Aula 09 – Análise Assintótica de Algoritmos Iterativos e Recursivos

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

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

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- $n^2 + 2n - 3$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- $n^2 + 2n - 3$
- Contamos realmente todas as operações?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- $n^2 + 2n - 3$
- Contamos realmente todas as operações?
- Não. Apenas as que consideramos relevantes

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- O que acontece se incluirmos as demais operações?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- O que acontece se incluirmos as demais operações?
- Adicionamos $2n + 3$ operações

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- O que acontece se incluirmos as demais operações?
- Adicionamos $2n + 3$ operações
- Mais $2\frac{n(n-1)}{2} + n - 1$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;                                          3
  for (i=1; i<n ; i++) {                                  2n
    aux = v[i];                                           n − 1
    j = i;                                                n − 1
    while ((j > 0) &&
                  (aux < v[j-1])) {                       2 n(n−1)/2 + n − 1
      v[j] = v[j-1];                                      n(n−1)/2
      j--;                                                n(n−1)/2
    }
    v[j] = aux;                                           n − 1
  }
}
```

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Ou seja:

$$6n + 2n(n-1) - 1$$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Ou seja:

$$6n + 2n(n-1) - 1$$

$$= 6n + 2n^2 - 2n - 1$$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Ou seja:

$$6n + 2n(n-1) - 1$$

$$= 6n + 2n^2 - 2n - 1$$

$$= 2n^2 + 4n - 1$$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Comparemos as 2 versões:

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Comparemos as 2 versões:
  - $v_1 : n^2 + 2n - 3$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Comparemos as 2 versões:
  - $v_1 : n^2 + 2n - 3$
  - $v_2 : 2n^2 + 4n - 1$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Comparemos as 2 versões:
  - $v_1 : n^2 + 2n - 3$
  - $v_2 : 2n^2 + 4n - 1$
- Ou seja $v_2 \approx 2 * v_1$
  - Diferem basicamente por uma constante

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E isso importa?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E isso importa?
  - Se o objetivo for fazer uma estimativa mais precisa, com certeza!

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E isso importa?
  - Se o objetivo for fazer uma estimativa mais precisa, com certeza!
  - Mas se o objetivo for fazer uma análise assintótica do algoritmo, certamente não

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E isso porque as duas contagens são muito parecidas, a menos de uma constante multiplicativa e um número fixo de operações

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E isso porque as duas contagens são muito parecidas, a menos de uma constante multiplicativa e um número fixo de operações
- Então, qual seria a complexidade desse algoritmo?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- $\Theta(n^2 + 2n - 3)$, ou simplesmente $\Theta(n^2 + n)$ ou $\Theta(n^2)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- $\Theta(n^2 + 2n - 3)$, ou simplesmente $\Theta(n^2 + n)$ ou $\Theta(n^2)$
  - $n^2 + n \leq n^2 + 2n - 3 \leq 2n^2 + 2n$, para $n \geq 3$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- $\Theta(n^2 + 2n - 3)$, ou simplesmente $\Theta(n^2 + n)$ ou $\Theta(n^2)$
  - $n^2 + n \leq n^2 + 2n - 3 \leq 2n^2 + 2n$, para $n \geq 3$
  - $n^2 + n \leq 2n^2 + 4n - 1 \leq 4n^2 + 4n$, para $n \geq 2$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E por que Θ?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E por que Θ?
- Já temos o cálculo "exato" $\longrightarrow$ temos um limite assintótico firme

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- E por que Θ?
- Já temos o cálculo "exato" $\longrightarrow$ temos um limite assintótico firme
- E precisamos mesmo disso?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Suponha que nos interessa apenas um limite superior

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Suponha que nos interessa apenas um limite superior
- Como isso nos ajuda a calcular a complexidade desse algoritmo?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Temos um laço proporcional à entrada: $O(n)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Temos um laço proporcional à entrada: $O(n)$
  - Faz, no máximo, n iterações

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Temos um laço proporcional à entrada: $O(n)$
  - Faz, no máximo, n iterações
- E outro proporcional à entrada ($O(n)$) dentro deste

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Então o algoritmo é $O(n)O(n)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Então o algoritmo é $O(n)O(n)$
- E, lembrando que $O(f(n))O(g(n)) = O(f(n)g(n))$, temos que $O(n)O(n) = O(n^2)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Ou seja, uma simples inspeção já nos diz que o algoritmo é $O(n^2)$, no pior caso

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Ou seja, uma simples inspeção já nos diz que o algoritmo é $O(n^2)$, no pior caso
- Mas ele não era $\Theta(n^2 + n)$?

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Sim, mas lembre que $\Theta(n^2 + n) \Rightarrow O(n^2 + n)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Sim, mas lembre que $\Theta(n^2 + n) \Rightarrow O(n^2 + n)$
- E que $O(f(n) + g(n)) = O(f(n)) + O(g(n))$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Então $O(n^2 + n) = O(n^2) + O(n)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Então $O(n^2 + n) = O(n^2) + O(n)$
- Mas $O(f(n)) + O(g(n)) = O(max(f(n), g(n)))$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Então $O(n^2 + n) = O(max(n^2, n)) = O(n^2)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Então $O(n^2 + n) = O(max(n^2, n)) = O(n^2)$
- E assim $\Theta(n^2 + n) \Rightarrow O(n^2)$

Análise de Algoritmos Iterativos

- Já calculamos o número de operações executadas pelo algoritmo de ordenação por inserção

```
void insercao(int v[], int n) {
  int i, j, aux;
  for (i=1; i<n ; i++) {
    aux = v[i];
    j = i;
    while ((j > 0) &&
                  (aux < v[j-1])) {
      v[j] = v[j-1];
      j--;
    }
    v[j] = aux;
  }
}
```

- Então $O(n^2 + n) = O(max(n^2, n)) = O(n^2)$
- E assim $\Theta(n^2 + n) \Rightarrow O(n^2)$
- O limite só ficou mais "frouxo"

Análise de Algoritmos Iterativos

- A não ser que haja chamadas a métodos ou algum outro artifício que esconda operações, calcular a complexidade de algoritmos iterativos, usando a notação assintótica, não é tão difícil

Análise de Algoritmos Iterativos

- A não ser que haja chamadas a métodos ou algum outro artifício que esconda operações, calcular a complexidade de algoritmos iterativos, usando a notação assintótica, não é tão difícil
- Principalmente para o cálculo de limite superior, caso em que basta lembrar das operações com a notação O. Em especial:

Análise de Algoritmos Iterativos

- A não ser que haja chamadas a métodos ou algum outro artifício que esconda operações, calcular a complexidade de algoritmos iterativos, usando a notação assintótica, não é tão difícil
- Principalmente para o cálculo de limite superior, caso em que basta lembrar das operações com a notação O. Em especial:
  - $O(f(n) + g(n)) = O(f(n)) + O(g(n))$

Análise de Algoritmos Iterativos

- A não ser que haja chamadas a métodos ou algum outro artifício que esconda operações, calcular a complexidade de algoritmos iterativos, usando a notação assintótica, não é tão difícil
- Principalmente para o cálculo de limite superior, caso em que basta lembrar das operações com a notação O. Em especial:
  - $O(f(n) + g(n)) = O(f(n)) + O(g(n))$
  - $O(f(n)) + O(g(n)) = O(max(f(n), g(n)))$

Análise de Algoritmos Iterativos

- A não ser que haja chamadas a métodos ou algum outro artifício que esconda operações, calcular a complexidade de algoritmos iterativos, usando a notação assintótica, não é tão difícil
- Principalmente para o cálculo de limite superior, caso em que basta lembrar das operações com a notação O. Em especial:
  - $O(f(n) + g(n)) = O(f(n)) + O(g(n))$
  - $O(f(n)) + O(g(n)) = O(max(f(n), g(n)))$
  - $O(f(n))O(g(n)) = O(f(n)g(n))$

Análise de Algoritmos Recursivos

- Mas e quando o algoritmo é recursivo?

Análise de Algoritmos Recursivos

- Mas e quando o algoritmo é recursivo?
  - Teremos que observar a relação de recorrência

Análise de Algoritmos Recursivos

- Mas e quando o algoritmo é recursivo?
  - Teremos que observar a relação de recorrência

Exemplo: Busca Sequencial

- Considere a busca sequencial recursiva:

```
se n=1:
  se A[0] é o elemento buscado: achou
  senão: não está no arranjo
senão:
  se o elemento atual é o buscado: achou
  senão:
    busque nos n-1 elementos restantes
```

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

```
int buscaRec(int A[],int x,int n){
  if(n == 1) {
    if(A[0] == x) return 0;
    else return -1;
  } else {
    if(A[n-1] == x) return n-1;
    return buscaRec(A, x, n-1);
  }
}
```

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

```
int buscaRec(int A[],int x,int n){
  if(n == 1) {
    if(A[0] == x) return 0;
    else return -1;
  } else {
    if(A[n-1] == x) return n-1;
    return buscaRec(A, x, n-1);
  }
}
```

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

```
int buscaRec(int A[],int x,int n){
  if(n == 1) {
    if(A[0] == x) return 0;
    else return -1;
  } else {
    if(A[n-1] == x) return n-1;
    return buscaRec(A, x, n-1);
  }
}
```

$$T(n) = \begin{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

```
int buscaRec(int A[],int x,int n){
  if(n == 1) {
    if(A[0] == x) return 0;
    else return -1;
  } else {
    if(A[n-1] == x) return n-1;
    return buscaRec(A, x, n-1);
  }
}
```

$$T(n) = \begin{cases} 1, \text{ se } n = 1 \\ \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

```
int buscaRec(int A[],int x,int n){
  if(n == 1) {
    if(A[0] == x) return 0;
    else return -1;
  } else {
    if(A[n-1] == x) return n-1;
    return buscaRec(A, x, n-1);
  }
}
```

$$T(n) = \begin{cases} 1, \text{ se } n = 1 \\ T(n-1)+1, \text{para } n \geq 2 \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

- É equivalente a:

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ T(n-1) + O(1), & \text{para } n \geq 2 \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

- É equivalente a:

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ T(n-1) + O(1), & \text{para } n \geq 2 \end{cases}$$

Note que $O(1)$ engloba qualquer custo constante para a operação

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

- É equivalente a:

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ T(n-1) + O(1), & \text{para } n \geq 2 \end{cases}$$

- E expandindo...

$$T(n) = T(n-1) + O(1)$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

- É equivalente a:

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ T(n-1) + O(1), & \text{para } n \geq 2 \end{cases}$$

- E expandindo...

$$\begin{aligned} T(n) &= T(n-1) + O(1) \\ &= ((T(n-2) + O(1)) + O(1) \end{aligned}$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$= (((T(n-3) + O(1)) + O(1)) + O(1)$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
&= (((T(n-3)+O(1))+O(1))+O(1) \\
&= \ldots
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
&= (((T(n-3)+O(1))+O(1))+O(1) \\
&= \ldots \\
&= (\ldots((T(n-k)\underbrace{+O(1))+\ldots+O(1))+O(1)}_{\text{k vezes}}
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
&= (((T(n-3)+O(1))+O(1))+O(1) \\
&= \ldots \\
&= (\ldots((T(n-k)\underbrace{+O(1))+\ldots+O(1))+O(1)}_{\text{k vezes}} \\
&= T(1)+kO(1), \text{ quando } T(n-k)=T(1)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
&= (((T(n-3)+O(1))+O(1))+O(1) \\
&= \ldots \\
&= (\ldots((T(n-k)\underbrace{+O(1))+\ldots+O(1))+O(1)}_{\text{k vezes}} \\
&= T(1)+kO(1), \text{ quando } T(n-k)=T(1) \\
&= T(1)+(n-1)O(1) \text{ (pois } n-k=1)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
&= (((T(n-3)+O(1))+O(1))+O(1) \\
&= \ldots \\
&= (\ldots((T(n-k)\underbrace{+O(1))+\ldots+O(1))+O(1)}_{\text{k vezes}} \\
&= T(1)+kO(1), \text{ quando } T(n-k)=T(1) \\
&= T(1)+(n-1)O(1) \text{ (pois } n-k=1) \\
&= O(1)+(n-1)O(1) \text{ (pois } T(1)=O(1))
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$T(n) = O(1) + (n-1)O(1)$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$\begin{aligned} T(n) &= O(1) + (n-1)O(1) \\ &= O(1) + nO(1) - O(1) \end{aligned}$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$\begin{aligned} T(n) &= O(1) + (n-1)O(1) \\ &= O(1) + nO(1) - O(1) \\ &= O(1) + nO(1) \end{aligned}$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
T(n) &= O(1) + (n-1)O(1) \\
&= O(1) + nO(1) - O(1) \\
&= O(1) + nO(1) \\
&= O(1) + O(n)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
T(n) &= O(1) + (n-1)O(1) \\
&= O(1) + nO(1) - O(1) \\
&= O(1) + nO(1) \\
&= O(1) + O(n) \\
&= O(n)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
T(n) &= O(1) + (n-1)O(1) \\
&= O(1) + nO(1) - O(1) \\
&= O(1) + nO(1) \\
&= O(1) + O(n) \\
&= O(n)
\end{aligned}
$$

?

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
T(n) &= O(1) + (n-1)O(1) \\
&= O(1) + nO(1) - O(1) \\
&= O(1) + nO(1) \\
&= O(1) + O(n) \\
&= O(n)
\end{aligned}
$$

?

- Por que $O(1) - O(1) = O(1)$?

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

$$
\begin{aligned}
T(n) &= O(1) + (n-1)O(1) \\
&= O(1) + nO(1) - O(1) \\
&= O(1) + nO(1) \\
&= O(1) + O(n) \\
&= O(n)
\end{aligned}
$$

?

- Por que $O(1) - O(1) = O(1)$?
  - Porque $f(n) \in O(1) \Rightarrow f(n) \leq c \times 1$, e não necessariamente as constantes c são iguais

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Considere agora o problema das Torres de Hanoi:

```
se n=1:
  mova o disco do pino de origem para
                              o de destino
senão:
  Mova n-1 discos do pino de origem
                          para o auxiliar
  Mova um disco do pino de origem para
                              o de destino
  Mova n-1 discos do pino auxiliar
                         para o de destino
```

$$T(n) = \begin{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Considere agora o problema das Torres de Hanoi:

```
se n=1:
  mova o disco do pino de origem para
                              o de destino
senão:
  Mova n-1 discos do pino de origem
                          para o auxiliar
  Mova um disco do pino de origem para
                              o de destino
  Mova n-1 discos do pino auxiliar
                         para o de destino
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Considere agora o problema das Torres de Hanoi:

```
se n=1:
  mova o disco do pino de origem para
                              o de destino
senão:
  Mova n-1 discos do pino de origem
                          para o auxiliar
  Mova um disco do pino de origem para
                              o de destino
  Mova n-1 discos do pino auxiliar
                          para o de destino
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ T(n-1) \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Considere agora o problema das Torres de Hanoi:

```
se n=1:
  mova o disco do pino de origem para
                              o de destino
senão:
  Mova n-1 discos do pino de origem
                          para o auxiliar
  Mova um disco do pino de origem para
                              o de destino
  Mova n-1 discos do pino auxiliar
                          para o de destino
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ T(n-1) + O(1) \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Considere agora o problema das Torres de Hanoi:

```
se n=1:
  mova o disco do pino de origem para
                              o de destino
senão:
  Mova n-1 discos do pino de origem
                          para o auxiliar
  Mova um disco do pino de origem para
                              o de destino
  Mova n-1 discos do pino auxiliar
                          para o de destino
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 1. \\ T(n-1) + O(1) + T(n-1) & \text{para } n \geq 2 \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

```
void hanoi(char ori, char dst, char aux, int n) {
  if(n == 1) {
    printf("Mova de %c para %c.\n", ori, dst);
  } else {
    hanoi(ori, aux, dst, n-1);
    hanoi(ori, dst, aux, 1);
    hanoi(aux, dst, ori, n-1);
  }
}
```

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

```
void hanoi(char ori, char dst, char aux, int n) {
  if(n == 1) {
    printf("Mova de %c para %c.\n", ori, dst);
  } else {
    hanoi(ori, aux, dst, n-1);
    hanoi(ori, dst, aux, 1);
    hanoi(aux, dst, ori, n-1);
  }
}
```

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

```
void hanoi(char ori, char dst, char aux, int n) {
  if(n == 1) {
    printf("Mova de %c para %c.\n", ori, dst);
  } else {
    hanoi(ori, aux, dst, n-1);
    hanoi(ori, dst, aux, 1);
    hanoi(aux, dst, ori, n-1);
  }
}
```

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

```
void hanoi(char ori, char dst, char aux, int n) {
  if(n == 1) {
    printf("Mova de %c para %c.\n", ori, dst);
  } else {
    hanoi(ori, aux, dst, n-1);
    printf("Mova de %c para %c.\n", ori, dst);
    hanoi(aux, dst, ori, n-1);
  }
}
```

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

```
void hanoi(char ori, char dst, char aux, int n) {
  if(n == 1) {
    printf("Mova de %c para %c.\n", ori, dst);
  } else {
    hanoi(ori, aux, dst, n-1);
    printf("Mova de %c para %c.\n", ori, dst);
    hanoi(aux, dst, ori, n-1);
  }
}
```

$$T(n) = \begin{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

```
void hanoi(char ori, char dst, char aux, int n) {
  if(n == 1) {
    printf("Mova de %c para %c.\n", ori, dst);
  } else {
    hanoi(ori, aux, dst, n-1);
    printf("Mova de %c para %c.\n", ori, dst);
    hanoi(aux, dst, ori, n-1);
  }
}
```

$$T(n) = \begin{cases} 1 & \text{se } n = 1. \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

```
void hanoi(char ori, char dst, char aux, int n) {
  if(n == 1) {
    printf("Mova de %c para %c.\n", ori, dst);
  } else {
    hanoi(ori, aux, dst, n-1);
    printf("Mova de %c para %c.\n", ori, dst);
    hanoi(aux, dst, ori, n-1);
  }
}
```

$$T(n) = \begin{cases} 1 & \text{se } n = 1. \\ 2 * T(n-1)+1 & \text{para } n \geq 2 \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$T(n) = 2T(n-1)+1$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &= 2(2T(n-2)+1)+1 \end{aligned}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$\begin{aligned} T(n) &= 2T(n-1)+1 \\ &= 2(2T(n-2)+1)+1 \\ &= 4T(n-2)+3 \end{aligned}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2(2T(n-2)+1)+1 \\
&= 4T(n-2)+3 \\
&= 4(2T(n-3)+1)+3
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2(2T(n-2)+1)+1 \\
&= 4T(n-2)+3 \\
&= 4(2T(n-3)+1)+3 \\
&= 8T(n-3)+7
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2(2T(n-2)+1)+1 \\
&\quad 4T(n-2)+3 \\
&= 4(2T(n-3)+1)+3 \\
&\quad 8T(n-3)+7 \\
&= \ldots
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
T(n) &= 2T(n-1)+1 \\
&= 2(2T(n-2)+1)+1 \\
&\quad 4T(n-2)+3 \\
&= 4(2T(n-3)+1)+3 \\
&\quad 8T(n-3)+7 \\
&= \ldots \\
&= 2^k T(n-k)+(2^k-1)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$= \quad 2^k T(1) + (2^k - 1) (\text{quando } T(n-k) = T(1))$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
&= 2^k T(1) + (2^k - 1) (\text{quando } T(n-k) = T(1)) \\
&= 2^{n-1} T(1) + (2^{n-1} - 1) (\text{pois } n - k = 1)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
&= 2^k T(1) + (2^k - 1) (\text{quando } T(n-k) = T(1)) \\
&= 2^{n-1} T(1) + (2^{n-1} - 1) (\text{pois } n - k = 1) \\
&= 2^{n-1} + (2^{n-1} - 1) (\text{pois } T(1) = 1)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
&= 2^k T(1) + (2^k - 1) (\text{quando } T(n-k) = T(1)) \\
&= 2^{n-1} T(1) + (2^{n-1} - 1) (\text{pois } n - k = 1) \\
&= 2^{n-1} + (2^{n-1} - 1) (\text{pois } T(1) = 1) \\
&= 2 \times 2^{n-1} - 1
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
&= 2^k T(1) + (2^k - 1) (\text{quando } T(n-k) = T(1)) \\
&= 2^{n-1} T(1) + (2^{n-1} - 1) (\text{pois } n - k = 1) \\
&= 2^{n-1} + (2^{n-1} - 1) (\text{pois } T(1) = 1) \\
&= 2 \times 2^{n-1} - 1 \\
&= 2^n - 1
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
&= 2^k T(1) + (2^k - 1) (\text{quando } T(n-k) = T(1)) \\
&= 2^{n-1} T(1) + (2^{n-1} - 1) (\text{pois } n - k = 1) \\
&= 2^{n-1} + (2^{n-1} - 1) (\text{pois } T(1) = 1) \\
&= 2 \times 2^{n-1} - 1 \\
&= 2^n - 1 \\
&= 2^n - 1
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

$$
\begin{aligned}
&= 2^k T(1) + (2^k - 1) (\text{quando } T(n-k) = T(1)) \\
&= 2^{n-1} T(1) + (2^{n-1} - 1) (\text{pois } n - k = 1) \\
&= 2^{n-1} + (2^{n-1} - 1) (\text{pois } T(1) = 1) \\
&= 2 \times 2^{n-1} - 1 \\
&= 2^n - 1 \\
&= 2^n - 1 \\
&= \Theta(2^n)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in \Theta(2^n)$.

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in \Theta(2^n)$. E o que é n?

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in \Theta(2^n)$. E o que é n?
  - n é o número de discos $\rightarrow$ valor de entrada do algoritmo

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in \Theta(2^n)$. E o que é n?
  - n é o número de discos $\rightarrow$ valor de entrada do algoritmo
- Mas não havíamos visto no início que n era o tamanho do problema? Como fica então?

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in \Theta(2^n)$. E o que é n?
  - n é o número de discos $\rightarrow$ valor de entrada do algoritmo
- Mas não havíamos visto no início que n era o tamanho do problema? Como fica então?
- Algumas vezes é mais útil relaxar essa definição e redefinir n

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in \Theta(2^n)$. E o que é n?
  - n é o número de discos $\rightarrow$ valor de entrada do algoritmo
- Mas não havíamos visto no início que n era o tamanho do problema? Como fica então?
- Algumas vezes é mais útil relaxar essa definição e redefinir n
  - Nesse caso, dizemos que $T(n) \in \Theta(2^n)$, onde n é o número de discos na torre

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in \Theta(2^n)$. E o que é n?
  - n é o número de discos $\rightarrow$ valor de entrada do algoritmo
- Mas não havíamos visto no início que n era o tamanho do problema? Como fica então?
- Algumas vezes é mais útil relaxar essa definição e redefinir n
  - Nesse caso, dizemos que $T(n) \in \Theta(2^n)$, onde n é o número de discos na torre
  - Ou, alternativamente, que $T(n)$ é $\Theta(2^n)$ no valor da entrada

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Considere agora o fatorial recursivo:

```
Entrada: inteiro n

Se n=0, retorne 1
Senão
  retorne n multiplicado pelo
  fatorial de n-1
```

$$T(n) = \begin{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Considere agora o fatorial recursivo:

```
Entrada: inteiro n

Se n=0, retorne 1
Senão
  retorne n multiplicado pelo
  fatorial de n-1
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 0. \\ \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Considere agora o fatorial recursivo:

```
Entrada: inteiro n

Se n=0, retorne 1
Senão
  retorne n multiplicado pelo
  fatorial de n-1
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 0. \\ O(1) \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Considere agora o fatorial recursivo:

```
Entrada: inteiro n

Se n=0, retorne 1
Senão
  retorne n multiplicado pelo
  fatorial de n-1
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 0. \\ O(1) + T(n-1) & \text{para } n \geq 1 \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Considere agora o fatorial recursivo:

```
Entrada: inteiro n

Se n=0, retorne 1
Senão
  retorne n multiplicado pelo
  fatorial de n-1
```

$$T(n) = \begin{cases} O(1), & \text{se } n = 0. \\ O(1) + T(n-1) & \text{para } n \geq 1 \end{cases}$$

- Já vimos que $T(n) \in O(n)$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

```
long fatorial(long n) {
  if (n==0) return 1;
  return n * fatorial(n-1);
}
```

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

```
long fatorial(long n) {
  if (n==0) return 1;
  return n * fatorial(n-1);
}
```

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

```
long fatorial(long n) {
  if (n==0) return 1;
  return n * fatorial(n-1);
}
```

$$T(n) = \begin{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

```
long fatorial(long n) {
  if (n==0) return 1;
  return n * fatorial(n-1);
}
```

$$T(n) = \begin{cases} 0, & \text{se } n = 0. \\ \end{cases}$$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

```
long fatorial(long n) {
  if (n==0) return 1;
  return n * fatorial(n-1);
}
```

$$T(n) = \begin{cases} 0, & \text{se } n = 0. \\ T(n-1)+1 & \text{para } n \geq 1 \end{cases}$$

Análise de Algoritmos Recursivos

Notas finais

- Note que, mesmo a notação assintótica nos ajudando com as relações de recorrência, ainda assim temos que resolvê-las

Análise de Algoritmos Recursivos

Notas finais

- Note que, mesmo a notação assintótica nos ajudando com as relações de recorrência, ainda assim temos que resolvê-las
  - Não há como ter uma ideia da complexidade por uma simples inspeção, como era o caso com algoritmos iterativos

Análise de Algoritmos Recursivos

Notas finais

- Note que, mesmo a notação assintótica nos ajudando com as relações de recorrência, ainda assim temos que resolvê-las
  - Não há como ter uma ideia da complexidade por uma simples inspeção, como era o caso com algoritmos iterativos
- E não há realmente como fugir disso...

Análise de Algoritmos Recursivos

Notas finais

- Note que, mesmo a notação assintótica nos ajudando com as relações de recorrência, ainda assim temos que resolvê-las
  - Não há como ter uma ideia da complexidade por uma simples inspeção, como era o caso com algoritmos iterativos
- E não há realmente como fugir disso...
- Mas em alguns casos, podemos obter uma boa ajuda

Análise de Algoritmos Recursivos

Notas finais

- Note que, mesmo a notação assintótica nos ajudando com as relações de recorrência, ainda assim temos que resolvê-las
  - Não há como ter uma ideia da complexidade por uma simples inspeção, como era o caso com algoritmos iterativos
- E não há realmente como fugir disso...
- Mas em alguns casos, podemos obter uma boa ajuda
  - Veremos na próxima aula

Referências

- Ziviani, Nivio. Projeto de Algoritmos: com implementações em Java e C++. Cengage. 2007.

- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.

Aula 09 – Análise Assintótica de Algoritmos Iterativos e Recursivos

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023