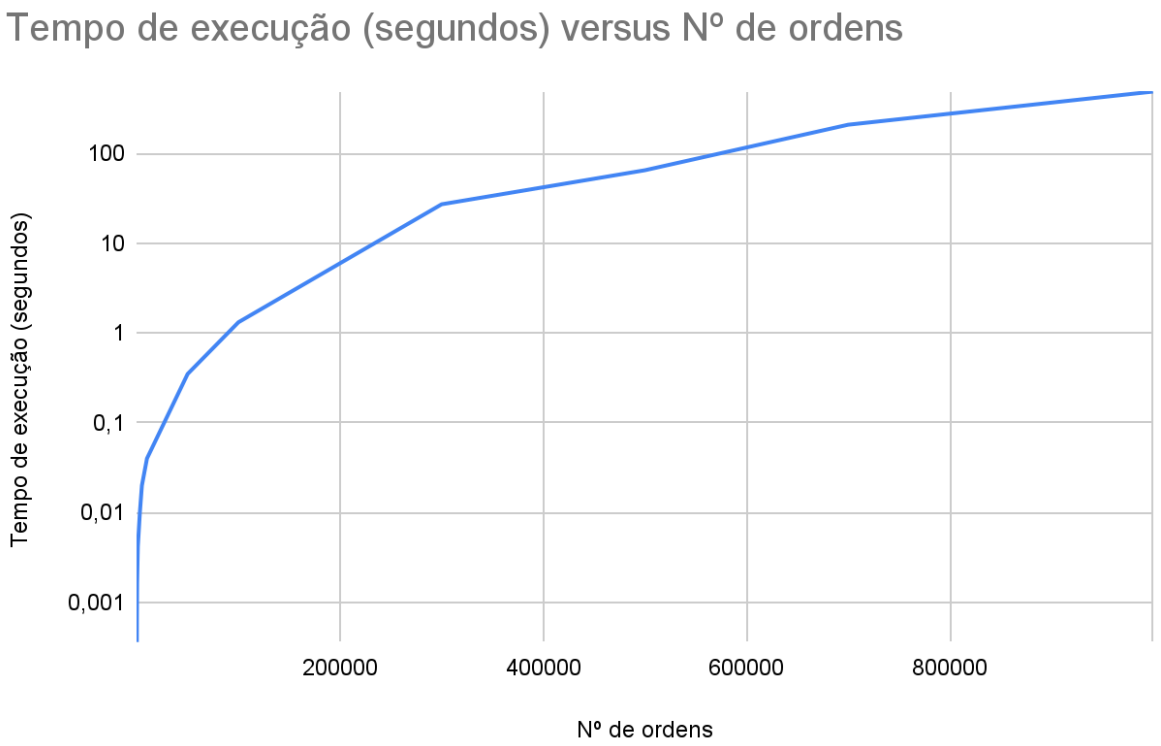


Figura 5: **Gráfico em escala logarítmica comparando o tempo de execução do programa em relação à quantidade de ações a serem negociadas por cada ordem enviada.**

Se o algoritmo analisado fosse exponencial, no momento em que fosse gerado o gráfico com escala logarítmica , a aparência seria de uma reta, porém o resultado obtido foi um gráfico de função logarítmica, confirmando a suspeita de que o algoritmo seria polinomial.

Conclusões

A solução apresentada se mostrou simples, eficiente e de fácil entendimento. Apesar de ser iterativa, ela foi desenvolvida de forma razoavelmente clara, não precisando de uma implementação complicada. Acreditamos ter desenvolvido uma solução interessante para o problema proposto, conseguindo trazer de maneira rápida resultados que envolvam a leitura e interpretação de um grande volume de ordens, apesar das oportunidades de aperfeiçoamento do algoritmo apontadas anteriormente para melhorar seu desempenho.

Da análise dos resultados, percebe-se que a quantidade de ordens interpretadas do arquivo não influenciou significativamente no tempo de execução do algoritmo com entradas de até 100.000 ordens. O tempo de execução só cresceu de forma relevante a partir da entrada de arquivos com um número mais volumoso de ordens, pois até antes disso, a diferença entre os tempos de execução era praticamente irrelevante.

Por fim, constata-se que o objetivo do trabalho proposto foi concluído porque a solução apresentada informou os resultados solicitados a partir dos arquivos de entrada disponibilizados.

Referências

[1] CORMEN, T. H.; LEIERSON, E. C.; Rivest, R. L.: **"Introduction to Algorithms"**. Mc-Graw Hill Book Co., The MIT Electrical Engineering and Computer Science Series, Cambridge, 1990.