Aula 09 – Análise Assintótica de Algoritmos Iterativos e Recursivos

Norton Trevisan Roman
norton@usp.br

18 de setembro de 2018

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 $3(n-1)$ operações

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 $3(n-1)$ operações
- Mais $2\frac{n(n-1)}{2}$

Análise de Algoritmos Iterativos

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

`static void insercao(int[] v) {`	
`  for (int i=1; i<v.length; i++) {`	$3(n-1)$
`    int aux = v[i];`	$n-1$
`    int j = i;`	$n-1$
`    while ((j > 0) &&`	
`                 (aux < v[j-1])) {`	$2\frac{n(n-1)}{2}$
`      v[j] = v[j-1];`	$\frac{n(n-1)}{2}$
`      j--;`	$\frac{n(n-1)}{2}$
`    }`	
`    v[j] = aux;`	$n-1$
`  }`	
`}`	

Análise de Algoritmos Iterativos

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

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

- Ou seja:

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

Análise de Algoritmos Iterativos

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

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

- Ou seja:

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

Análise de Algoritmos Iterativos

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

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

- Ou seja:

$$6(n-1) + 2n(n-1)$$
$$= 6n - 6 + 2n^2 - 2n$$
$$= 2n^2 + 4n - 6$$

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

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

- Comparemos as 2 versões:
  - $v_1 : n^2 + 2n - 3$
  - $v_2 : 2n^2 + 4n - 6$

Análise de Algoritmos Iterativos

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

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

- Comparemos as 2 versões:
  - $v_1 : n^2 + 2n - 3$
  - $v_2 : 2n^2 + 4n - 6$
- Ou seja $v_2 = 2v_1$
  - Diferem apenas por uma constante

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 idênticas, a menos de uma constante

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 idênticas, a menos de uma constante
- Então, qual seria a complexidade desse algoritmo?

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 $\Theta(n^2 + n)$

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 $\Theta(n^2 + n)$
  - $n^2 + n \leq n^2 + 2n - 3 \leq 2n^2 + 2n$, para $n \geq 3$

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 $\Theta(n^2 + n)$
  - $n^2 + n \leq n^2 + 2n - 3 \leq 2n^2 + 2n$, para $n \geq 3$
  - $n^2 + n \leq 2n^2 + 4n - 6 \leq 4n^2 + 4n$, para $n \geq 2$

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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" $\rightarrow$ temos um limite assintótico firme

Análise de Algoritmos Iterativos

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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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" $\rightarrow$ 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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

```
static void insercao(int[] v) {
  for (int i=1; i<v.length; i++) {
    int aux = v[i];
    int 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 A[0] é o elemento buscado: achou
  senão:
    busque nos n-1 elementos
```

Exemplo: Busca Sequencial

- Nesse caso:

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

Análise de Algoritmos Recursivos

Exemplo: Busca Sequencial

- Nesse caso:

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

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

Exemplo: Busca Sequencial

- Nesse caso:

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

- E expandindo...

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

Exemplo: Busca Sequencial

- Nesse caso:

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

- 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

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

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

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

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

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

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

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

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

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

?

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
```

$$T(n) = \left\{ \right.$$

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

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in O(2^n)$.

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

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

Exemplo: Torres de Hanoi

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

Exemplo: Torres de Hanoi

- Então $T(n) \in O(2^n)$. 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 $T(n) \in O(2^n)$. 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 $T(n) \in O(2^n)$. 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 $T(n) \in O(2^n)$, onde n é o número de discos na torre

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- Então $T(n) \in O(2^n)$. 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 $T(n) \in O(2^n)$, onde n é o número de discos na torre
  - Ou, alternativamente, que $T(n)$ é $O(2^n)$ no valor da entrada

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- E se quiséssemos a complexidade em termos do tamanho da entrada?

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- E se quiséssemos a complexidade em termos do tamanho da entrada?
- Note que n é um número implementado com m bits

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- E se quiséssemos a complexidade em termos do tamanho da entrada?
- Note que n é um número implementado com m bits
  - Ou seja, o tamanho da entrada é m bits

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- E se quiséssemos a complexidade em termos do tamanho da entrada?
- Note que n é um número implementado com m bits
  - Ou seja, o tamanho da entrada é m bits
- Podemos escrever n então como $n = 2^{m-1}d_{m-1} + 2^{m-2}d_{m-2} + \ldots + 2^0 d_0$, onde $d_i \in \{0, 1\}$, $0 \leq i < m$ é um dígito binário

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- No pior caso, n usa todos os bits disponíveis ($d_i = 1$, $0 \leq i < m$), então $n = 2^{m-1} + 2^{m-2} + \ldots + 2^0$

Exemplo: Torres de Hanoi

- No pior caso, n usa todos os bits disponíveis ($d_i = 1$, $0 \leq i < m$), então $n = 2^{m-1} + 2^{m-2} + \ldots + 2^0$
- Ou seja

$$n = O(2^{m-1} + 2^{m-2} + \ldots + 2^0)$$

Exemplo: Torres de Hanoi

- No pior caso, n usa todos os bits disponíveis ($d_i = 1$, $0 \leq i < m$), então $n = 2^{m-1} + 2^{m-2} + \ldots + 2^0$
- Ou seja

$$\begin{aligned} n &= O(2^{m-1} + 2^{m-2} + \ldots + 2^0) \\ &= O(2^{m-1}) + O(2^{m-2}) + \ldots + O(2^0) \end{aligned}$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- No pior caso, n usa todos os bits disponíveis ($d_i = 1$, $0 \leq i < m$), então $n = 2^{m-1} + 2^{m-2} + \ldots + 2^0$
- Ou seja

$$
\begin{aligned}
n &= O(2^{m-1} + 2^{m-2} + \ldots + 2^0) \\
&= O(2^{m-1}) + O(2^{m-2}) + \ldots + O(2^0) \\
&= O(2^m)
\end{aligned}
$$

Análise de Algoritmos Recursivos

Exemplo: Torres de Hanoi

- No pior caso, n usa todos os bits disponíveis ($d_i = 1$, $0 \leq i < m$), então $n = 2^{m-1} + 2^{m-2} + \ldots + 2^0$
- Ou seja

$$\begin{aligned} n &= O(2^{m-1} + 2^{m-2} + \ldots + 2^0) \\ &= O(2^{m-1}) + O(2^{m-2}) + \ldots + O(2^0) \\ &= O(2^m) \end{aligned}$$

- E $T(n) = O(2^n) = O(2^{2^m})$, onde m é o tamanho da entrada em bits

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
```

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

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
```

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

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

- Mais uma vez, n não é o tamanho da entrada, mas a própria entrada

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Mais uma vez, n não é o tamanho da entrada, mas a própria entrada
  - Ou seja, o número do qual calculamos o fatorial

Exemplo: Fatorial recursivo

- Mais uma vez, n não é o tamanho da entrada, mas a própria entrada
  - Ou seja, o número do qual calculamos o fatorial
- E mais uma vez, podemos relaxar nossa definição, dizendo que o fatorial recursivo é $O(n)$ no valor da entrada

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Mais uma vez, n não é o tamanho da entrada, mas a própria entrada
  - Ou seja, o número do qual calculamos o fatorial
- E mais uma vez, podemos relaxar nossa definição, dizendo que o fatorial recursivo é $O(n)$ no <u>valor</u> da entrada
- E se quiséssemos sua complexidade em relação ao <u>tamanho</u> da entrada?

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Novamente lembramos que n é um valor escrito em binário

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Novamente lembramos que n é um valor escrito em binário
- E que, no pior caso, $n \in O(2^m)$, onde m é o número de bits usados para representar n

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- Novamente lembramos que n é um valor escrito em binário
- E que, no pior caso, $n \in O(2^m)$, onde m é o número de bits usados para representar n
- Então

$$\begin{aligned} T(n) &= O(n) \\ &= O(O(2^m)) \\ &= O(2^m) \end{aligned}$$

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- E assim, podemos dizer tanto que o fatorial recursivo é $O(n)$ no valor da entrada

Exemplo: Fatorial recursivo

- E assim, podemos dizer tanto que o fatorial recursivo é $O(n)$ no valor da entrada
- Quanto que ele é $O(2^n)$ no tamanho da entrada

Exemplo: Fatorial recursivo

- E assim, podemos dizer tanto que o fatorial recursivo é $O(n)$ no valor da entrada
- Quanto que ele é $O(2^n)$ no tamanho da entrada
- A utilidade prática dessa informação é que vai determinar qual das formas usar

Exemplo: Fatorial recursivo

- E assim, podemos dizer tanto que o fatorial recursivo é $O(n)$ no valor da entrada
- Quanto que ele é $O(2^n)$ no tamanho da entrada
- A utilidade prática dessa informação é que vai determinar qual das formas usar
  - Se queremos apenas as operações, independentemente do hardware, usamos o valor

Análise de Algoritmos Recursivos

Exemplo: Fatorial recursivo

- E assim, podemos dizer tanto que o fatorial recursivo é $O(n)$ no valor da entrada
- Quanto que ele é $O(2^n)$ no tamanho da entrada
- A utilidade prática dessa informação é que vai determinar qual das formas usar
  - Se queremos apenas as operações, independentemente do hardware, usamos o valor
  - Se queremos saber como cresce a necessidade de hardware (como o tamanho da palavra, por exemplo), então usamos o tamanho da entrada

Notas finais

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

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...

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.