UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE – UFRN
BACHARELADO EM TECNOLOGIA DA INFORMAÇÃO – BTI

Relatório: análise empírica dos algoritmos de busca (sequencial e binária), ordenação (insertionSort, selectionSort, quickSort e mergeSort) e inserção-remoção (vetor sequencial e lista ligada)

Componentes:
Bianca Cristiane Ferreira Santiago
2016038862
Jaine Rannow Budke
2016035243

29 de junho de 2017
Natal, RN

Sumário

1. INTRODUÇÃO

Este relatório tem como objetivo explicitar a metodologia utilizada para o desenvolvimento do projeto; as pesquisas realizadas e o aprendizado adquirido no decorrer do semestre da disciplina. Bem como os detalhes de implementação; da organização dos arquivos em respectivos diretórios ao funcionamento do programa; e os resultados e conclusões obtidos.

Em síntese, este documento visa esclarecer para o leitor todas as diretivas necessárias para o entendimento do funcionamento do projeto e, explicar como as análises empíricas foram realizadas.

2. ANÁLISE EMPÍRICA ALGORITMOS DE INSERÇÃO-REMOÇÃO

A análise foi realizada tendo como base os algoritmos de inserção e remoção, implementados em uma lista ligada e em um vetor. O projeto foi desenvolvido utilizando a linguagem C++ e o conceito de boas práticas de programação, como modularização e o uso do makefile e padrão Doxygen de documentação.

2.1. Organização do projeto

O projeto foi implementado com modularização em arquivos. Os arquivos .cpp estão localizados na pasta /src. O arquivo sequencias.cpp é composto pela implementação dos dois algoritmos de inserção (em vetor e em lista ligada) e ps dois algoritmos de remoção (em vetor e em lista ligada), enquanto o main.cpp possui funções para fazer a medição do tempo dos algortimos de inserção e remoção - levando em consideração chaves de busca diferentes -, bem como a implementação funções redirecionamento de dados para arquivos no formato txt.

2.2. Funcionamento dos algoritmos

Ao executar o algoritmo, são gerados doze arquivos no formato txt. Cada grupos de três arquivos são referentes à Inserção em Vetor, Remoção em Vetor, Inserção em Lista Ligada e Remoção em Lista Ligada.

Assim, os arquivos possuem referência ao pior, melhor e médio caso, para tamanhos de base de dados diferentes. A lógica implementada para encontrar cada caso segue:

- Gera base de dados de tamanho n;
- Processo repetido 100 vezes para um mesmo tamanho de base;
  - Marca tempo de execução;
  - Verifica se é o melhor ou pior tempo e salva o valor, se for o caso;
  - Incrementa variável com soma de todos os tempos, com o novo tempo marcado;
- Faz média de todos os tempos (soma/100).

Os tamanhos da base de dados (n) com que cada algoritmo é executado são pré-definidos e equivalentes a:

- { 500000, 600000, 700000, 800000, 900000, 1000000, 5000000, 6000000, 7000000, 8000000, 9000000, 10000000 }

O tempo de execução de cada algoritmo é medido utilizando funções da biblioteca chrono. Os algoritmos de inserção e remoção em vetor e em lista ligada foram executados com tamanhos de base de dados diferentes devido a sua complexidade e consequente tempo de execução.

2.3. Gráficos para análise empírica

A inserção e remoção em um vetor (Gráfico 1 e Gráfico 3, respectivamente) e inserção e remoção em lista ligada (Gráfico 2 e Gráfico 4) tiveram um crescimento característico da complexidade desses algoritmos. Ou seja, um crescimento linear ao aumentar o tamanho da base de dados analisada. Vale ressaltar que em todos os casos, a linha de melhor não chega a aparecer devido à velocidade em que é processada. Ao aumentar a escala, é possível sua observação, no entanto, tal alteração exige uma imensa capacidade de execução, não encerrando o processo.

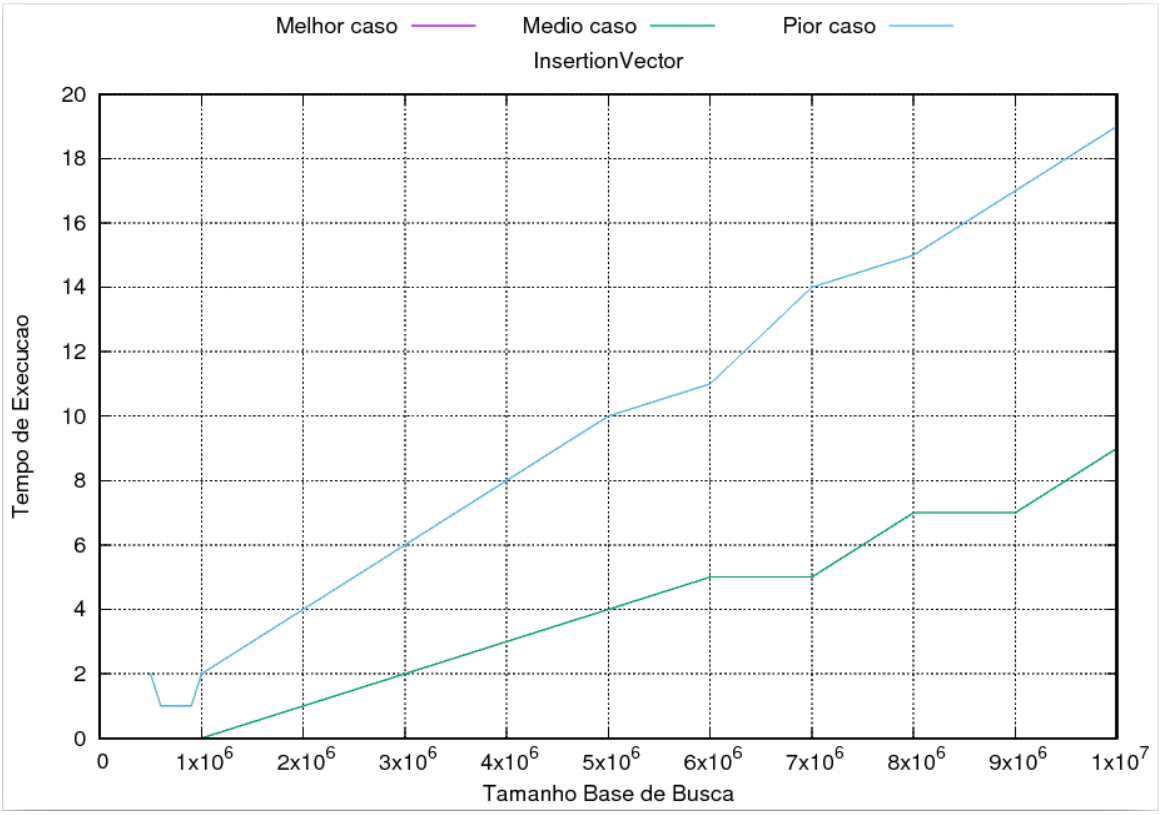

Gráfico 1 – Análise do algoritmo de inserção em vetor

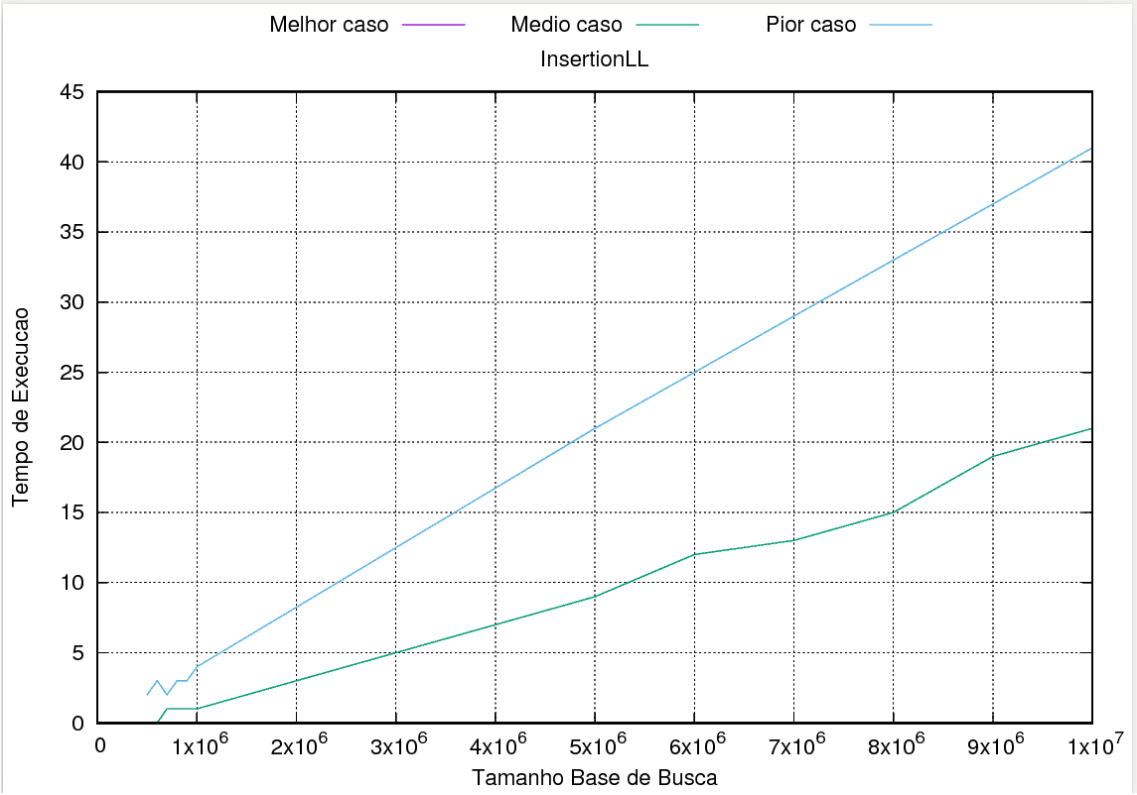

Gráfico 2 – Análise do algoritmo de inserção em lista ligada

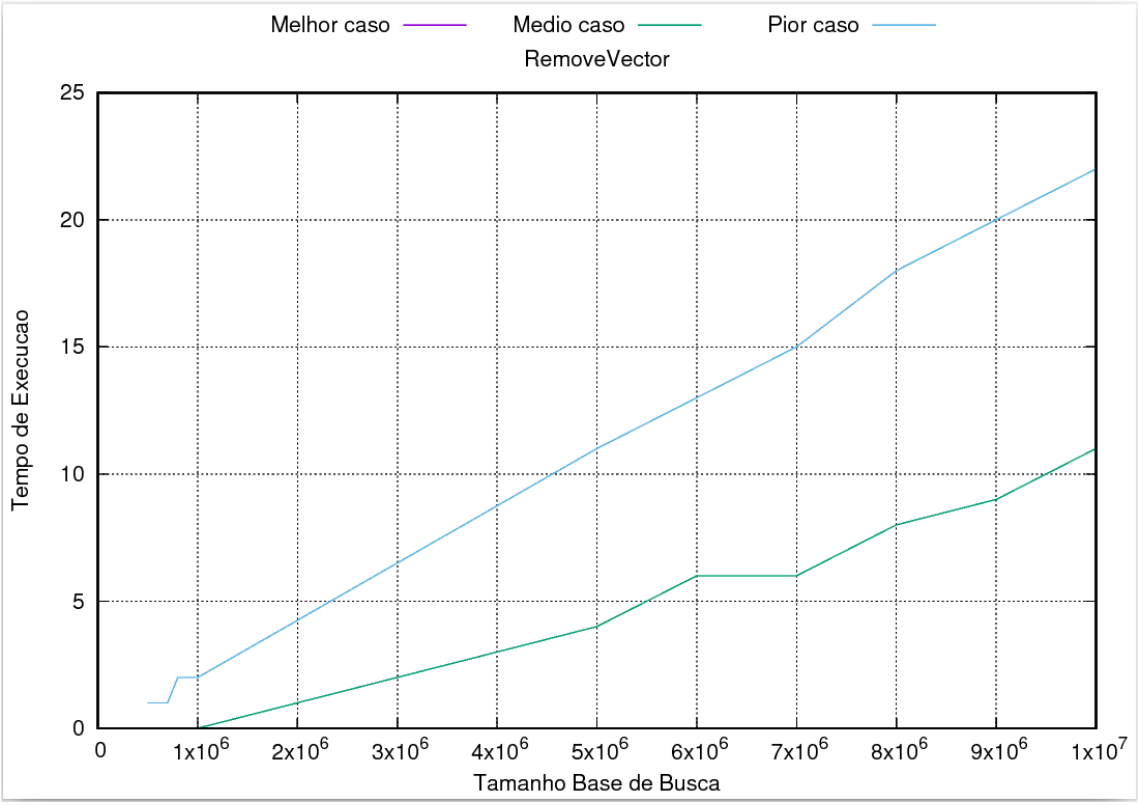

Gráfico 3 – Análise do algoritmo de remoção em vetor

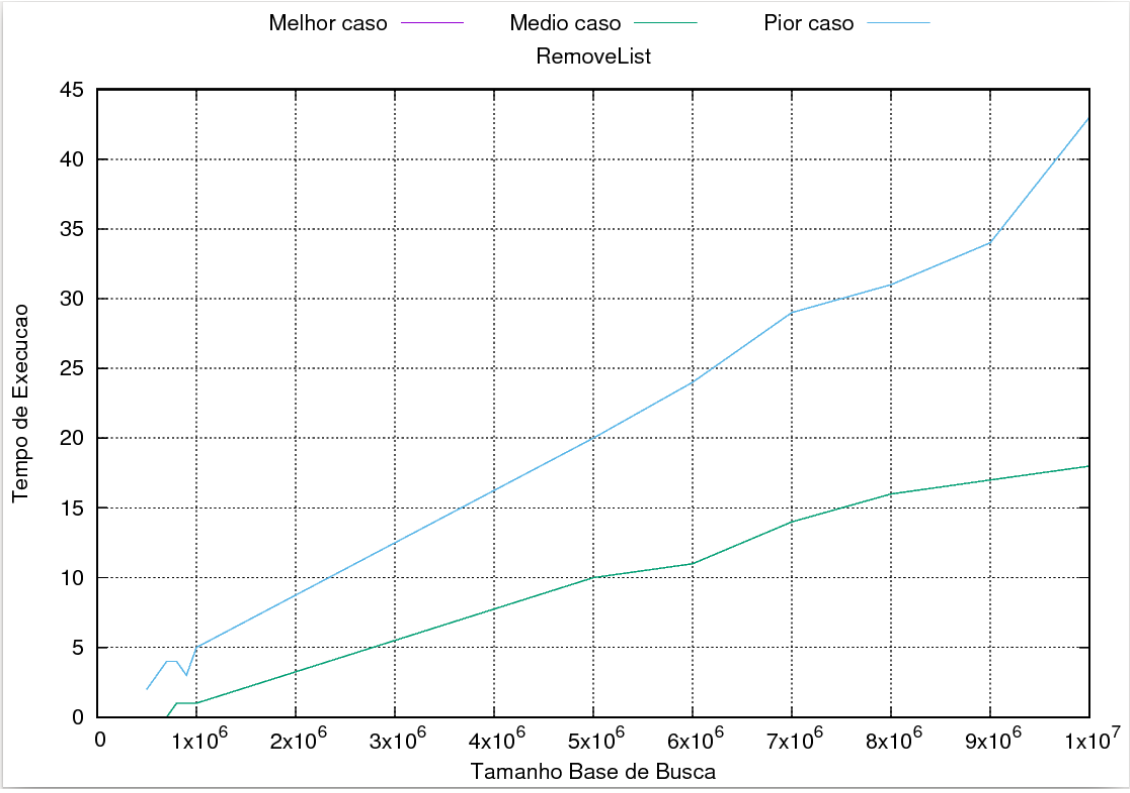

Gráfico 4 – Análise do algoritmo de remoção em lista ligada

3. ANÁLISE EMPÍRICA ALGORITMOS DE ORDENAÇÃO

A análise foi realizada tendo como base os algoritmos de ordenação, implementados através do insertionSort, selectionSort, quickSort e mergeSort. O projeto foi desenvolvido utilizando a linguagem C++ e o conceito de boas práticas de programação, como modularização e o uso do makefile e padrão Doxygen de documentação.

3.1. Organização do projeto

O projeto foi implementado com modularização em arquivos. Os arquivos .cpp estão localizados na pasta /src. O arquivo ordenacao.cpp é composto pela implementação dos quatro algoritmos de ordenação, enquanto o main.cpp faz a chamada da análise dos algoritmos, bem como direciona os resultados para arquivos no formato txt. Já a manager.cpp possui funções para gerar bases de dados aleatórias e funções para fazer a medição do tempo dos algortimos de ordenação - levando em consideração chaves de busca diferentes.

3.2. Funcionamento dos algoritmos

Ao executar o algoritmo, são gerados doze arquivos no formato txt. Grupos de três arquivos são referentes a um mesmo tipo de ordenação:

1 – insertionSort;
2 – selectionSort;
3 – quickSort;
4 – mergeSort.

Assim, os arquivos possuem referência ao pior, melhor e médio caso, para tamanhos de base de dados diferentes. A lógica implementada para encontrar cada caso segue:

- Processo repetido 100 vezes para um mesmo tamanho de base:
  - Gera base de dados de tamanho n;
  - Marca tempo de execução;
  - Verifica se é o melhor ou pior tempo e salva o valor, se for o caso;
  - Incrementa variável com soma de todos os tempos, com o novo tempo marcado;
- Faz média de todos os tempos (soma/100).

Os tamanhos da base de dados (n) com que cada algoritmo é executado são pré-definidos e equivalentes a:

- Algoritmos insertionSort e selectionSort:
  - { 1000, 5000, 10000, 15000 }
- Algoritmos quickSort e mergeSort:
  - { 1000, 5000, 10000, 15000, 50000, 100000, 500000, 600000, 700000, 800000, 900000, 1000000 }

O tempo de execução de cada algoritmo é medido utilizando funções da biblioteca chrono. Os algoritmos de ordenação insertionSort e selectionSort foram executados com tamanhos de base de dados diferentes devido a sua complexidade e consequente tempo de execução.

3.3. Gráficos para análise empírica

A ordenação por insertionSort (Gráfico 1 e Gráfico 3) e por selectionSort (Gráfico 2 e Gráfico 4) tiveram um crescimento característico da complexidade desses algoritmos. Ou seja, um crescimento linear ao aumentar o tamanho da base de dados analisada. Vale ressaltar que os gráficos 1 e 2 apresentam o resultado em uma escala diferente da análise do MergeSort e QuickSort, apenas para melhor visualização, enquanto os gráficos 3 e 4 demonstram o resultado utilizando a mesma escala, de modo que se possa notar a disparidade no tempo de execução entre os algoritmos de ordenação para uma mesma base de dados.

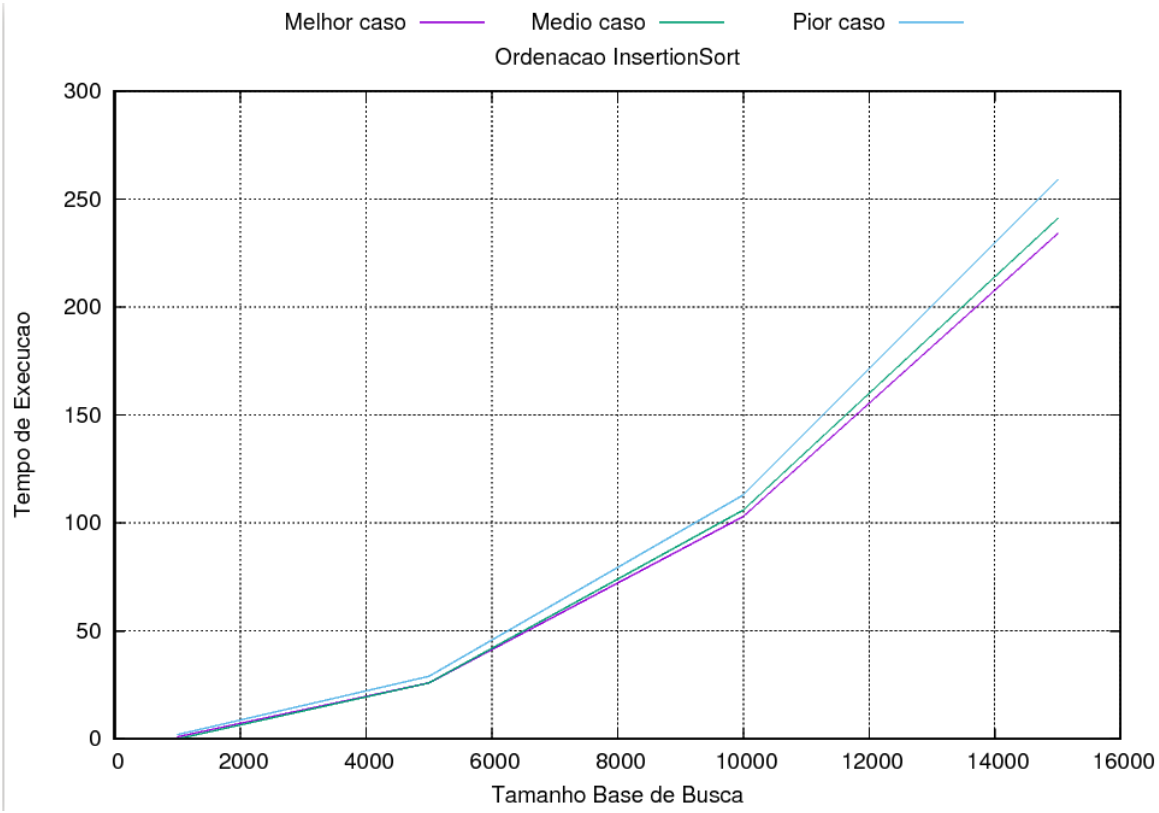

Gráfico 1 – Análise do algoritmo de ordenação insertionSort

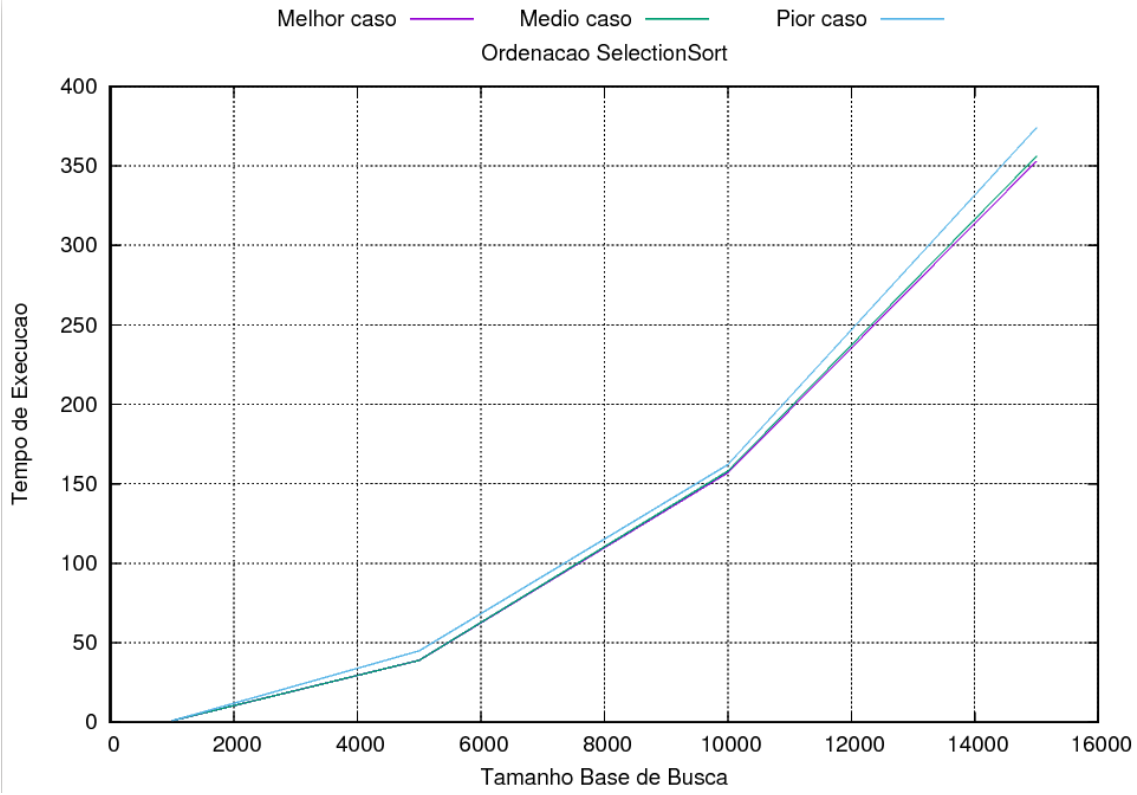

Gráfico 2 – Análise do algoritmo de ordenação selectionSort

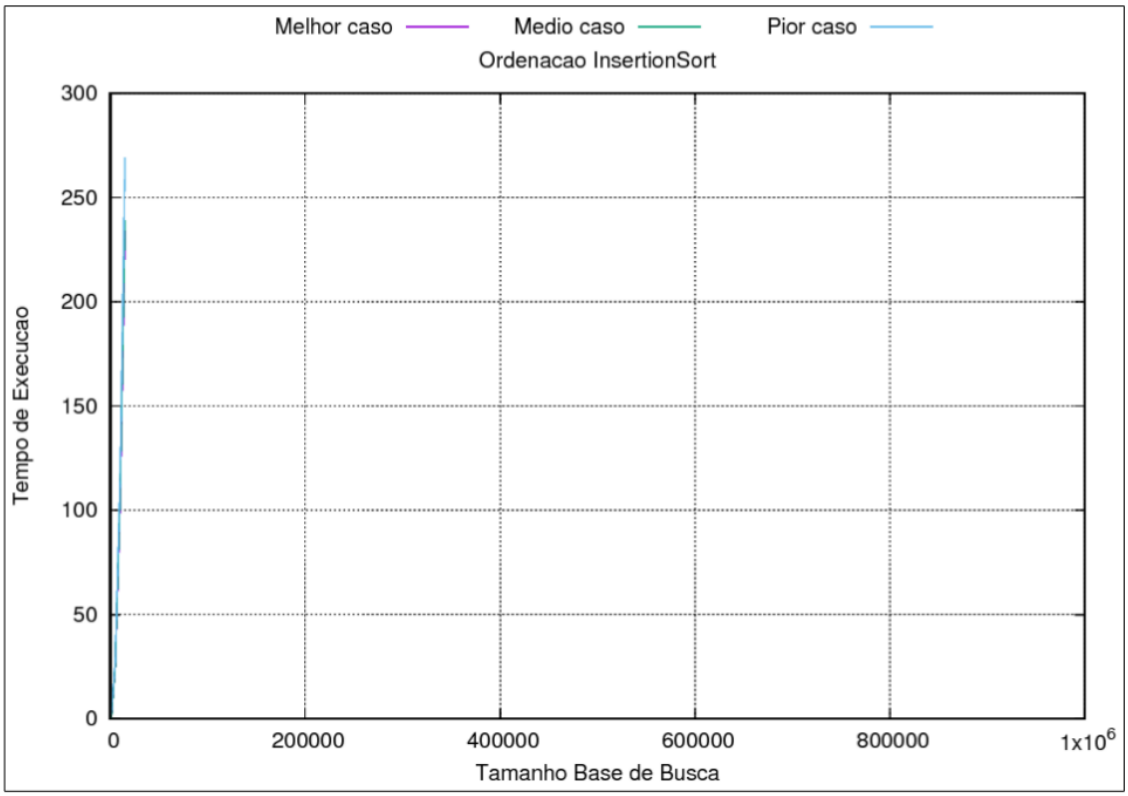

Gráfico 3 – Análise do algoritmo de ordenação insertionSort (com ajuste na escala)

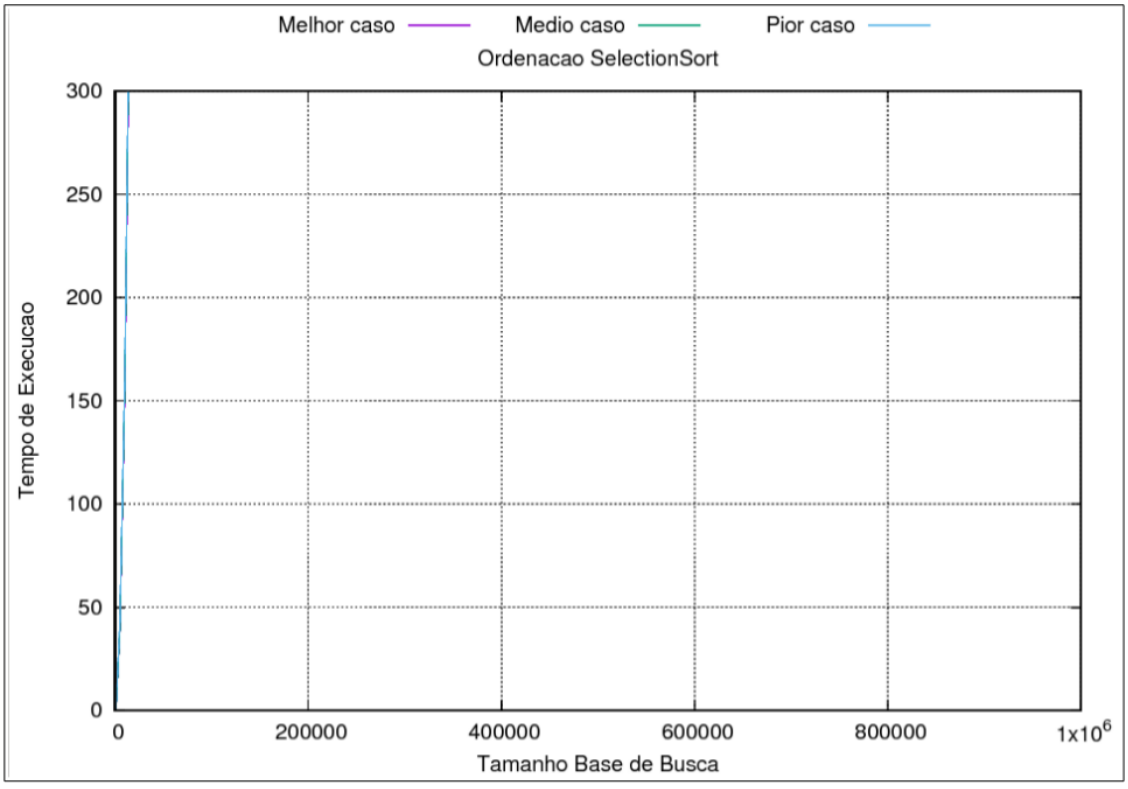

Gráfico 4 – Análise do algoritmo de ordenação selectionSort (com ajuste na escala)

A ordenação por quickSort (Gráfico 5) e por mergeSort (Gráfico 6) tiveram um crescimento característico da complexidade desses algoritmos, ou seja, nlog(n). Nota-se, portanto, que o tempo de execução desses algoritmos é bem menor se comparado com o inserionSort e o selectionSort.

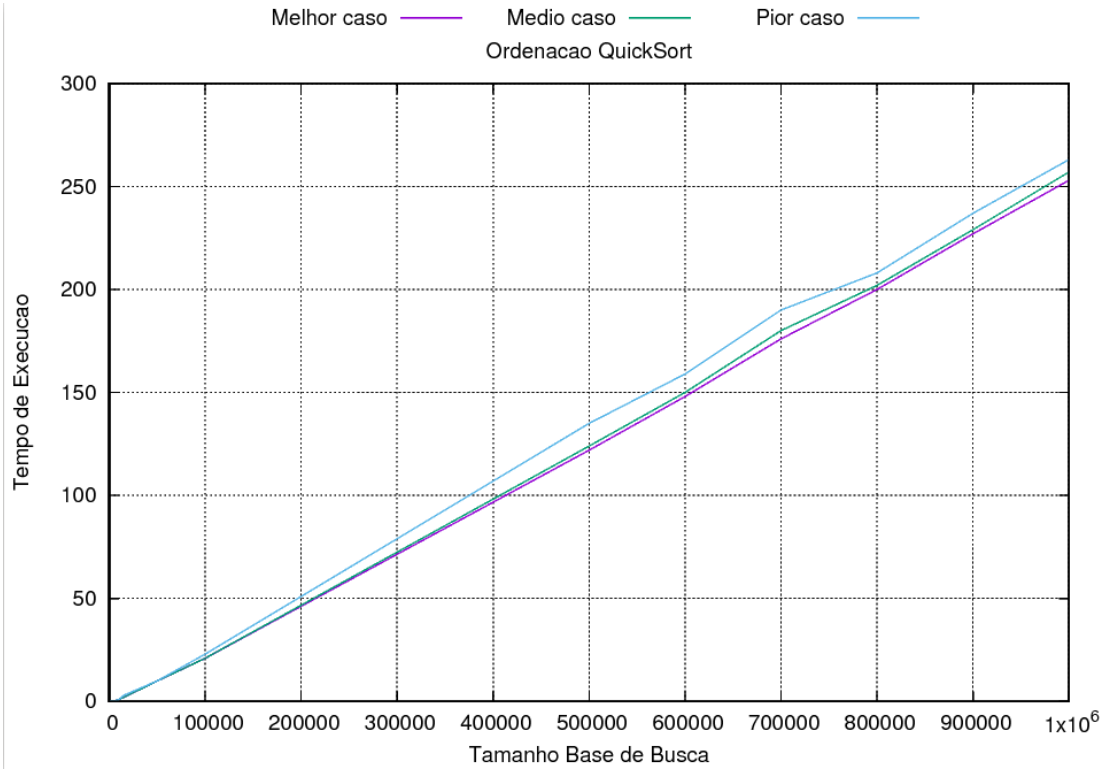

Gráfico 5 – Análise do algoritmo de ordenação quickSort

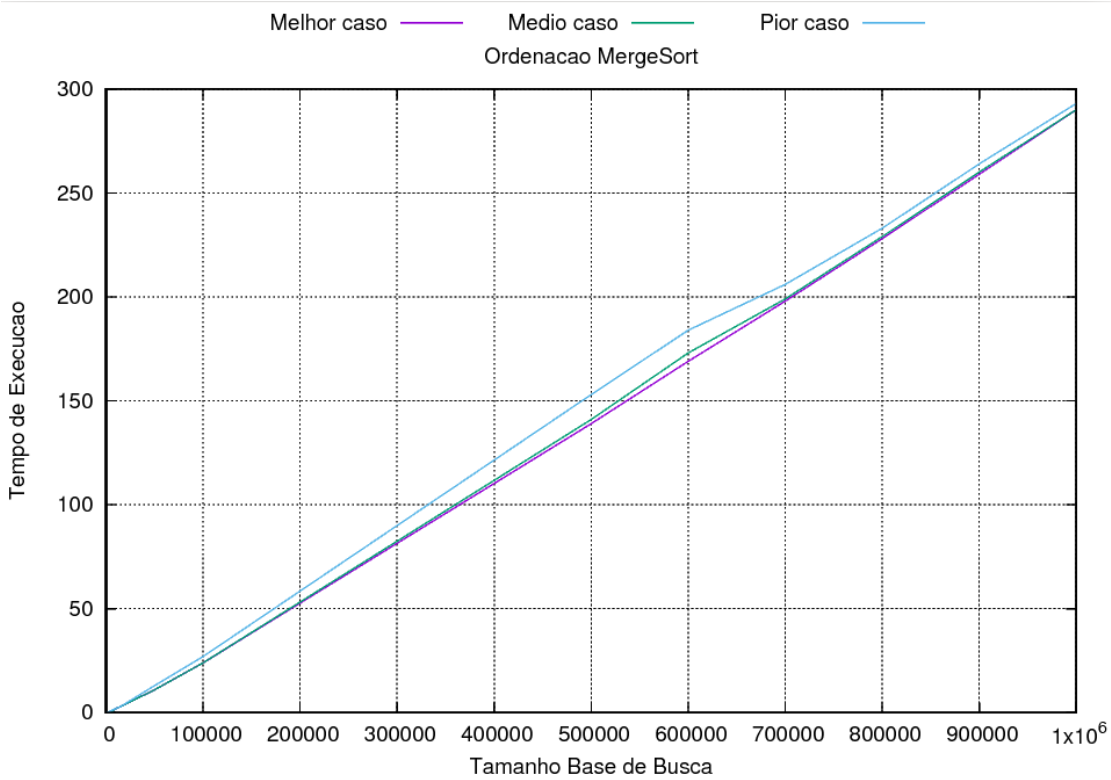

Gráfico 6 – Análise do algoritmo de ordenação mergeSort

4. ANÁLISE EMPÍRICA ALGORITMOS DE BUSCA

A análise foi realizada tendo como base os algoritmos de busca sequencial e binária, implentados de forma recursiva e iterativa. O projeto foi implementado utilizando a linguagem C++ e o conceito de boas práticas de programação, como o uso do makefile para compilação do projeto modularizado e o uso do padrão Doxygen de documentação.

4.1. Organização do projeto

O projeto foi implementado com modularização em arquivos. Os arquivos .cpp estão localizados na pasta /src. O arquivo buscas.cpp é composto pela implementação dos quatro algoritmos de busca, enquanto o main.cpp possui funções para gerar bases de dados aleatórias e ordenadas, funções para fazer a medição do tempo dos algortimos de busca - levando em consideração chaves de busca diferentes -, bem como a implementação de direcionar os dados para arquivos no formato txt.

4.2. Funcionamento do algoritmo

Ao executar o algoritmo, são gerados doze arquivos no formato txt. Grupos de tres arquivos
são referentes a um mesmo tipo de busca:

1 - binária recursiva;
2 - binária iterativa;
3 - sequencial recursiva;
4 - sequencial iterativa.

Cada um desses tres arquivos possui relação com uma chave de busca diferente, que foram
escolhidas de modo que:

1. Melhor caso sequencial: primeiro elemento do vetor da base de busca.
* base[0].
2. Pior caso sequencial e pior caso binária: valor superior ao último elemento do vetor de busca (elemento não pertencente ao vetor).
* base[tamBase-1]+1.
3. Melhor caso binária: elemento posicionado na metade do vetor.
* base[tamBase/2].

Dentro de cada um dos arquivos txt tem duas colunas com valores equivalentes ao tamanho da base de busca e o tempo de execução. Cada algoritmo de busca é executado com 14 tamanhos diferentes de base de busca = { 100, 500, 1000, 5000, 10000, 20000, 30000, 40000, 50000, 60000,
70000, 80000, 90000, 100000 } e o tempo de execução de cada algoritmo é medido por meio da função clock(), da biblioteca time.h.

A análise do algoritmo foi feita por meio de gráficos gerados através da ferramenta científica gnuplot. Os gráficos gerados possuem as coordenadas x e y representadas, respectivamente, como tamanho da base de busca e o tempo de execução dos algoritmos. Foram gerados quatro gráficos, cada qual equivalente a um tipo de busca.

4.3. Gráficos para a análise empírica

A busca sequencial iterativa (Gráfico 1), independente do tamanho da base de busca, permaneceu com um tempo de execução muito próximo de 0 quando a chave de busca foi equivalente ao primeiro valor da base de dados, sendo, portanto, o melhor caso deste algoritmo. Quando a chave de busca foi alterada para o elemento do meio do vetor da base, o tempo de execução permaneceu na média para a faixa de valores de tamanho de busca analisada. Ao executar a busca procurando um valor não pertencente ao vetor, o algoritmo teve o pior desempenho, sendo caracterizado, portanto, como pior caso desta busca.

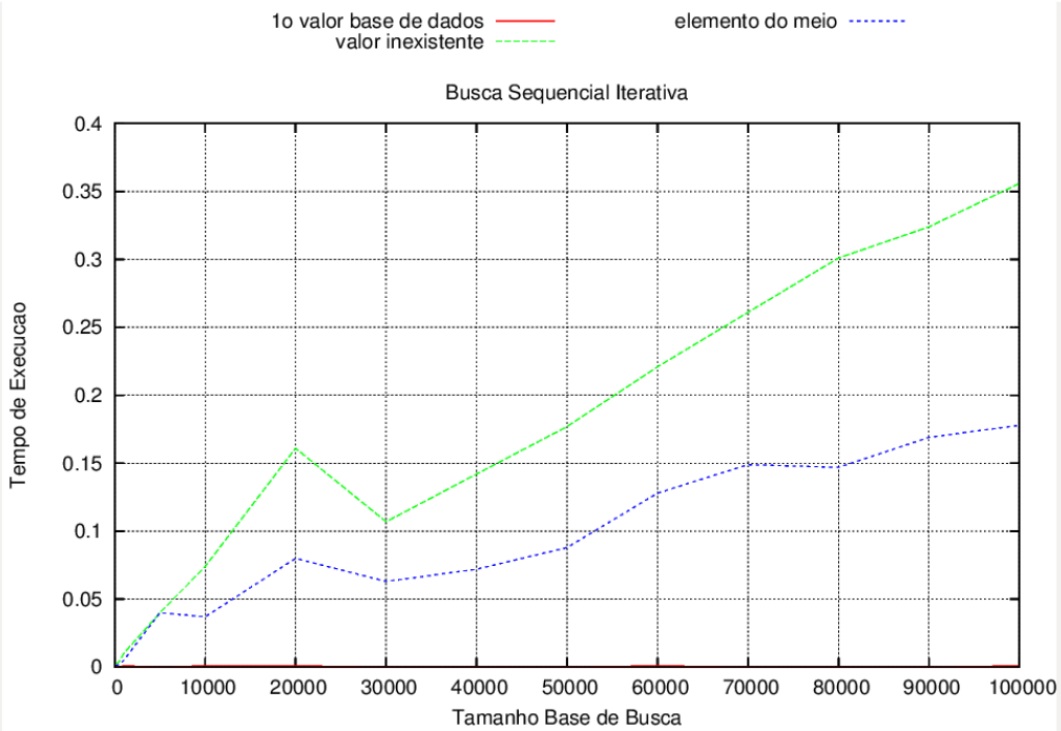

Gráfico 1 – Análise da busca sequencial iterativa

A busca sequencial recursiva (Gráfico 2) teve um desempenho semelhante à primeira. Para as chaves de busca analisadas, o pior, melhor e médio caso permaneceram os mesmos, alterando apenas os valores de tempo de execução.

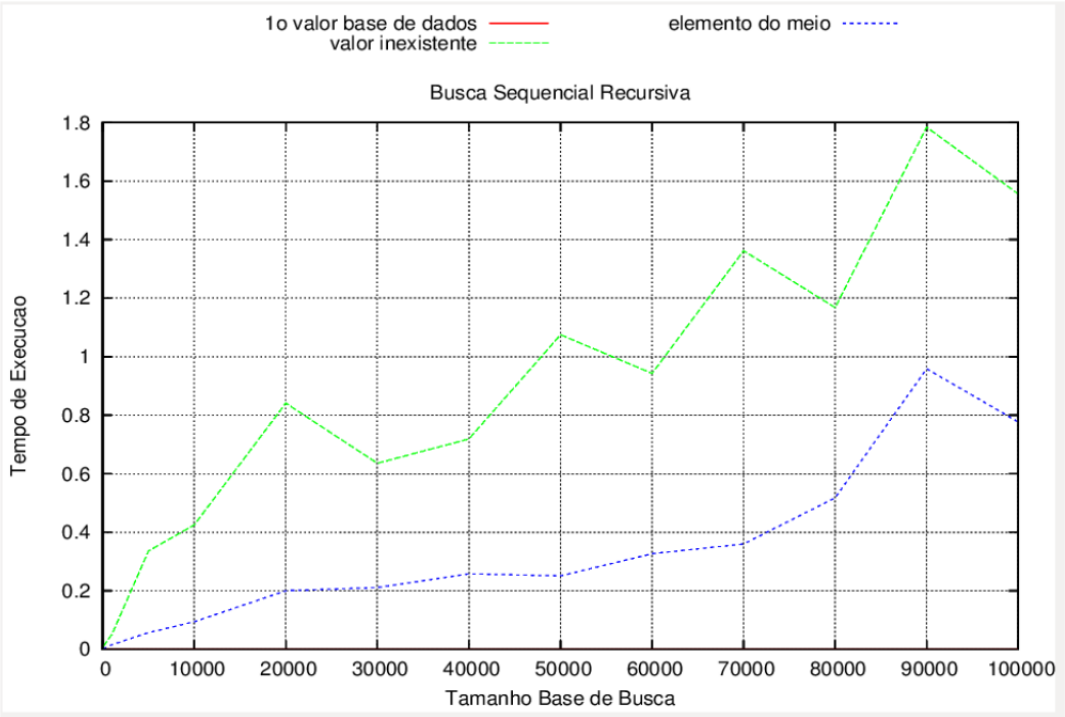

Gráfico 2 – Análise da busca sequencial recursiva

A busca binária recursiva (Gráfico 3) permaneceu com o menor tempo de execução quando a chave de busca foi equivalente ao elemento do meio da base de dados, sendo, portanto, o melhor caso deste algoritmo. Quando a chave de busca foi alterada para o primeiro elemento do vetor da base, o tempo de execução permaneceu na média para a faixa de valores de tamanho de busca analisada. Ao executar a busca procurando um valor não pertencente ao vetor, o algoritmo teve o pior desempenho, sendo caracterizado, portanto, como pior caso desta busca.

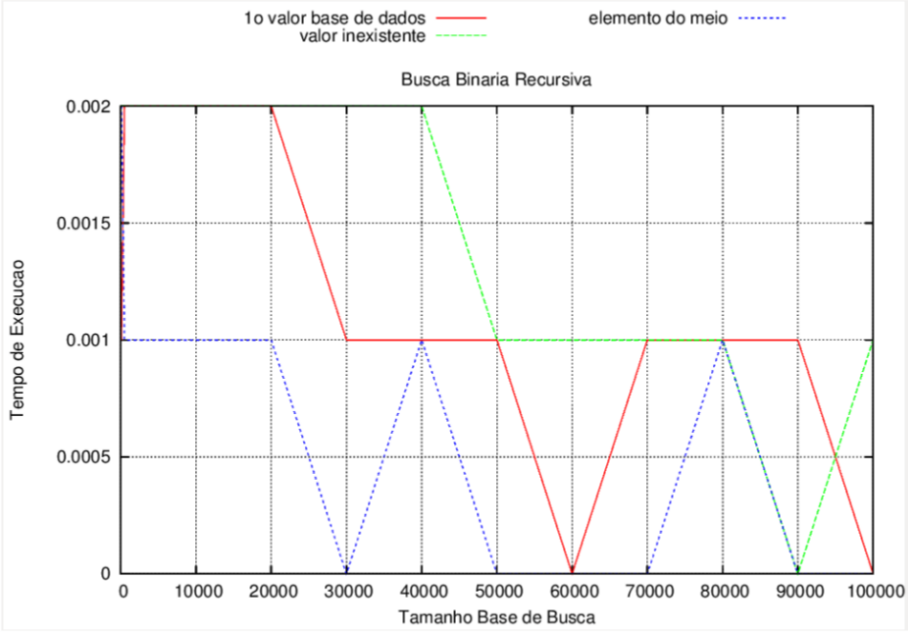

Gráfico 3 – Análise da busca binária recursiva

A busca binária iterativa (Gráfico 4) teve um desempenho semelhante à primeira. Para as chaves de busca analisadas, o pior, melhor e médio caso permaneceram os mesmos, alterando apenas os valores de tempo de execução.

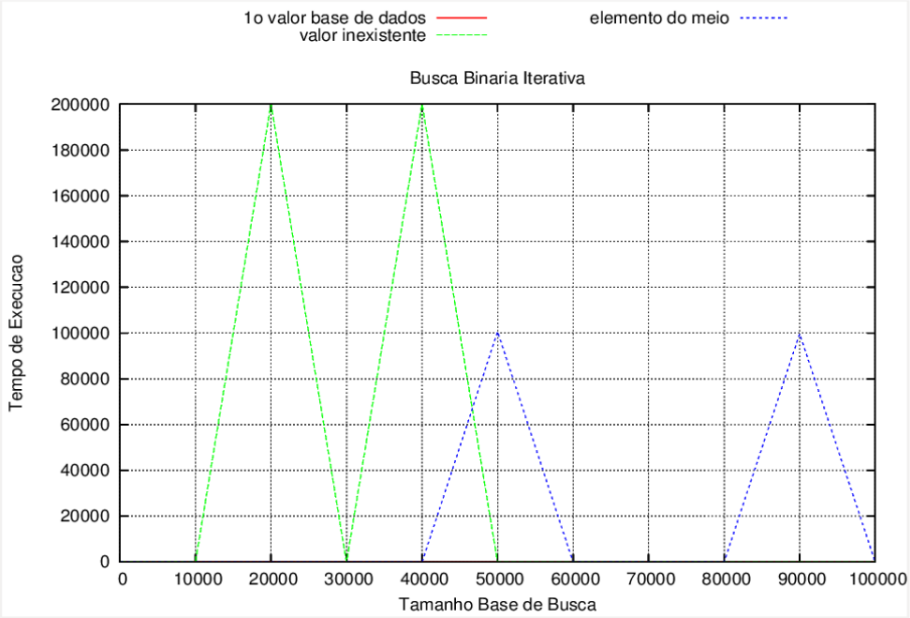

Gráfico 4 – Análise da busca binária iterativa

5. CONSIDERAÇÕES FINAIS

As análises empíricas apresentaram resultados condizentes com a complexidade dos respectivos algoritmos no pior, melhor e médio caso, em função do tamanho do vetor. Desta forma, constatou-se na prática e por experimentação o que foi aprendido de forma teórica ao longo das aulas.