ANÁLISE DE ALGORITMOS

Prof. José Carlos Althoff

2022

Retomando o estudo da complexidade

Retomando a análise do algoritmo Insertion Sort.

Video: https://www.youtube.com/watch?v=ROalU379l3U

- O tempo de execução do algoritmo é a soma dos tempos de execução para cada instrução executada;
- Uma instrução que demanda C_i passos para ser executada e é executada n vezes contribuirá com $C_i n$ para o tempo de execução total;
- Então para calcular T(n) o tempo de execução de insertion –sort de uma entrada de n valores , somamos os produtos das colunas dos custos vezes as interações, obtendo:

Insertion Sort em Java

```java
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int chave = arr[i];
            int j = i - 1;

                while (j >= 0 && arr[j] > chave) {
                    arr[j + 1] = arr[j];
                    j = j - 1;
                }
                arr[j + 1] = chave;
            }
        }
}
```

Análise do melhor caso.

1- A lista já está ordenada;
2 – O wille nunca será executado pois arr[j] $\leq chave$;
3 – O primeiro “for” percorre n-1 elementos.

Insertion Sort em Java

```java
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int chave = arr[i];
            int j = i - 1;
```

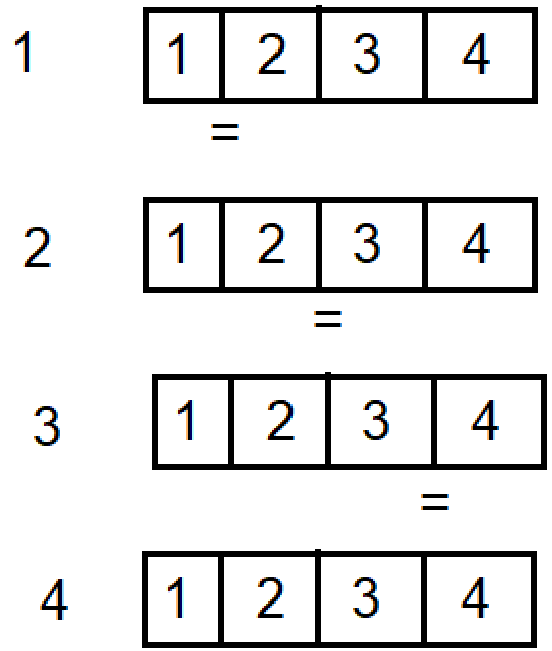
```java
                while (j >= 0 && arr[j] > chave) {
                    arr[j + 1] = arr[j];
                    j = j - 1;
                }
                arr[j + 1] = chave;
            }
        }
    }
```

Vamos agora imaginar que temos a seguinte entrada:

1	2	3	4

O vetor está todo organizado.
(melhor caso).

O Insertion Sort compara todos os elementos do vetor.



Observe que ele fez n interações

Insertion Sort em Java

```java
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int chave = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > chave) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = chave;
        }
    }
}
```

Cada linha teremos:

C (custos) vezes n(quantidade vezes que a linha é executada)

```java
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int chave = arr[i];
            int j = i - 1;
```

```java
            while (j >= 0 && arr[j] > chave) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = chave;
        }
    }
}
```

SE ESTÁ ORDENADO

n

$n-1$

NÃO PASSA

$\Rightarrow n-1$

```java
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int chave = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > chave) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = chave;
        }
    }
}
```

Se está ordenado

n

$n-1$

Não passa

$\Rightarrow n-1$

No melhor caso fica:

$$F(n) = C_1 \cdot n + C_2(n-1) + C_3(n-1) + C_4(n-1) + C_8(n-1)$$

$$F(n) = (C_1 + C_2 + C_3 + C_4 + C_8) \cdot n + (-C_2 - C_3 - C_4 - C_8)$$

$$F(m) = (C_1 + C_2 + C_3 + C_4 + C_8) \cdot m + (-C_2 - C_3 - C_4 - C_8)$$

FAZENDO: $C_1 + C_2 + C_3 + C_4 + C_8 = a$

E

$-C_2 - C_3 - C_4 - C_8 = b$

TEREMOS

$$F(m) = am + b$$ FUNÇÃO LINEAR

O Insertion Sort no Pior Caso

Vamos pensar no vetor:

4	3	2	1

1	4 =	3	2	1
2	3	4 =	2	1
3	3 =	2	4	1
4	2	3 =	4	1
5	2	3	4 =	1
6	2	3 =	1	4
7	2 =	1	3	4
8	1	2	3 =	4
9	1	2	3	4

Veja que o vetor tem apenas 4 elementos.
E tivemos 9 interações.
Podemos calcular estas interações baseado na
Entrada da seguinte maneira:

$$\frac{n(n+1)}{2} - 1 =$$

$$\frac{4(4+1)}{2} - 1 =$$

$$\frac{20}{2} - 1 =$$

$$10 - 1 = 9$$

```java
public class InsertionSort {
    public static void insertionSort(int[] arr) {
        int n = arr.length;
        for (int i = 1; i < n; i++) {
            int chave = arr[i];
            int j = i - 1;

            while (j >= 0 && arr[j] > chave) {
                arr[j + 1] = arr[j];
                j = j - 1;
            }
            arr[j + 1] = chave;
        }
    }
}
```

n

$n-1$

NÃO ESTÁ ORDENADO

$\Rightarrow \sum_{j=1}^{n} t_j$

$\rightarrow \sum_{j=1}^{n} t_j - 1$

$\Rightarrow n-1$

Assim no pior tempo do insertion-sort é:

$$T(n) = C_1 n + C_2(n-1) + C_3(n-1) + C_5\left(\frac{n(n+1)}{2} - 1\right) + C_6\left(\frac{n(n+1)}{2}\right) + C_7\left(\frac{n(n+1)}{2}\right) + C_8(n-1)$$

Devemos observar que:

$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$

Aqui temos a condição de While quantas vezes ele é executado para ordenar o arranjo.

$$\sum_{j=2}^{n} (j-1) = \frac{n(n+1)}{2}$$

$$F(n) = C_1 n + C_2 n - C_2 + C_4 n - C_4 + \frac{C_5}{2}n^2 + \frac{C_5}{2}n - C_5 + \frac{C_6}{2}n^2 + \frac{C_6}{2}n + \frac{C_7}{2}n^2 + \frac{C_7}{2}n + C_8 n - C_8$$

Os Valores C1, C2, C3, C4, C5, C6, C7, C8 são considerados constante.

Escrevemos:

$$F(n)=\left(\frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2}\right)n^2 + \left(C_1 + C_2 + C_4 + \frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2} + C_8\right)n - (C_2 + C_4 + C_5 + C_8)$$

Fazendo: $a = \left(\frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2}\right)$; $b = \left(C_1 + C_2 + C_4 + \frac{C_5}{2} + \frac{C_6}{2} + \frac{C_7}{2} + C_8\right)$ e $c = -(C_2 + C_4 + C_5 + C_8)$

Fica: $F(n)= an^2 + bn + c$

UMA FUNÇÃO
2º GRAU

Normalmente estamos interessados na procura do pior caso pois estabelece um limite superior para o tempo de execução para qualquer entrada. Conhecê-lo nos da a garantia de que um algoritmo nunca demorará mais do que esse tempo.