UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE DIGITAL

RELATÓRIO DE ANÁLISE EMPÍRICA

NATAL/RN
2020

UNIVERSIDADE FEDERAL DO RIO GRANDE DO NORTE
INSTITUTO METRÓPOLE

DIEGO FILGUEIRAS BALDERRAMA
PAULO VANZOLINI MOURA DA SILVA
VINICIUS OLIVEIRA DA SILVA

RELATÓRIO DE ANÁLISE EMPÍRICA

Trabalho referente à nota parcial da Unidade I da disciplina DIM0119 - Estrutura de Dados Básica I - T02, orientado pelo Prof. Mr. Guilherme Fernandes de Araújo.

NATAL/RN
2020

SUMÁRIO

1. INTRODUÇÃO

A análise empírica de algoritmos, também conhecida como *profiling* ou análise de desempenho, consiste em definir critérios de comparação, sejam eles tempo ou uso de memória, entre dois ou mais algoritmos. Podemos determinar qual é mais eficiente, qual oferece uma solução mais simplista e suas complexidades a partir uma grande bateria de testes com tamanhos variados de entrada e diferentes dados. Um dos tipos de análise de algoritmos é a de complexidade pessimista que testa os algoritmos no seu pior caso, para assim conseguirmos diferenciar um algoritmo mais eficiente, em termos de tempo, de um menos eficiente.

Durante a verificação dos resultados, também é possível comparar os gráficos gerados e ter uma noção do comportamento dos algoritmos estudados em diferentes ocasiões, principalmente nos casos em que as entradas foram maiores, que indica a tendência dos algoritmos para entradas ainda maiores, visto que os recursos tempo e memória são limitados e não podemos definir seus desempenhos para todos os casos com exatidão.

2. DESENVOLVIMENTO

Neste trabalho, a comparação foi feita nos algoritmos de busca linear e busca binária baseada nos seus tempos de execução (em milissegundos) com diferentes tamanhos de entrada N.

Os algoritmos de busca foram implementados na linguagem C++ utilizando o compilador g++ (9.3.0) e a IDE Microsoft Visual Studio Code (1.49.0). Além disso, foram utilizadas as ferramentas Git e GitHub para versionamento de código e desenvolvimento remoto entre os integrantes do grupo. O repositório no GitHub pode ser encontrado através do link https://github.com/viniciu21/Empirical_analysis.

Como objetivo era medir a complexidade pessimista (pior caso), os valores das amostras foram todos preenchidos com o número 0 e definimos o valor buscado como 1, assim o código seria executado o máximo de vezes possíveis de acordo com o tamanho de N.

Para os dois algoritmos, o N variou de 50 a 100.000, aumentando de 50 em 50, e para cada N foram feitos 100 testes para cada busca, nos quais foram aplicados uma média aritmética na tentativa de eliminar os picos de processamento paralelo e resultados mais consistentes e homogêneos.

O algoritmo de busca linear foi implementada da seguinte forma:

```
int *lsearch(int *first, int *last, short target){

    while (first != last){  // Travel the array
        if (target == (*first)) // Found the target
            return first;   // Returns the element pointer

        first++;
    }

    return last;    // Failed
};
```

Já o algoritmo de busca binária foi implementado como exibido abaixo:

```
int *bsearch(int *first, int *last, short target){

    int count = last - first;   // Holds the array's elements' number

    while (count != 0){ // Keeps running across the array while the array is not done

        int step{count / 2};    // Evaluates the step value
        int *mid = first + step;    // Evaluates the middle element pointer

        if (target == (*mid))   // Found the target
            return mid; // The function is done and it returns a pointer to the element

        else{   // The target wasn't found
            if (target < (*mid))    // The target is on the left hand side of the array
                count = step;   // Resizes the current array turning it into the left half sub array

            else{   // The target is on the right hand side of the array
                first = mid + 1;    // Resizes the current array turning it into the right half sub array
                count = count - step - 1;   // Updates the number of elements of the array
            }
        }
    }
    return last; // Failed
};
```

Para a posterior geração dos gráficos, os resultados dos testes foram processados e armazenados em um arquivo de texto para cada algoritmo.

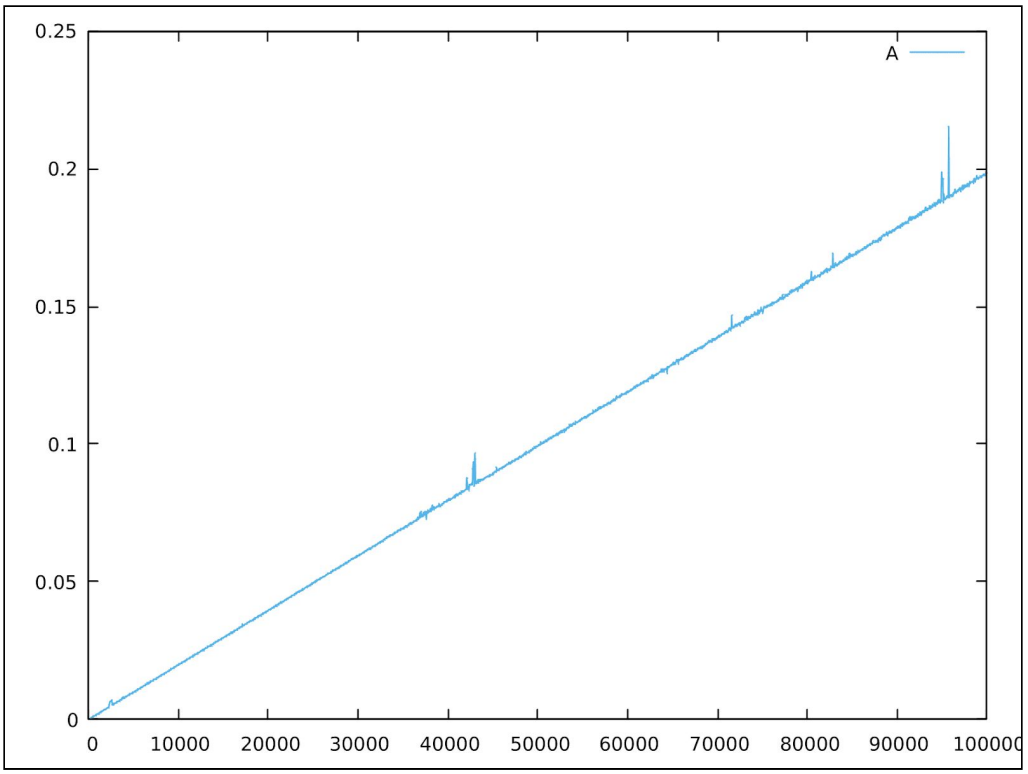
3. RESULTADOS E CONSIDERAÇÕES FINAIS

Para rodar o programa e gerar os gráficos foi utilizado um computador com as seguintes especificações e ferramentas:

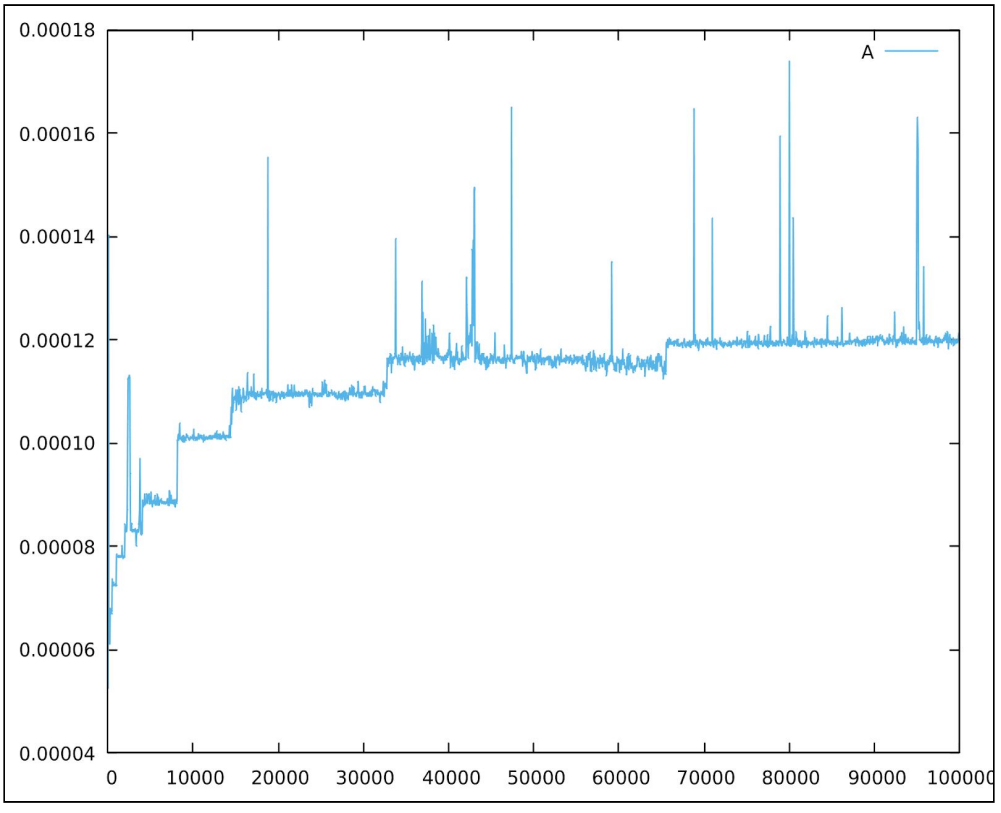
- Sistema Operacional: Ubuntu 20.04.
- Processador: AMD Ryzen 5 2600 6C/12T 3.4 GHZ.
- Memória RAM: 8GB 2400Mhz.
- Arquitetura: x86_64.
- Linguagem: C++11.
- Compilador: G++ 9.3.0.
- Gnuplot: 5.2.8

Como a busca linear passa de elemento por elemento do vetor, o tempo de execução vai ser bastante influenciado pelo tamanho do vetor, no pior caso o tempo vai aumentar conforme o tamanho do vetor de dados aumenta. Já que, se temos um vetor de tamanho N, o pior caso seria o alvo da busca não ser encontrado, o programa faria N iterações. Dessa forma, a variação do gráfico da busca linear no

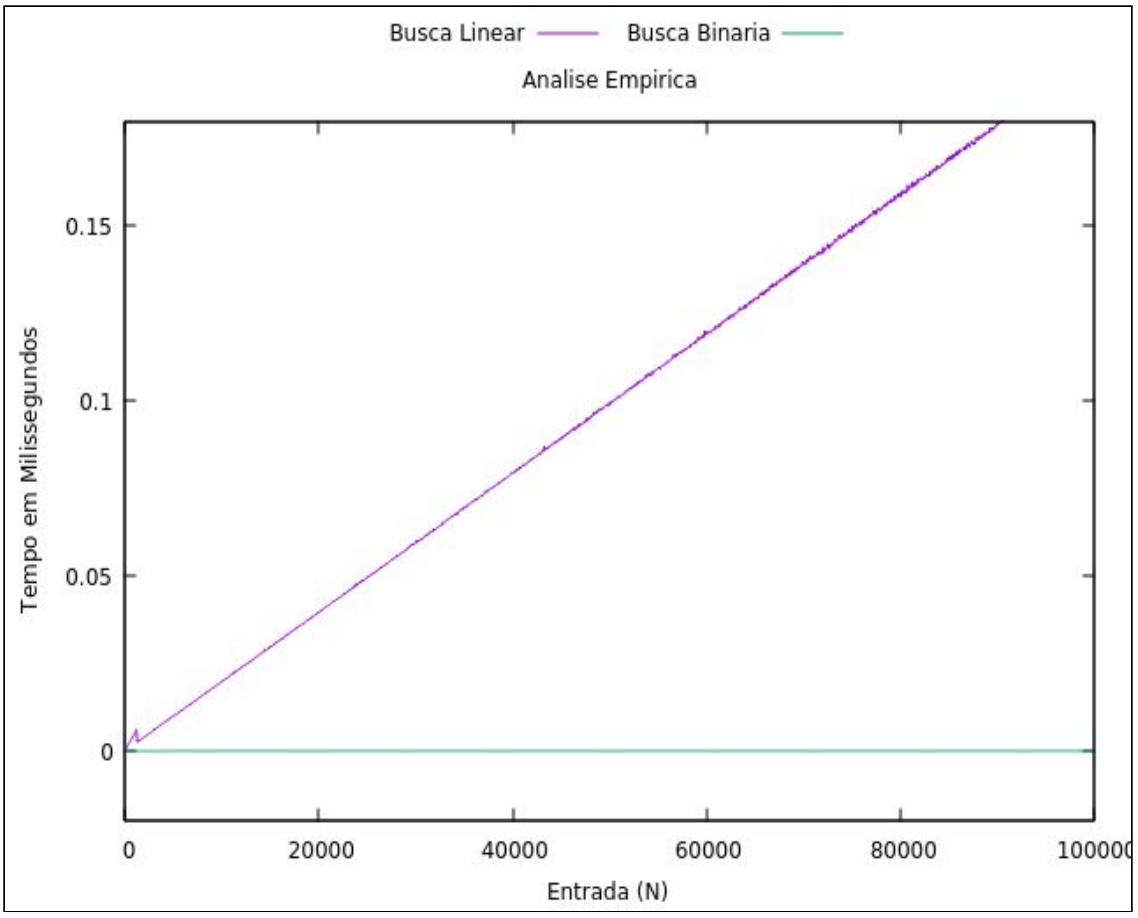
pior dos casos é de 1, ao aumentar o tamanho dos dados, o tempo de execução também aumentará proporcionalmente como se observou no gráfico a seguir: o eixo X representa o tamanho da entrada (N) e o eixo Y o tempo (ms).



Já na busca binária, o tempo de execução vai ser pouco influenciado pelo tamanho do vetor como na busca linear, já que a cada iteração metade dos dados é eliminado, ao invés de eliminar apenas um dado como na busca linear. Por isso, o crescimento do gráfico da busca binária do tempo de execução (ms) pelo tamanho da entrada (N) é logarítmico. Observe o gráfico abaixo:



No gráfico seguinte, observa-se a diferença entre o tempo de execução das buscas linear e binária:



Portanto, é possível observar que a busca binária possui clara vantagem sobre a busca linear no pior caso, porém, vale ressaltar que para realizar a busca binária os dados devem estar distribuídos em ordem não decrescente, caso contrário, ela não será executada corretamente.