	SCC0223 – Estruturas de Dados – Bacharelado em Ciência de Dados		
	Nome: Rodrigo Borges Delai	**Nº USP**: 11849074	**Turma**: 201-2022
	Nome: –	**Nº USP**: –	**Professor**: Leonardo T. P.
	Nome: –	**Nº USP**: –	**Data**: 16/10/2022

Exercício 03 - Contagem de operações e de tempo de execução

1. OBJETIVOS

A execução desta exercício tem por objetivo verificar o eficiência e o comportamento de alguns algorítmos na prática e compará-los com seus modelos teóricos. Para isso, pretende-se contar o número de operações via análise do algoritmo e, em seguida, observar seu tempo de execução, conforme varia-se o número de entradas.

2. METODOLOGIA

A contagem das operações será feita tendo em vista o pior caso. Para facilitar a leitura, serão utilizadas as seguintes letras para identificar cada operação: aritméticas e atribuições serão denotadas pela letra “a”; comparações pela letra “c”; acesso ao ponteiro ou vetor “p”; e retorno ou chamada de função será um “r”.

Já para a realização dos ensaios computacionais, serão utilizados o código disponível no Anexo I e os algoritmos abaixo descritos.

3. ALGORITMO DE INVERSÃO

Este algoritmo inverte a ordem dos elementos de um vetor dado. A partir do código, podemos contabilizar o número de operações.

```
void swap(int *a, int *b){                                  // r
    int temp = *a;                                          // a + p
    *a = *b;                                                // a + 2p
    *b = temp;                                              // a + p
}

int reverseTheOrder(numberVector_t numberVector)            // r
{
    int vectorSize = numberVector.vectorSize;               // a + p
    int* vector = numberVector.numbers;                     // a + p

    for(int i = 0; i < vectorSize/2; ++i){      // 1) a + c; 2) 2a + c
      swap((vector + i),(vector + vectorSize - i));         // 3a
    }
    return 1;                                               // r
```

```
}
```

Sendo “n” o número de elementos no vetor fornecido, tem-se a seguinte contruibuição acumulada:

$$r + a + p + a + p + a + c + ((n/2) - 1) \cdot (2a + c) + (n/2) \cdot (3a + r + 3a)$$

$$\Rightarrow \quad (nr/2) + 4na + (nc/2) + r + a + 3p\,.$$

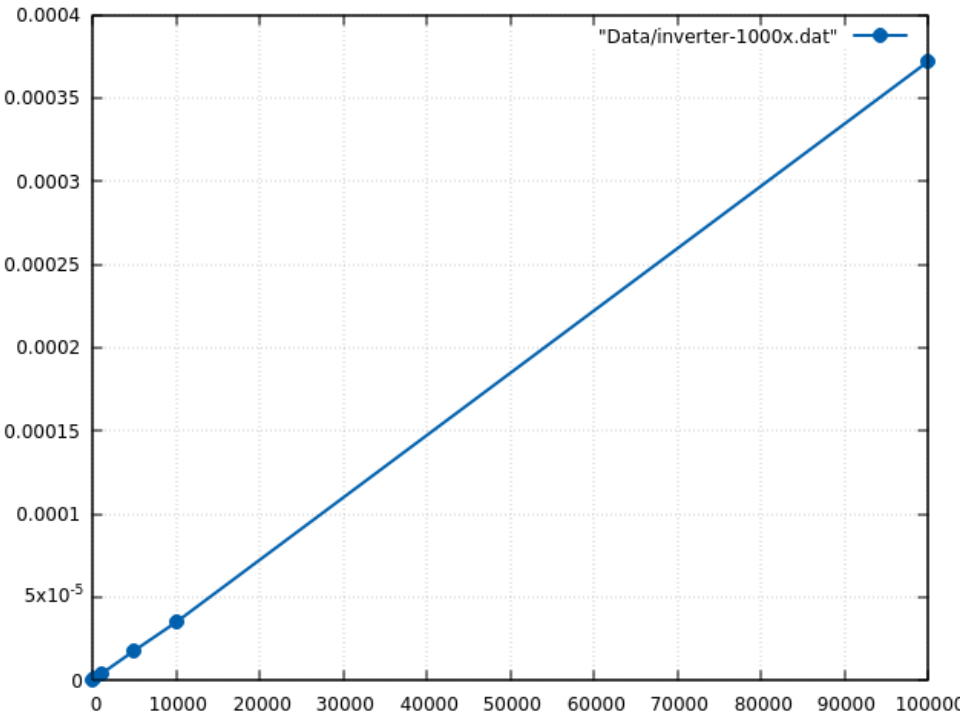
Supondo o custo computacional igual para as operações acima discriminadas $(r = a = c = p)$, tem-se:

$$(na/2) + 4na + (na/2) + a + a + 3a \quad \Rightarrow \quad 5na + 5a \quad \Rightarrow \quad 5a \cdot (n + 1)\,.$$

Portanto, pode-se concluir que o algoritmo de inversão possui complexidade linear, em termos de operações computacionais. Por isso, sua notação em Big O é O(n).

A partir de um ensaio computacional foi possível medir o tempo necessário para a execução do algoritmo em função do tamanho do vetor de entrada. Eis o resultado:

Figura 1 - Tempo de execução (em segundos) em função do número de elementos no vetor para o algoritmo de inversão.



Fonte: Elaborado pelo autor.

Assim, pôde-se verificar que, de fato, a relação entre o tempo de execução e o tamanho do vetor de entrada é linear.

3. ALGORITMO DE BUSCA SEQUENCIAL

Este algoritmo realiza uma busca sequencial em um vetor dado. A partir do código, podemos contabilizar o número de operações.

```
int LinearSearch(numberVector_t numberVector, int element)    // r
{
    int vectorSize = numberVector.vectorSize;                 // a + p

    for (int i = 0; i < vectorSize; ++i)        // 1) a + c; 2) 2a + c
        if (element == numberVector.numbers[i])               // c + p
            return i;                                         // r*

    return -1;                                                // r
}
```

Sendo “n” o número de elementos no vetor fornecido, tem-se a seguinte contruibuição acumulada:

$$r + a + p + a + c + (n-1)\cdot(2a + c) + n\cdot(c + p) + r$$

$$\Rightarrow\ 2an + np + 2nc + 2r + a + p\ .$$

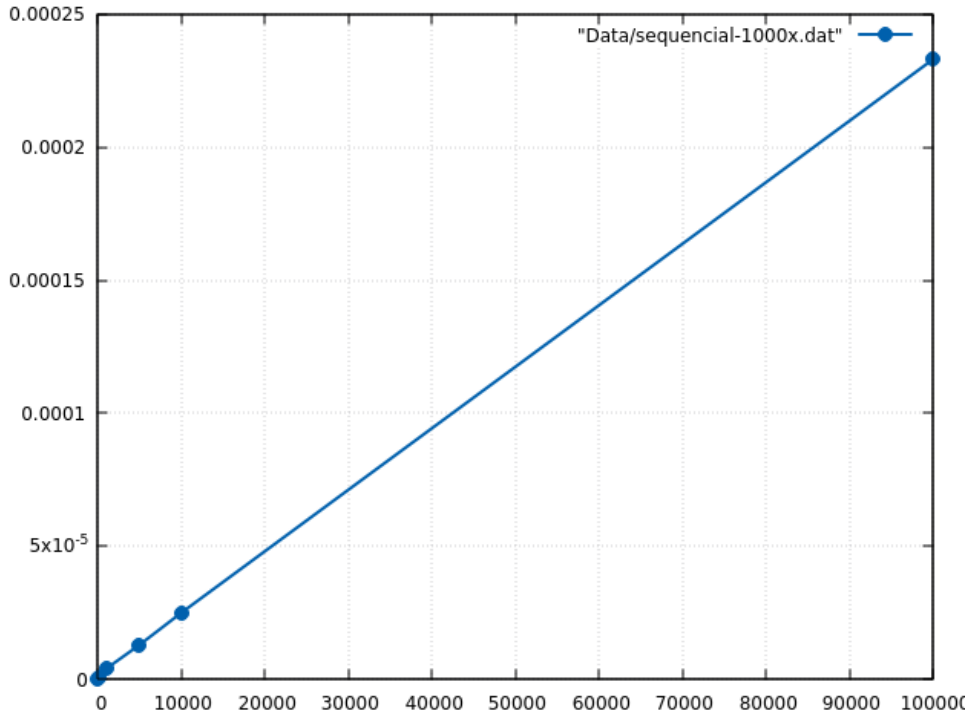
Supondo o custo computacional igual para as operações acima discriminadas $(r = a = c = p)$, tem-se:

$$2na + na + 2na + 2a + a + a \quad \Rightarrow \quad 5na + 4a \quad \Rightarrow \quad 5a\cdot(n + 4/5)\ .$$

Portanto, pode-se concluir que o algoritmo de busca sequencial possui complexidade linear, em termos de operações computacionais. Por isso, sua notação em Big O é O(n).

A partir de um ensaio computacional foi possível medir o tempo necessário para a execução do algoritmo em função do tamanho do vetor de entrada. Eis o resultado:

Figura 1 - Tempo de execução (em segundos) em função do número de elementos no vetor para o algoritmo de busca sequencial.



Fonte: Elaborado pelo autor.

Assim, pôde-se verificar que, de fato, a relação entre o tempo de execução e o tamanho do vetor de entrada é linear.

4. ALGORITMO DE BUSCA BINÁRIA (ITERATIVA)

Este algoritmo realiza uma busca binária com implementação iterativa em um vetor dado. Observe que, para aplicação do algoritmo, o vetor fornecido deverá ser ou estar ordenado. A partir do código, podemos contabilizar o número de operações.

```
int BinarySearch(numberVector_t numberVector, int element)    // r
{
    int high = numberVector.vectorSize - 1;                   // 2a + p
    int low = 0;                                              // a
    int mid;                                                  //
    int* vector = numberVector.numbers;                       // a + p

    while (high > low){                                       // c
        mid = (high + low) / 2;                               // 3a

        if (element == vector[mid]) return mid;           // c + p + r*
        else if (element > vector[mid]) low = mid + 1;    // c + p + 2a
        else high = mid - 1;                                  // 2a
    }
```

```
        return -1;                                                  // r
    }
```

Sendo “n” o número de elementos no vetor fornecido, tem-se a seguinte contruibuição acumulada:

$$r + 2a + a + a + p + log_2(n) \cdot (c + 3a + c + p + c + p + 2a + 2a) + r$$

$$\Rightarrow 2r + 4a + p + log_2(n) \cdot (3c + 7a + 2p) .$$

Supondo o custo computacional igual para as operações acima discriminadas $(r = a = c = p)$, tem-se:

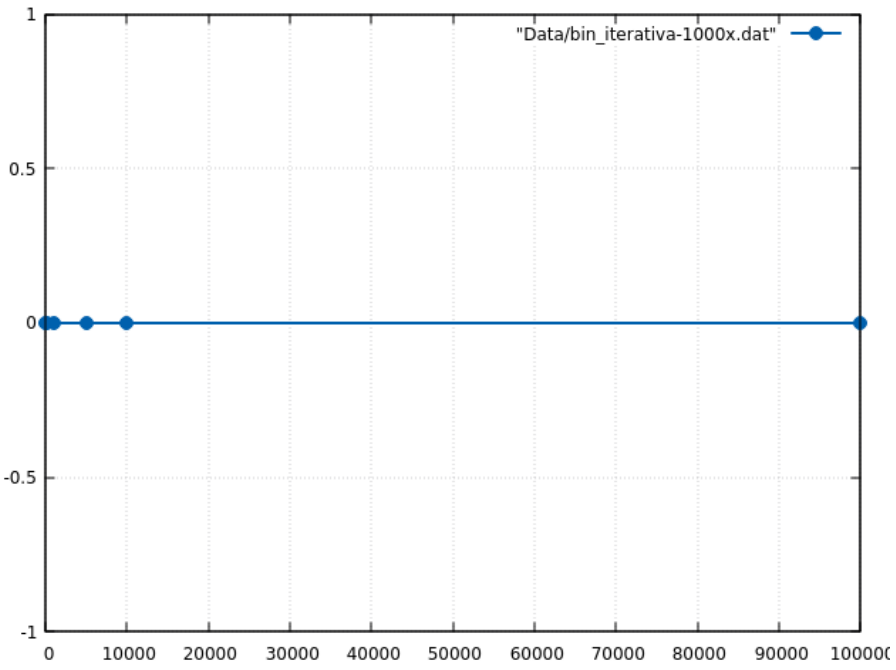
$$2a + 4a + a + log_2(n) \cdot (3a + 7a + 2a) \Rightarrow 12a \cdot log_2(n) + 7a$$

$$\Rightarrow 12a \cdot (log_2(n) + 7/12) .$$

Portanto, pode-se concluir que o algoritmo de busca binária possui complexidade logarítmica, em termos de operações computacionais. Por isso, sua notação em Big O é O(log(n)).

A partir de um ensaio computacional foi possível medir o tempo necessário para a execução do algoritmo em função do tamanho do vetor de entrada. Eis o resultado:

Figura 1 - Tempo de execução (em segundos) em função do número de elementos no vetor para o algoritmo de busca binária iterativa.



Fonte: Elaborado pelo autor.

Como esperado, mesmo com um vetor munido de 10^5 elementos, o algoritmo precisaria de algo na ordem de $log_2(10^5) < 17$ operações para completar a busca. Por isso, como o clock do computador em questão é da ordem de 2.8GHz, não foi possível precisar em segundos o tempo de execução para esta busca.

4. ALGORITMO DE BUSCA BINÁRIA (RECURSIVA)

Este algoritmo realiza uma busca binária com implementação iterativa em um vetor dado. Observe que, para aplicação do algoritmo, o vetor fornecido deverá ser ou estar ordenado. A partir do código, podemos contabilizar o número de operações.

```
int BinarySearch(int* vector, int element, int high, int low) // r
{
    if (high < low) return -1;                                  // c + r

    int mid = (high + low) / 2;                                 // 3a

    if (element == vector[mid]) return mid;                     // c + p + r*
      else if (element > vector[mid]) BinarySearch(vector, element,
high, mid + 1);                                                 // c + p + r + a
    else BinarySearch(vector, element, mid - 1, low); // r + a
}
```

Sendo “n” o número de elementos no vetor fornecido, tem-se a seguinte contruibuição acumulada:

$$r + log_2(n) \cdot (c + r + 3a + c + p + c + p + r + a)$$

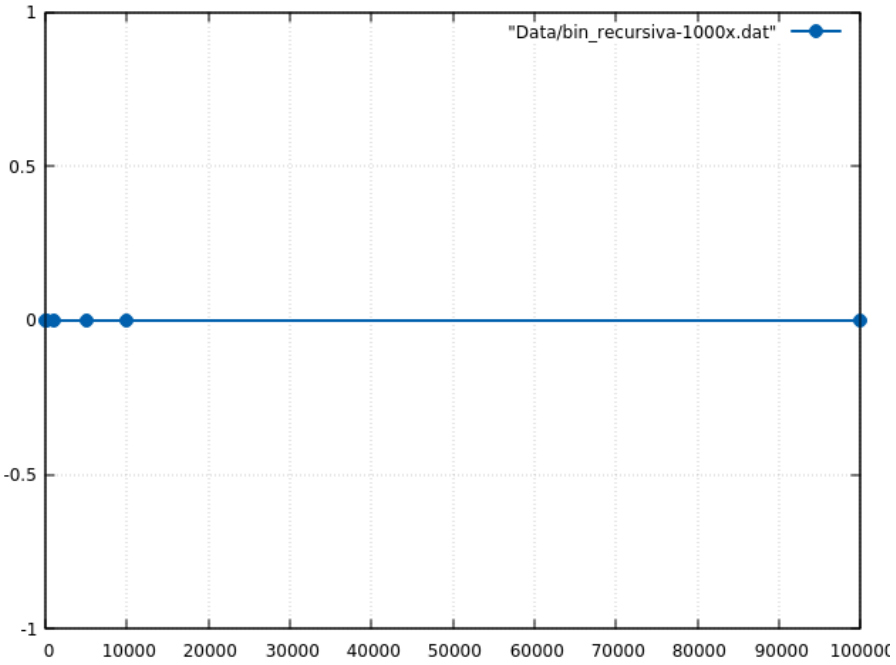
Supondo o custo computacional igual para as operações acima discriminadas $(r = a = c = p)$, tem-se:

$$a + log_2(n) \cdot 11a$$

Portanto, pode-se concluir que o algoritmo de busca binária possui complexidade logarítmica, em termos de operações computacionais. Por isso, sua notação em Big O é O(log(n)).

A partir de um ensaio computacional foi possível medir o tempo necessário para a execução do algoritmo em função do tamanho do vetor de entrada. Eis o resultado:

Figura 1 - Tempo de execução (em segundos) em função do número de elementos no vetor para o algoritmo de busca binária recursiva.



Fonte: Elaborado pelo autor.

Como esperado, mesmo com um vetor munido de 10^5 elementos, o algoritmo precisaria de algo na ordem de $log_2(10^5) < 17$ operações para completar a busca. Por isso, como o clock do computador em questão é da ordem de 2.8GHz, não foi possível precisar em segundos o tempo de execução para esta busca.