Algoritmos e Estrutura de Dados I

EXERCÍCIOS

Prof. Dra. Aline de Campos

Exemplo prático

1

```
public static int somavetor(int[] v){
    int soma = 0;
    for(int i=0; i<v.length; i++){
        soma = soma + v[i];
    }
    return soma;
}
```

Exemplo prático

1

```
public static int somavetor(int[] v) {
    int soma = 0;                          1
    for(int i=0; i<v.length; i++) {        1   n + 1   n
        soma = soma + v[i];                n
    }
    return soma;                           1
}
```

1 + 1 + n + 1 + n + n + 1 → 3n + 4 → O(n)

Análise passo a passo do tempo de execução do algoritmo, levando em conta cada instrução e somando os custos para chegar à complexidade final O(n)

1

Exemplo prático

- Algoritmo soma os elementos de um vetor

- O código acima é uma função que recebe um vetor de inteiros ‘ **v** ‘e retorna a soma de seus elementos

1) Análise de Complexidade:

- `` `int soma = 0;` `` (linha 2): Esta linha executa uma vez, então o custo é `` `1` ``.
- `` `for(int i = 0; i < v.length; i++)` `` (linha 3): O laço `` `for` `` tem três componentes:
  - `` `int i = 0;` `` executa uma vez no início, custo `` `1` ``.
  - `` `i < v.length;` `` é a condição de parada que é verificada `` `n + 1` `` vezes (onde `` `n` `` é o comprimento do vetor).
  - `` `i++` `` executa `` `n` `` vezes.
- `` `soma = soma + v[i];` `` (linha 4): Esta linha é executada `` `n` `` vezes, uma para cada elemento do vetor.
- `` `return soma;` `` (linha 6): Esta linha executa uma vez, então o custo é `` `1` ``.

1

Exemplo prático

- Algoritmo soma os elementos de um vetor

- O código acima é uma função que recebe um vetor de inteiros ‘ **v** ‘e retorna a soma de seus elementos

2) Somando os custos:

- Linha 2: `1`
- Linha 3: `1 + (n + 1) + n`
- Linha 4: `n`
- Linha 6: `1`

Somando tudo: `1 + 1 + (n + 1) + n + n + 1 = 3n + 4`.

3) Notação Big-O:

- A expressão `3n + 4` é simplificada na notação Big-O para `O(n)`, pois, em termos de complexidade, constantes e termos de ordem inferior não são considerados. Assim, o algoritmo tem complexidade linear.

Exercícios

2

```
public static boolean buscarvetor(int[] v, int e) {
    for(int i=0; i<v.length; i++) {
        if(v[i] == e)
            return true;
    }
    return false;
}
```

Exercícios

2

```
public static boolean buscarvetor(int[] v, int e) {
    for(int i=0; i<v.length; i++) {        1   n + 1   n
        if(v[i] == e)                      n
            return true;
    }
    return false;                          1
}
```

int i = 0 (executado 1 vez).
i < v.length (executado n + 1 vezes no pior caso).
v[i] == e (ocorre no máximo n vezes)
i++ (ocorre n vezes)
return false; (ocorre 1 vez no pior caso - quando e não está no vetor).

1 + n + 1 + n + n + 1

3n + 3

O(n)

Exercícios

3

```
public static boolean buscarmatriz(int[][] m, int e){
    for(int i=0; i<mat.length; i++) {
        for(int j=0; j<mat[i].length; j++) {
            if (mat[i][j] == elem)
                return true;
        }
    }
    return false;
}
```

Exercícios

3

```
public static boolean buscarmatriz(int[][] m, int e){
    for(int i=0; i<mat.length; i++) {
        for(int j=0; j<mat[i].length; j++) {
            if (mat[i][j] == elem)
                return true;
        }
    }
    return false;
}
```

1 + n + 1 + n + n + n . (n + 1) + n . n + n . n + 1

2 + 3n

$n^2 + n$

$n^2 + n^2$

1

$3n^2 + 4n + 3$

$O(n^2)$

int i=0
int j=0 Para cada linha i, percorre todos os elementos daquela linha. Se a matriz tem n linhas e n colunas (matriz quadrada), esse loop percorre n colunas, ou seja, n vezes para cada i.
i < m.length e j < m[i].length) ocorrem n + 1 vezes para i e aproximadamente n(n + 1) vezes para j.
m[i][j] == e ocorre até n^2 vezes no pior caso.
j++ ocorre n^2 vezes no pior caso.
return false ocorre 1 vez no pior caso (quando e não está na matriz).

Exercícios

4

```
public static int encontrarmin(int[] v) {
    int min = v[0];
    for(int i=1; i<v.length; i++) {
        if(v[i] < min)
            min = v[i];
    }
    return min;
}
```

Exercícios

4

```
public static int encontrarmin(int[] v) {
    int min = v[0];
    for(int i=1; i<v.length; i++) {
        if(v[i] < min)
            min = v[i];
    }
    return min;
}
```

1 (int min = v[0];)

1 (for(int i=1;) | n (i<v.length;) | n - 1 (i++))

n - 1 (if(v[i] < min))

n - 1 (min = v[i];)

1 (return min;)

1 + 1 + n + n - 1 + n - 1 + n - 1 + 1

4n + 3 - 3

4n

O(n)

int min = v[0]; (1 vez).
for (i = 1 até n-1)
i < v.length (n vezes)
i++ (n - 1 vezes – pq o for começa em 1 ao invés de 0)
v[i] < min (n - 1 vezes)
min = v[i] (apenas se v[i] for menor que min) (no pior caso, n - 1 vezes)
return min; (1 vez)

Exercícios

5

```java
public static int mediana(int[] v) {
    int mediana, meio;
    Arrays.sort(v);
    meio = v.length/2;

    if (v.length % 2 == 0)
        mediana = (v[meio-1] + v[meio])/2;
    else
        mediana = v[meio];

    return mediana;
}
```

Exercícios

5

```
public static int mediana(int[] v)
    int mediana, meio;
    Arrays.sort(v);
    meio = v.length/2;

    if (v.length % 2 == 0)
        mediana = (v[meio-1] + v[meio])/2;
    else
        mediana = v[meio];

    return mediana;
}
```

algoritmo de terceiros — n log n

1

1

1

O passo dominante do algoritmo é a ordenação do vetor, que é O(n log n) (Arrays.sort() usa QuickSort ou MergeSort).
As operações restantes são O(1), não afetam a complexidade final.

n log n + 1 + 1 + 1 + 1 + 1 — n log n + 4 — **O(n log n)**

Exercícios

6

```
public static int[] inverter(int[] v) {
    int aux, ini = 0, fim = v.length - 1;
    while(ini < fim) {
        aux = v[ini];
        v[ini] = v[fim];
        v[fim] = aux;
        ini++;
        fim--;
    }
    return vet;
}
```

Exercícios

6

```
public static int[] inverter(int[] v) {
    int aux, ini = 0, fim = v.length - 1;          3
    while(ini < fim) {                             (n/2) + 1
        aux = v[ini];                              n/2
        v[ini] = v[fim];                           n/2
        v[fim] = aux;                              n/2
        ini++;                                     1
        fim--;                                     1
    }
    return vet;
}
```

A variável ini começa em 0 e é incrementada em cada iteração, enquanto fim começa em tamanho e é decrementada em cada iteração. Assim, o laço será executado tamanho/2. Ele só itera até a metade do vetor.

$3 + n/2 + 1 + 3(n/2) + 2$

$5 + 3n/2$

$O(n)$

Exercícios

7

```
public static boolean duplicacao(int[] v){
for(int i=0; i<v.length; i++){
            for(int j=i+1; j<v.length; j++){
            if(v[j] == v[i])
                        return true;
        }
    }
    return false;
}
```

Exercícios

7

loop (i) percorre os elementos do vetor de 0 até n-1. **Segundo loop (j)** percorre os elementos **após i** (j = i + 1 até n-1).
If v[j] == v[i], retorna true imediatamente.
Se percorrer todo o vetor sem encontrar duplicatas, retorna false.

```
public static boolean duplicacao(int[] v) {
    for(int i=0; i<v.length; i++) {
        for(int j=i+1; j<v.length; j++) {
            if(v[j] == v[i])
                return true;
        }
    }
    return false;
}
```

1 | n + 1 | n | n | n-1 | n-1 | n.n | 1

2 + 2n + 3n + n^2 + 1

$3 + 5n + n^2$

$O(n^2)$