Trabalho Prático 1: Biblioteca do Filipe

Francis Carlos dos Santos

Matrícula: 2012022167

1 Introdução

O objetivo desse trabalho é implementar um sistema que ajude no controle do acervo de uma Biblioteca. Afim de aplicar os conteudos passados em sala, será simulado uma memória principal bem limitada. Cada entrada consiste em N livros quedeverão ser inseridos em até E estantes, onde cada prateleira comporta um maximo de L livros. Feito isso, deverá ser gerado um arquivo com todos os livros ordenados e um arquivo para cada estante. E por último serão realizadas K consultas que devem informar o consultante se o livro encontra-se: desponível, alugado ou se o livro não existe no sistema.

2 Solução do Problema

O problema foi divido em partes, cada parte tem como saída um arquivo que será utilizado para montar a saida final. O algoritmo basicamente, divide a entrada em fitas ordenadas, faz a ordenação em seguida são geradas as estantes e juntamente o arquivo de indices. Porultimo a consulta é feita utilizando os indices e em seguida é realizada a pesquisa binaria na estante. Esse fluxo pode ser visto na figura 1

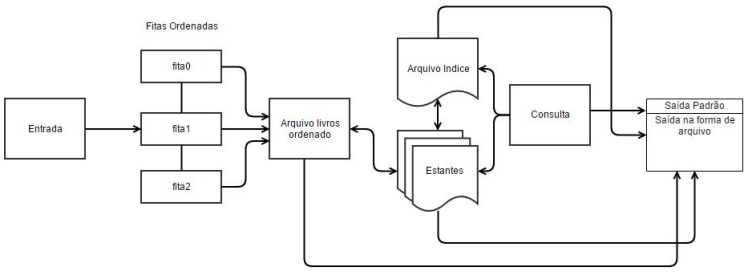

Figura 1: Diagrama representando a relação entre os arquivos gerados até a saida.

- *Limitação de Memória*: A limitação teórica de memória é a principal dificuldade para resolver o problema. Para contornar essa dificuldade, na fase de leitura todos os os registros da entrada serão escritos em um arquivo binário.

- *Ordenação em Memória secundaria*: Sempre teremos M registros na memória principal, e sempre teremos N >M. Portanto, para colocar os livros em ordem alfabética, um algoritmo de ordenação externa baseado no algoritmo de intercalação balanceada foi implementado. Esse algoritmo possui dois passos principais.

    1. *Criação de Blocos Ordenados*: Nessa fase os registros são lidos de M em M e ordenados e escritos em arquivos. Para garantir que o numero maximo de arquivos abertos não exceda o maximo permitido pelo sistema operacional. Cada arquivo é aberto para a escrita e fechado em seguida. Nesse processo também é utilizado um algoritmo de ordenação interna para a ordenação dos M registros na memória. Nesse problema, o registro será ordenado de acordo com no título do livro,ou seja, o numero de comparações é maior que o numero de cópias.Considerando que M não seria um numero muito grande, foi analisado que o algoritmo de ordenação por inserção teria uma boa perfamance.
    2. *Intercalação*: Na intercalação, todos os blocos gerados são intercalados em uma única fita de saída. As duas fitas pareadas serão sempre a fita de saida e uma fita contendo um bloco ordenado. Após ficar vazia a fita é deletada e a ultima tem o nome alterado para *livros_ordenados*, como requerido na especificação desse trabalho. A princípio foi realizada uma tentativa de implentação da ordenação externa utilizando somente 2M fitas no processo de escrita e intercalação, porém a dificuldade de manipular os ponteiros para varios arquivos binários contendo os blocos e para varios blocos dentro de um mesmo arquivo acabou por influenciar na busca de uma solução alternativa.

- *Montagem das Estantes*: Uma vez que os livros estão ordenados, a função *OrganizaEstantes* realiza a divisão dos livros nas estantes lendo até M arquivos por vez, para cada arquivo o contador *cont* é incrementado e a leitura continua até que o *cont* seja igual a L,número maximo de livros permitido em cada estante, ao fim de uma estante o contador é zerado. Isso porque, cada vez que o contador é igual a zero, representa que o proximo registro representa o primeiro livro de uma estante e o ultimo registro representa o ultimo registro dessa mesma estante, logo o arquivo de *indice* é construido enquanto as estantes estão sendo montadas. No fim da execução dessa função, todas os arquivos *estante* estarão criados, assim como o arquivo *indice*, que será usado para fazer a consulta.

- *Consulta*: A consulta é realizada utilizando busca binária. Primeiramente é encontrado a estante que o livro pertence. Feito isso a função que realiza a pesquisa binária é chamada, a busca é iniciada no meio da estante, até encontrar ou não o livro pesquisado.

As estruturas principais utilizadas para implementar as soluções citadas acima podem ser vistas no algoritmo 1.

2.1 Saida

A saída do problema é feita través de uma serie de arquivos. Todos os arquivos intermediarios utilizados para ordenar ou receber a entrada são excluidos antes do final da execução. Os arquivos são:

- *livros_ordenados*: este arquivo contem todos os registros contendo o titulo do livro e se o mesmo encontra-se disponível ou não.

- *estante*: o programa deverá gerar E arquivos estante no fimd e sua execução. Cada arquivo pode conter até L livros.
- *indice*: um arquivo indice, contendo o primeiro e ultimo livro em cada estante.

Algoritmo 1: Estruturas utilizadas

```
1  typedef struct
2    char *tituloLivro[52]
3    int disponibilidade
4  Biblioteca;

5  typedef struct
6    int inicio,fim,nLivro;
7  indice;
```

3 Análise de Complexidade

Nesta seção, será apresentada a análise do custo teórico de tempo e de espaço para as principais funções do algoritmo.

-

3.1 Análise Teórica do Custo Assintótico de Tempo

Nessa seção será analisado o comportamento assintótico do algoritmo. Esse processo será feito através da analise das funções principais do algoritmo.

- *leEscreveEntrada*: Essa função percorre todos os registros,livros, da entrada uma unica vez e os escreve em um arquivo. Logo assintóticamente essa função seria $O(\mathrm{N})$.
- *OrdenaExterno*: Uma das principais funções do algortimo, realiza a quebra do arquivo de entrada em blocos ordenados. Para uma entrada com N livros e uma memória de tamanho M, teremos $\mathsf{N} \div \mathsf{M}$ criados caso N seja multiplo de M e somado ao resto dessa divisão caso o numero de livros não seja um multiplo do tamanho da memória. Para cada um dos blocos a função de ordenação interna é chamada. Nesse caso a ordenação por inserção foi utilizada e tem melhor caso $O(\mathrm{n})$ e caso médio $O(\mathsf{N}^2)$. Nessa primeira parte a complexidade temporal assintótica da função seria $O(\frac{\mathsf{N}^3}{\mathsf{M}})$. A segunda parte dessa função realiza chamadas a função de intercalação que terá sua complexidade analisada no proximo item. O número máximo de passadas para essa função é N.
- *intercalacao*: Essa função é chamada $\frac{\mathsf{N}}{\mathsf{M}}$ vezes. Para cada chamada no máximo M arquivos são lidos do bloco ordenado e concatenado com um bloco de saída maior. Portanto essa função fará no maximo $\mathsf{N} - \mathsf{M}$ operações de comparação no maxímo $(\frac{\mathsf{N}}{\mathsf{M}} + 1)$ vezes. Portanto a complexidade dessa função será $O(\mathsf{N}^2)$. // Essa função terá no máximo N acessos ao arquivo.

- *OrganizaEstantes*: As E estantes são criadas nessa função. Essa função contém um loop que inicializa uma variavel de tamanho máximo E, logo $O(\mathrm{E})$. Depois a arquivo de livros ordenados é percorrido para se preecher as estantes, logo percorrendo um máximo de N registros, portanto $O(\mathrm{N})$. A complexidade assintótica dessa função será dada então por $O(\mathrm{N})$. *PesquisaBinaria*: A busca binária é feita sempre sobre uma unica estante. Portanto no pior caso essa busca irá percorrer L registros. E no melhor caso $O(\lg \mathsf{N})$.

Como explicado acima a maior comlexidade será a da função de ordenação, nesse caso essa função dominará assintóticamente as outras. Logo, a complexidade teórica de cuso assintótico desse algoritmo será $O(\frac{\mathrm{N}^3}{\mathrm{M}})$.

3.2 Análise Teórica do Custo Assintótico de Espaço

Nesse programa 2 estruturas principais foram utilizadas para implementar a solução, como mostrado em 1. No entanto o como o objetivo desse trabalho é aprender a usar e manipular estruturas em memória secundária, o máximo de de espaço consumido pelo programa será de $\mathsf{M} \times \mathsf{K}$ sendo K o espaço ocupado pela estrutura responsavél por armazenar os livros. Logo a complexidade de espaço seria $O(\mathrm{M})$.

4 Análise Experimental

A análise experimental dessa implementação foi dividida em varios casos. Como se trata de um problema de ordenação é interessante variar o tamanho da entrada N, o tamanho da memória e o número de consultas realizadas e analisar como o algoritmo responde a tais variações. O tempo de execução foi obtido por meio da utilização da função *time*. O testes foram realizados em um ambiente Elementary OS em uma mãquina com proessador Core i7-4720HQ 2,6GHz e 8GB de memória ram.

4.1 Variação do tamanho da entrada (N)

Os valores escolhidos para mostrar a execução foram valores de N variando de 100 a 3000. Os outros valores foram mantidos contantes com M=10, E=51,L=20 e K = 100. O gráfico 2 mostra como a o aumento da entrada se relaciona com o tempo de execução. Pode se dizer que o resultado observado não é necessariamente o previsto na analise de complexidade assintótica. Porém como será visto mais a frente, outros fatores tem enorme relevância para o comportamento assintótico do algoritmo. O gráfico 3 mostra o como a

utilização de memória varia de acordo com o valor da entrada. Como previsto na análise de complexidade de espaço o comportamento é linear.

4.2 Variação do tamanho da memória (M)

Como explicado antes, esse problema é também sensível ao tamanho da memória disponível. Para uma memória de tamanho muito grande, permitiria que a ordenação fosse feita mais rapidamente, uma vez que a memória interna seria maior. Esse é um dos motivos que

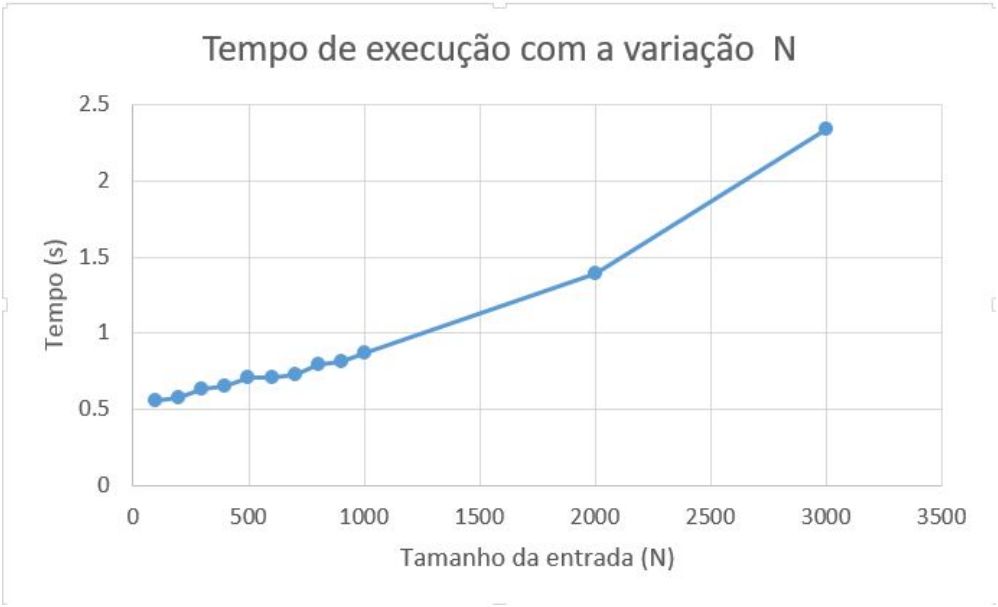

Figura 2: Gráfico que mostra o tempo de execução em relação ao aumento na entrada.

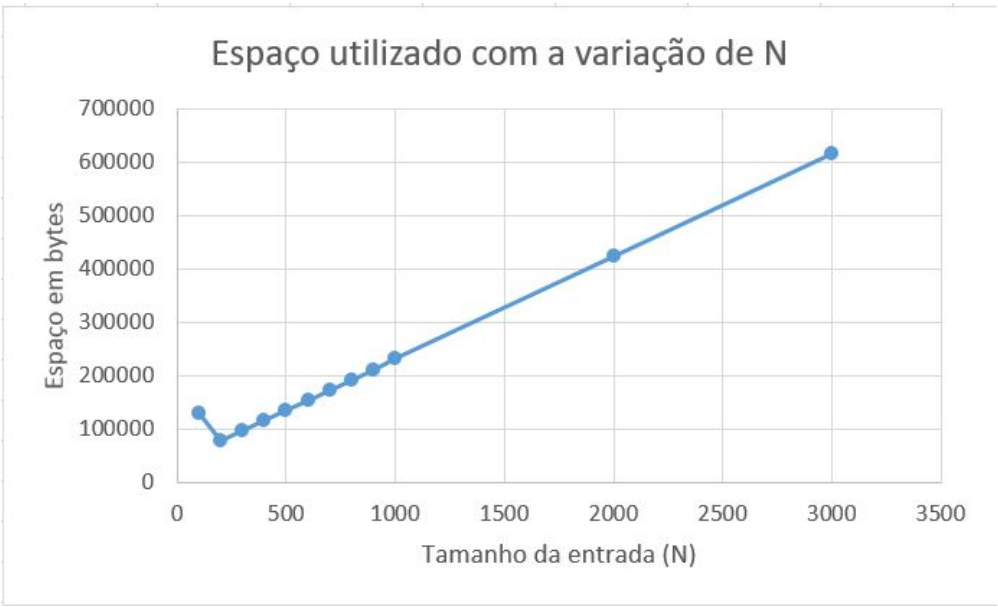

Figura 3: Gráfico que mostra o espaço utilizado com o aumento da entrada.

uma melhoria que podia ser feita no algoritmo seria a implementação de um algoritmo de ordenação como o Shellsort. Esse algoritmo tem um ótimo desempenho quando trabalha com casos de ordenação conde o numero de comparações e grande e o numero de cópias pequeno. Nesse teste o tamanho da memória M foi variado de 10 a 100, enquanto os outros parametros foram mantidos constantes com N = 2000, E = 11, L = 200 e K = 100. O grafico 4 mostra o comportamento do algoritmo para tal variação. Como esperado com o aumento de M, o tempo de execução decresce.

4.3 Variação do numero de buscas(K)

O número de buscas está relacionado a outra parte sensível do programa que é a pesquisa. O numero de buscas K foi variado de 100 a 1000. Os outros parâmetros foram mantidos

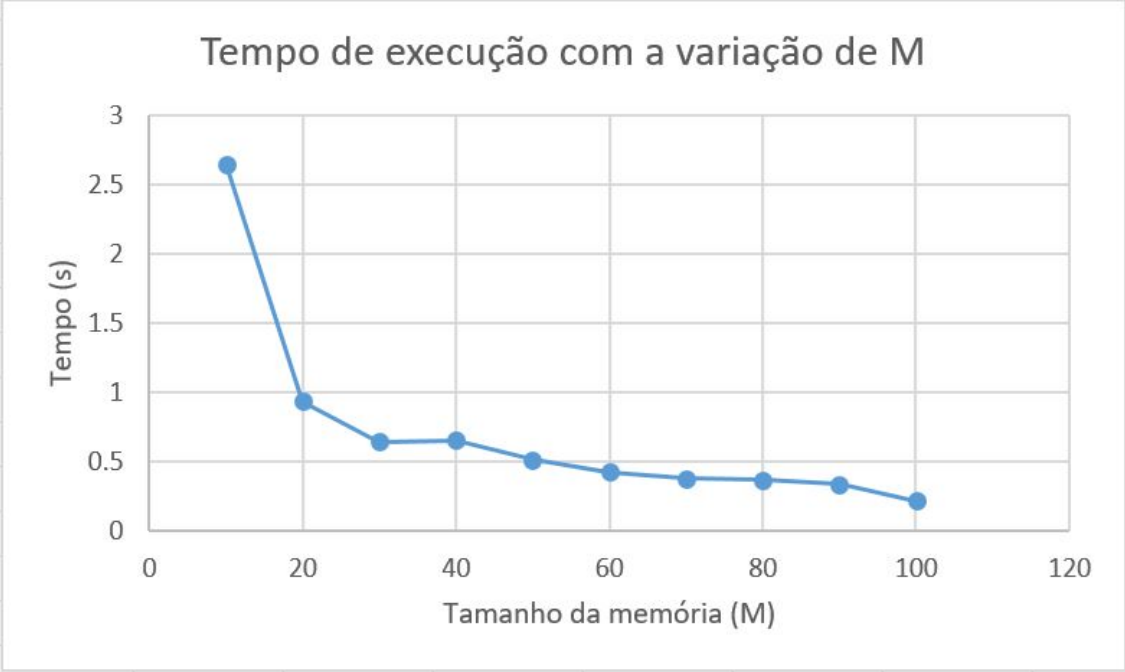

Figura 4: Gráfico que mostra o tempo de execução com o aumento da memória

constantes com N = 1000,M = 10, E = 51 e L = 20. Foi decidido fazer essa análise pois o numero de pesuisas se relaciona intimamente com o numero de passadas realizadas sobre as estantes. O gráfico 5 mostra a como a tempo de execução varia quando o numero de pesquisas realizada é grande.

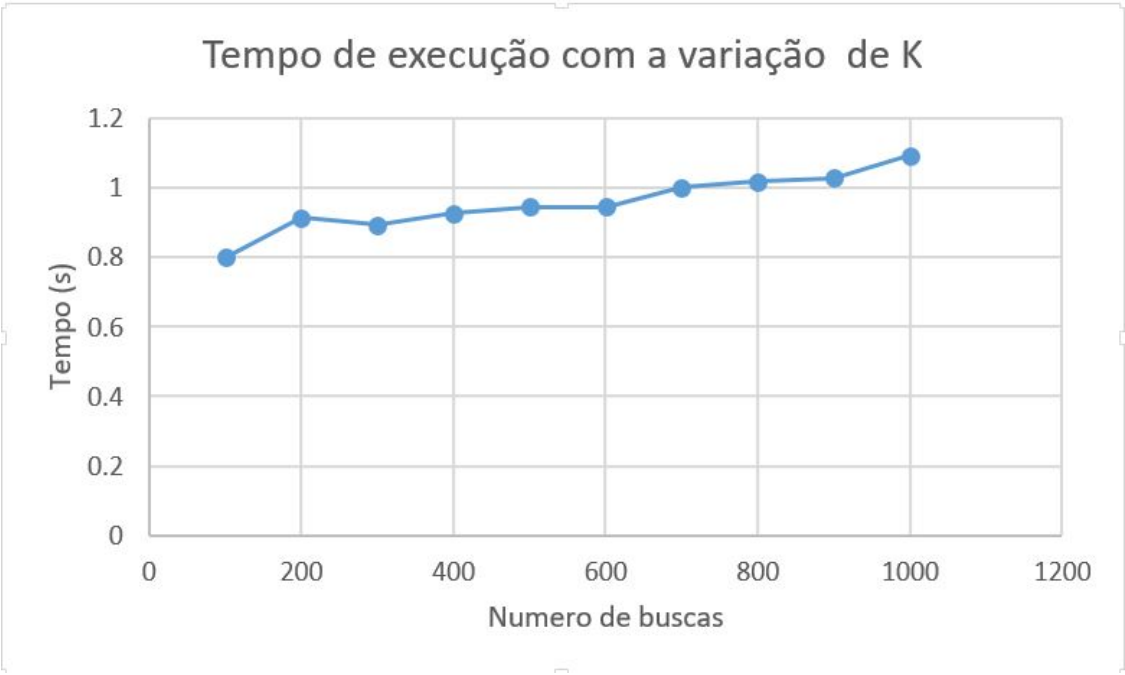

Figura 5: Gráfico que mostra o tempo de execução para uma variação no numero de pesquisas realizadas.

O gráfico 6 mostra o aumento também linear, como previsto, do uso de memória para

a execução do programa com a variação de K.

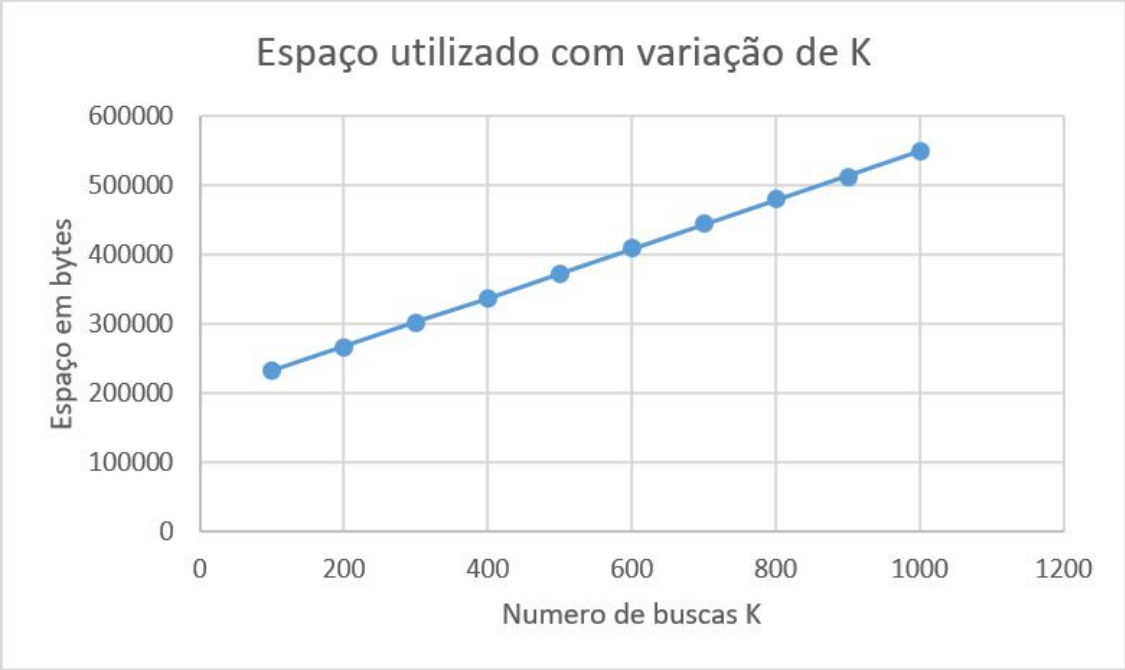

Figura 6: Gráfico que mostra o consumo de espaço para o aumento do número de pesquisas realizadas.

4.4 Variação da quantidade de livros por estante (L)

De acordo com a descrição do problema, sempre haverá espaço para todo os livros, logo o nesse caso é intressante variar o numero de livros por estante pois esse parametro influencia diretamente no numero de passadas aos arquivos e mais fortemente na busca binaria. Para esse experimento, foram fixados N=2000, M = 20, E = 201 e K = 50. O gráfico 7, mostra a variação do tempo em relação a variação da quantidade de livros por estante.

5 Conclusões

Nesse trabalho foi implementado um sistema com o foco a utilização e manipulação de dados armazenados em memória secundaria. O problema a ser resolvido foi interessante pois nos levou a nalisar varios fatores que podem influenciar na complexidade deum problema, como a diferença de desempenho de determinados algoritmos de ordenação com relação ao número de cópias e comparações. Interessante também perceber a influencia que chamadas sistemicas como as feitas pelas funções *remove()*, utilizada para apagar arquivos temporarios utilizados e a função *malloc* se usadas indiscriminadamente podem ter no algoritmo. Por fim,o trabalho foi implementado, não sem dificuldades mas não foram percebidos erros para nenhum caso de test. Para um trabalho futuro eu consideraria implementar a ordenação utilizando-se duas o Shellsort quando M $>$ 20 e o insertion sort para quando M $<$ 20.

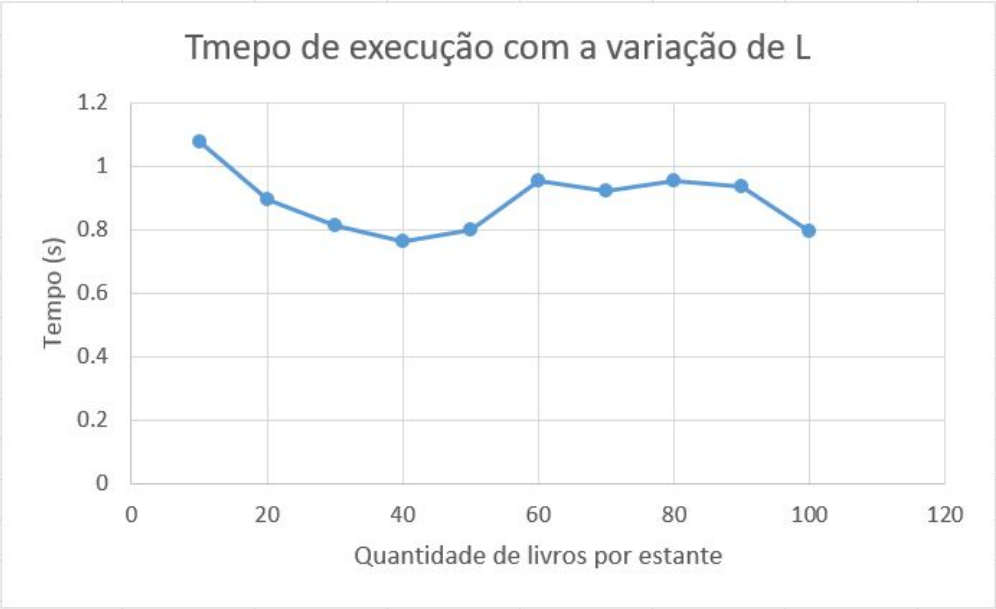

Figura 7: Gráfico que mostra o tempo de execução para diferentes valores de L.

6 Referências Bibliográgicas

1 - Robert Sedgewick and Kevin Wayne. Algorithms, 4th Edition. Addison-Wesley, 2011.
2 - http://www.cplusplus.com