Instituto de Ciências Matemáticas e de Computação
Departamento de Ciências de Computação
SCC0503 - Algoritmos e Estruturas de Dados II

Relatório Exercício 03

Alunos: Wagnysson Moura Luz
Professor: Leonardo Tórtoro Pereira

Outubro
2022

Conteúdo

1 Introdução **1**

2 Desenvolvimento **2**
2.1 Busca Sequencial 2
2.2 Busca Binária 2
2.3 Busca Binária Recursiva 3
2.4 Inversão de Vetor 4

3 Resultados **6**
3.1 Busca Sequencial 6
3.2 Busca Binária 7
3.3 Busca Binária Recursiva 8
3.4 Inversão de Vetor 9

1 Introdução

O exercício 3 teve como objetivo testar o tempo de execução e analisar a complexidade de quatro algoritmos diferentes em seus piores casos, sendo eles, um de busca sequencial, um de busca binária, um de busca binária recursiva e um de inverter um vetor dado. Para cada um, foi preciso utilizar os arquivos e um programa de contagem de operações fornecidos pelo professor. Por fim, com os dados obtidos foram construídos gráficos para comparar os resultados relacionados ao tempo de execução dos diferentes algoritmos e para auxiliar na análise de complexidade.

2 Desenvolvimento

Os quatro algoritmos foram testados com 6 vetores aleatórios gerados pelo programa fornecido pelo professor e que continham 10, 100, 1000, 5000, 10000 e 100000 números inteiros. Para cada vetor foram testadas 100 execuções, sempre no pior caso, para obter o tempo médio de execução. O programa do professor gerava um arquivo contendo os tamanhos dos vetores testados e o tempo de execução dos programas para cada tamanho de vetor.

Todos os vetores utilizados foram ordenados com Bubble Sort antes de serem executados os algoritmos. Isto era necessário apenas para os algoritmos de busca binária e busca binária recursiva, não faria diferença nos outros dois, pelo menos não nos aspectos analisados neste exercício.

2.1 Busca Sequencial

Neste algoritmo é dado um vetor, o tamanho desse vetor e um elemento a ser buscado como argumentos para a função de busca sequencial, a qual percorre o vetor dado comparando o elemento buscado com todos os presentes no vetor de forma sequencial.

Segue imagem da implementação.

```c
#include <stdio.h>
#include <stdlib.h>

int busca_sequencial(int *vetor, int tamanho, char elementoProcurado) {
    int i;

    for (i = 0; i < tamanho; i++) {
        if (vetor[i] == elementoProcurado) {
            return i;
        }
    }

    return -1;
}
```

Figura 1: Algoritmo de busca sequencial implementado para o exercício em questão

Seu pior caso é procurar um elemento que se encontra na última posição do vetor.

2.2 Busca Binária

Para este algoritmo é passado um elemento a ser buscado em um vetor, o vetor em questão e o tamanho do vetor. Após ordenar o vetor (operação

realizada antes de calcular o tempo de execução), a função utiliza estes argumentos para checar se o elemento do meio do vetor é o elemento buscado, caso não seja, verifica se ele é maior que este elemento do meio, se for, o elemento do meio e os menores que ele são ignorados e a busca é feita novamente no restante do vetor checando o elemento do meio e assim até que o elemento buscado seja encontrado, ou não. O processo é análogo para o caso do elemento buscado ser menor que o elemento do meio do vetor.

Segue imagem da implementação.

```c
#include <stdio.h>
#include <stdlib.h>

int busca_binaria(int elem_buscado, int *vetor, int tamanho){
    int inicio = 0, meio, final;

    final = tamanho-1;

    do{
        meio = (inicio+final)/2;
        if (elem_buscado == vetor[meio]){
            return meio;
        }else{
            if(elem_buscado>vetor[meio]){
                inicio = meio+1;
            }else{
                final = meio-1;
            }
        }


    }while(inicio <= final);

    return -1;
}
```

Figura 2: Algoritmo de busca binária implementado para o exercício em questão

O pior caso desse algoritmo é quando o elemento procurado está fora do vetor.

2.3 Busca Binária Recursiva

Neste algoritmo é passado como argumentos o elemento buscado, o vetor, o índice do primeiro elemento e o índice do último elemento. Ele realiza as mesmas verificações de condição que o algoritmo de busca binária simples, mas em vez de haver uma estrututra while/do-while que é executada até

encontrar o elemento buscado, o algoritmo chama a função própria função novamente alterando os indices do início e ou do fim do vetor utilizando o valor da variável "meio"até encontrar o elemento buscado.

Segue imagem da implementação

```c
#include <stdio.h>
#include <stdlib.h>

int busca_binaria_rec (int elem_buscado, int vetor[], int e, int d)
{
  int meio = (e + d)/2;
  if (vetor[meio] == elem_buscado)
    return meio;
  if (e >= d)
    return -1; // não encontrado
  else
    if (vetor[meio] < elem_buscado)
      return busca_binaria_rec(elem_buscado, vetor, meio+1, d);
    else
      return busca_binaria_rec(elem_buscado, vetor, e, meio-1);
}
```

Figura 3: Algoritmo de busca binária recursiva implementado para o exercício em questão

O pior caso desse algoritmo é o mesmo que o da busca binária.

2.4 Inversão de Vetor

Neste algoritmo, a função de inverter um vetor recebe como argumentos um vetor e o tamanho dele. A partir daí ele inverte a posição do primeiro elemento com o último, do segundo com o penúltimo e assim por diante.

Segue imagem da implementação.

```c
#include <stdio.h>
#include <stdlib.h>

int *inverte_vetor(int *vetor, int tamanho){
    int i, temp, final;

    final = tamanho-1;

    for (i = 0; i < tamanho/2; i++){
        temp = vetor[i];
        vetor[i] = vetor[final-i];
        vetor[final-i] = temp;
    }
    return vetor;
}
```

Figura 4: Algoritmo de inverter um vetor implementado para o exercício em questão

O pior caso deste algoritmo depende do tamanho do vetor que ele irá inverter.

3 Resultados

3.1 Busca Sequencial

Ao realizar os testes com o programa fornecido pelo professor obteve-se os seguintes resultados:

Tamanho do Vetor	Tempo de Execução (em segundos)
10	0.000000
100	0.000001
1000	0.000002
5000	0.000010
10000	0.000019
100000	0.000182

Para melhor observação foi gerado um gráfico utilizando o GNUplot

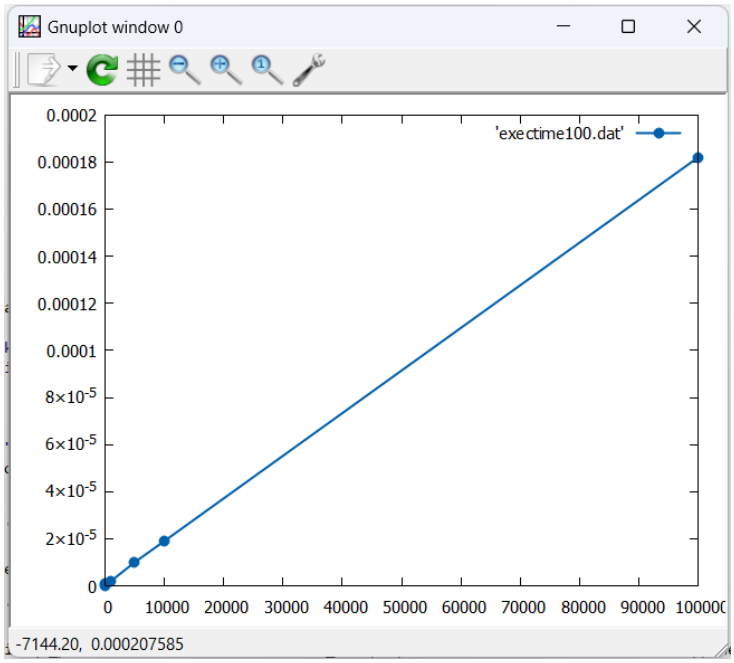

Figura 5: Gráfico Tamanho do vetor x Tempo de Execução (em seg)

Analisando as operações feitas no algoritmo temos apenas comparações entre números dois a dois, ou seja, no pior caso teremos n comparações, então podemos concluir que o algoritmo tem uma complexidade de $O(n)$ e para reforçar esse argumento vemos que o gráfico é linear.

3.2 Busca Binária

Com o programa e os arquivos fornecidos pelo professor não havia como saber que elemento estava dentro do vetor, logo não havia como saber qual estava fora, sendo assim, foi buscado o último elemento do vetor.

Ao realizar os testes obteve-se os seguintes resultados:

Tamanho do Vetor	Tempo de Execução (em segundos)
10	0.000000
100	0.000000
1000	0.000000
5000	0.000000
10000	0.000000
100000	0.000000

Plotando os dados no GNUplot temos:

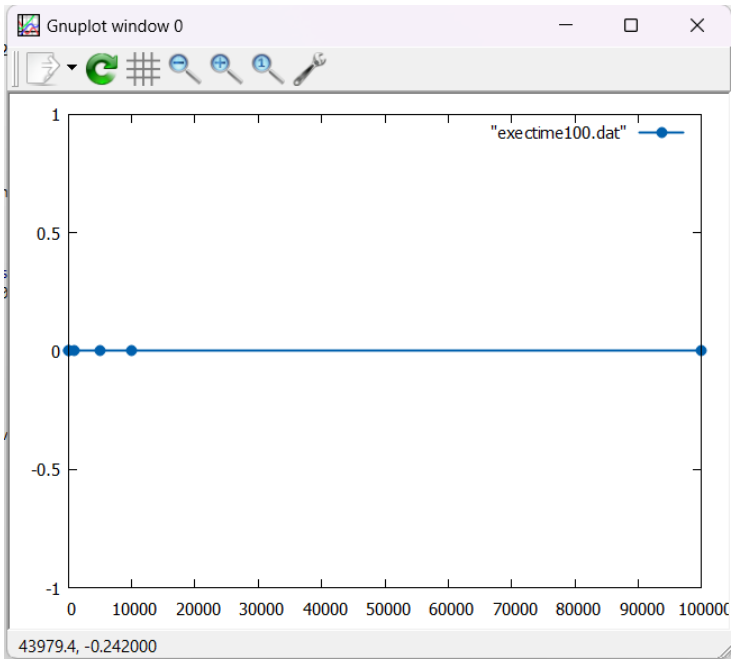

Figura 6: Gráfico tamanho do vetor x tempo de execução (em seg)

Os tempos de execução foram tão pequenos que não apareceram no nível de precisão utilizado, o que é esperado tendo em vista que esse algoritmo é mais rápido que o de busca sequencia.

A complexidade deste algoritmo, no pior caso, é $O(log_2 n)$, sendo n o tamanho do vetor dado como entrada e $log_2 n$ é a quantidade de operações

que serão realizadas até encontrar o elemento buscado. Isto se dá pois o vetor é partido ao meio e o que sobrou é partido ao meio e assim por diante, para encontrar o número de elementos inicial do vetor podemos multiplicar 2 por ele mesmo na quantidade de vezes que o vetor foi dividido (seja x este valor e n o tamanho inicial do vetor) e temos $2^x = n \Leftrightarrow log_2 n = x$.

3.3 Busca Binária Recursiva

Aqui é o mesmo caso da busca binária, foi utilizado o último elemento do vetor.

Ao realizar os testes obteve-se os seguintes resultados:

Tamanho do Vetor	Tempo de Execução (em segundos)
10	0.000000
100	0.000000
1000	0.000000
5000	0.000000
10000	0.000000
100000	0.000000

Plotando os dados no GNUplot temos:

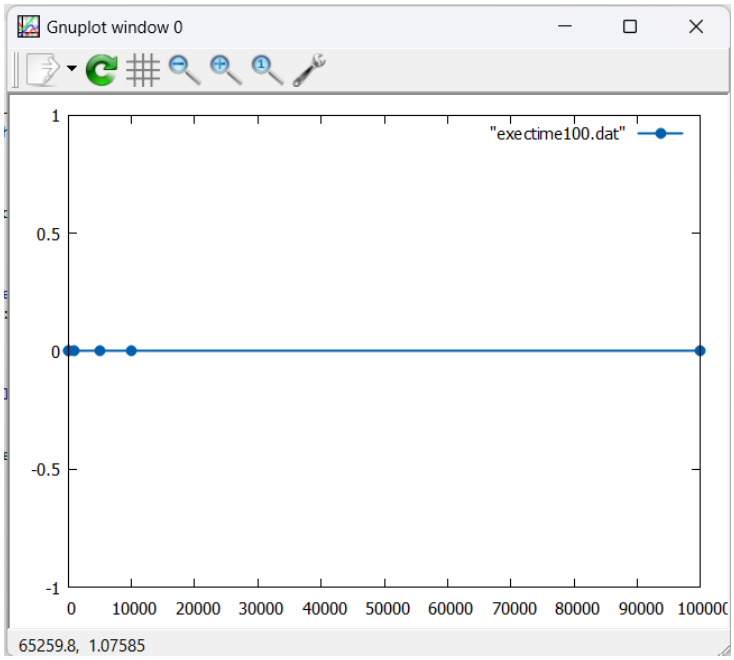

Figura 7: Gráfico tamanho do vetor x tempo de execução (em seg)

A complexidade aqui, no pior caso, é $O(log_2 n)$ assim como a busca recursiva.

3.4 Inversão de Vetor

Testando o tempo de execução da função obtivemos os seguintes valores

Tamanho do Vetor	Tempo de Execução (em segundos)
10	0.000000
100	0.000001
1000	0.000002
5000	0.000007
10000	0.000014
100000	0.000142

Segue o gráfico gerado com o GNUplot

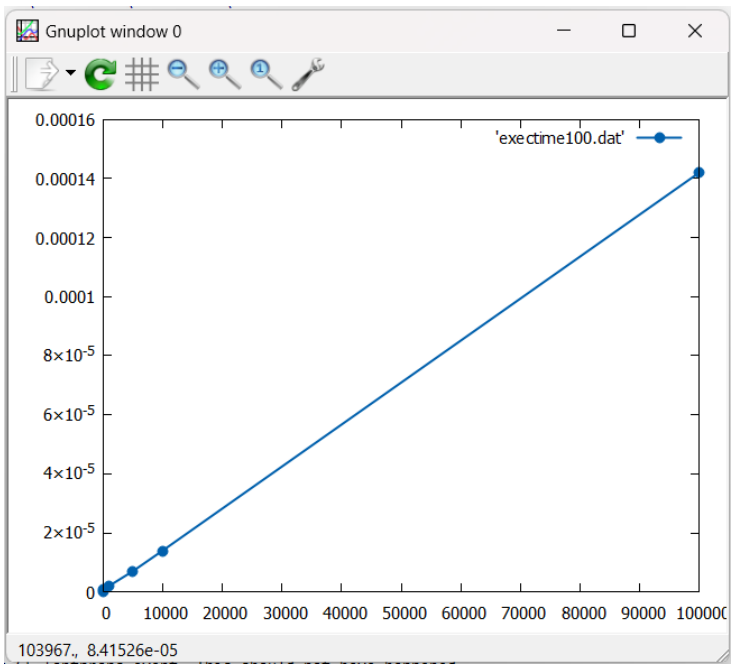

Figura 8: Gráfico Tamanho do vetor x Tempo de Execução (em seg)

Por ele consegue-se observar que a complexidade é linear, no pior caso, $O(n/2)$. Isto porque são feitas apenas $n/2$ comparações sendo n o tamanho do vetor e tendo em vista que a cada operação 2 elementos são colocados no lugar correto.