SIN110 Algoritmos e Grafos

aula 03

Análise de Algoritmos

- algoritmos iterativos
- recursão e recorrências
- algoritmos recursivos

SIN110 Algoritmos e Grafos

Algoritmos Iterativos

Exemplos de aplicação

Classificação por Inserção:

		pior caso	
Inserção (A, n)		**contagem**	**consumo**
1	**para** j ← 2 **até** n **faça**	n	$O(n)$
2	x ← A[j]	(n-1)	$O(n)$
3	i ← j - 1	(n-1)	$O(n)$
4	**enquanto** i>0 **e** A[i]>x **faça**	n+(n-1)+...+2	$nO(n) = O(n^2)$
5	A[i+1] ← A[i]	(n-1)+(n-2)+...+1	$nO(n) = O(n^2)$
6	i ← i - 1	(n-1)+(n-2)+...+1	$nO(n) = O(n^2)$
7	A[i+1] ← x	(n-1)	$O(n)$
		$(3n^2+7n-8)/2$	***$O(3n^2+4n) = O(n^2)$***

Exemplos de aplicação

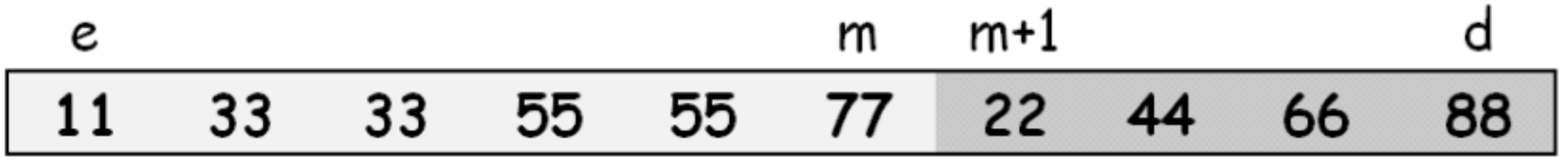
Outra análise:

		melhor caso	
Inserção (A, n)		**contagem**	**consumo**
1	**para** j ← 2 **até** n **faça**	n	$\Omega(n)$
2	x ← A[j]	(n-1)	$\Omega(n)$
3	i ← j - 1	(n-1)	$\Omega(n)$
4	**enquanto** i>0 e A[i]>x **faça**	(n-1)	$\Omega(n)$
5	A[i+1] ← A[i]	0	0
6	i ← i - 1	0	0
7	A[i+1] ← x	(n-1)	$\Omega(n)$
		5n-4	$\Omega(5n) = \Omega(n)$

Exemplo de Análise

Intercalação de dois vetores:

Supondo A[e..m] e A[m+1..d] em ordem crescente; queremos colocar A[e..d] em ordem crescente:



Intercala (A, e, m, d)	contagem	consumo
0 **crie** vetor B[e..d]	n=d-e+1	Θ(n)
1 **para** i ← e **até** m **faça**		
2 B[i] ← A[i]	2n+2	Θ(n)
3 **para** j ← m+1 **até** d **faça**		
4 B[d+m+1-j] ← A[j]		
5 i ← e	2	Θ(1)
6 j ← d		
7 **para** k ← e **até** d **faça**	n	nΘ(1) = Θ(n)
8 **se** B[i] ≤ B[i]	n	nΘ(1) = Θ(n)
9 **então** A[k] ← B[i]		
10 i ← i+1	2n	nΘ(1) = Θ(n)
11 **senão** A[k] ← B[j]		
12 j ← j-1		
Total:	**7n+4**	***Θ(5n+1) = Θ(n)***

+ Exemplo

→ *Seja M uma matriz n × m de números reais tal que:*

(a) cada linha de M está ordenada em ordem crescente (da esquerda para a direita) e,
(b) cada coluna de M está ordenada em ordem crescente (de cima para baixo).

Projete um algoritmo (baseado em comparações) que recebe M e um inteiro x e determina se x aparece em M (ou seja, se existem índices i, j tais que M[i, j] = x).

- *Analise a correção e complexidade de sua solução.*

Resolvendo ...

→ busca sequencial pesquisando as linhas e as colunas ...

```
Busca_seq(M, n, m,x)
 1. para i ← 1 até n faça
 2.         para j ← 1 até m faça
 3.                 se M[i,j] = x
 4.                     então devolve (i,j)
 5. devolve (-1,-1)
```

Análise: correção

```
Busca_seq(M, n, m,x)
 1. para i ← 1 até n faça
 2.         para j ← 1 até m faça
 3.                 se M[i,j] = x
 4.                         então devolve (i,j)
 5. devolve (-1,-1)
```

O algoritmo pára: ação dos contadores i, j que limita o número de execuções.

O algoritmo verifica corretamente a presença de x em M[1..n, 1..m] ao percorrer todos os elementos comparando-os e devolvendo o par (-1,-1) se não encontra

Análise: complexidade

Busca_seq(M, n, m,x)	**nº operações**
1. para i ← 1 até n faça	n+1
2. para j ← 1 até m faça	(m+1)+(m+1)+… = n(m+1)
3. se M[i,j] = x	m + m + … = n*m
4. então devolve (i,j)	0
5. devolve (-1,-1)	1

Total = $n[1 + (m+1) + m] + 1 = 2n(m+1)+1$

se n = m … temos $T(n) = 2n^2 + 2n + 1 = O(n^2)$

SIN110 Algoritmos e Grafos

Recursão

Recursão

- Estratégia de programação
- Expressões matemáticas
- Estruturas de dados

Ideia:

- Se *o problema é pequeno, resolva-o diretamente, como puder.*
- *Se o problema é grande, reduza-o a um problema menor do mesmo tipo*.

Recursão

Na definição de funções matemáticas, expressões algébricas, tipos de dados, etc. e geralmente são constituídas de duas partes:

- *parte base* e,
- *parte recursiva.*

Recursão - exemplo

i) Na função fatorial temos:

$Fat(0) = 1$ na parte base, e

$Fat(n) = n * Fat(n-1)$ na parte recursiva

Que resultaria:

```
Fat(n)
1    k ← 1
2    enquanto n > 0 faça
3              k ← k * n
4              n ← n-1
5    devolve k
```

Recursão - exemplo

ii) Definição dos números de Fibonacci:

$Fib(0) = 1$ e $Fib(1) = 1$ na parte base, e

$Fib(n) = Fib(n-1) + Fib(n-2), \quad n>1$ na parte recursiva

Com o seguinte algoritmo:

```
Fib(n)
1    i ← 0
2    k ← j ← 1
3    enquanto n > 0 faça
4                 k ← i + j
5                 j ← k
6                 i ← j
7                 n ← n-1
8    devolve k
```

Recursão - exemplo

iii) *Soma recursiva*

Problema:

Escreva um algoritmo recursivo que calcule a soma dos elementos do vetor A[1..n].

1. Qual a *entrada* e qual a *saída* de nosso algoritmo? Nosso algoritmo *recebe* um número *n* e um vetor *A* e *devolve* um único número, que deve ser igual à soma dos elementos de A[1..n].

2. O problema faz sentido para qualquer n $\geq$ 0 e portanto nosso algoritmo aceita n = 0, Quando n = 0, a resposta correta é 0.

3. Feita essa discussão, e fácil escrever o algoritmo:

```
Soma(n, A)
1  se n = 0
2     então S ← 0
3     senão S ← Soma(n-1, A) + A[n]
4  devolve S
```

Recursão - exemplo

```
Soma(n, A)
1  se n = 0
2     então S ← 0
3     senão S ← Soma(n-1, A) + A[n]
4  devolve S
```

O algoritmo faz a soma "da esquerda para a direita". Como faríamos para somar "da direita para esquerda"? Para fazer isso é preciso *generalizar o problema*.

O problema generalizado tem dados *n, A e k ≥ 1* e pede a soma *A[k] + ... + A[n]*.

Eis o algoritmo:

```
Soma-dir_esq(k,n,A)
1  se k > n
2     então S ← 0
3     senão S ← A[k] + Soma-dir_esq(k+1,n,A)
4  devolve S
```

Recursão - exemplo

iv) Busca em vetor ordenado

Problema: *verificar se x é elemento de um vetor ordenado crescente A[e..d]*

A solução desse problema precisa encontrar *j* tal que $A[j] \leq x < A[j+1]$, sinalizando com $j = -1$ se não for encontrado.

Vejamos a solução com uma versão de "*busca linear*" e outra de "*busca binária*":

```
Busca-linear(x,A,e,d)
1  se e = d+1
2     então devolve -1
3     senão se x = A[d]
4                então  devolve d
5             se x < A[d]
6                então devolve Busca-linear(x,A,e,d-1)
7                senão devolve -1
```

Recursão - exemplo

```
Busca-binária(x,A,e,d)
1  se e = d+1
2     então devolve -1
3     senão  m ← (e+d)/2
4               se x = A[m]
5                  então devolve m
6               se x < A[m]
7                  então devolve Busca-binária(x,A,e,m-1)
8                   senão devolve Busca-binária(x,A,m+1,d)
```

Para analisar a complexidade e correção de algoritmos que contém uma chamada recursiva, como nos exemplos, precisamos descrever o tempo de execução através de uma *recorrência*.

Exercícios

1. Escreva uma versão recursiva do algoritmo classificação por inserção.

2. Escreva uma versão recursiva do algoritmo classificação por seleção.

SIN110 Algoritmos e Grafos

Recorrências

Recursão

Se o problema é pequeno, resolva-o diretamente, como puder. Se o problema é grande, reduza-o a um problema menor do mesmo tipo.

exemplo:

```
Bbin(x,A,e,d)
1 se e = d+1
2    então devolve -1
3    senão m ← ⌊(e+d)/2⌋
4          se x = A[m]
4           então devolve m
5         se x < A[m]
6            então devolve Bbin(x,A,e,m-1)
7            senão devolve Bbin(x,A,m+1,d)
```

Para analisar a complexidade e correção de algoritmos que contém uma chamada recursiva, precisamos descrever o tempo de execução através de uma *recorrência*.

Recorrência

Equação ou desigualdade que descreve uma função em termos de uma variação dela mesma.

Exemplo:

$$f(n) = \begin{cases} c_1 & \text{se } n = 1 \\ f(n/2) + c_2 & \text{se } n > 1 \end{cases}$$

Na equação, o termo para $n = 1$ é chamado de condição inicial e o termo para $n > 1$ é denominado termo geral.

Em algumas situações, podem aparecer mais que uma condição inicial e vários termos gerais em uma mesma recorrência.

Recorrência - analise

Precisamos *resolver* a recorrência que define uma função, digamos *f(n) que descreve a complexidade.*

*Ob*ter uma "fórmula fechada" para *f(n).*

"fórmula fechada":

- É uma expressão que envolve um número fixo de operações aritméticas
- Não deve conter expressões da forma "1+2+3+...+n".

Recorrência

Resolver recorrências é uma arte!

Empregamos:

- Método da Substituição,
- Árvore de Recursão ou.
- Teorema Mestre.

Método da Substituição

1. devemos pressupor a forma da solução;
2. usar indução matemática para mostrar que a solução funciona.

Método da Substituição – exemplo 1

Seja a recorrência

$$T(1) = 1$$
$$T(n) = T(n-1)+3n+2 \quad \text{para } n = 2,3,4,\ldots$$

Que define a função T sobre inteiros positivos:

n	1	2	3	4	5	6
T(n)	1	9	20	34	51	71

Nossa hipótese será a função: $T(n) = (3/2)n^2+(7/2)n-4$

Verificando temos:

Se n=1 então $T(n) = 1 = 3/2+7/2-4$

Para $n \geq 2$ suponha que a fórmula está certa para n-1:

Então: $T(n) = T(n-1) + 3n + 2$

$=^{hip} (3/2)(n-1)^2+(7/2)(n-1)-4+3n+2$

$= (3/2)n^2-3n+3/2+(7/2)n-7/2-4+3n+2$

$=(3/2)n^2 + (7/2)n - 4$ *e está correta!*

Método da Substituição – exemplo 1 (cont.)

Ficou parecido com um "truque" !

"Hipótese" assim só tem essa eficiência se o "resolvedor" for muito experimentado!

Podemos pensar em tentar uma solução a partir dos dados obtidos com a equação, como vemos acima, supondo uma função como $An^2 + Bn + C$ para essa recorrência, e em seguida determinar os valores aproximados para os coeficientes A, B e C.

Método da Substituição – exemplo 2

Considere outra recorrência

$$G(1) = 1$$

$$G(n) = 2G(n/2)+7n+2 \quad \text{para } n = 2,4,\ldots,2^i, \ldots$$

Nossa hipótese agora por ser uma função da forma nlgn ou n^2 , vejamos alguns valores de G(n):

n	1	2	4	8	16	32	64	128	256
G(n)	1	18	66	190	494	1214	2878	6654	15102
nlgn	0	2	8	24	64	160	384	896	2048
n^2	1	4	16	64	256	1024	4096	16384	65536

Método da Substituição – exemplo 2 (cont.)

Faltando definir uma constante multiplicativa,

Observamos que nossa função para a fórmula fechada tem um termo dominante como ***nlgn*** ao invés de n^2 .

Pesquisando empiricamente chegaremos à fórmula fechada:

$G(n) = 7n\lg n + 3n - 2$ *para* $n = 1, 2, 4, 8, 16, \ldots$

Prova:

Se n=1 então $G(n) = 7(1\lg 1) + 3(1) - 2 = 1$

Para $n \geq 2$ temos

Então: $G(n) = 2G(n/2) + 7n + 2$

$=^{hip} 2(7(n/2)\lg(n/2)+3(n/2)-2) + 7n + 2$

$= 7n(\lg n-1) + 3n - 4 + 7n + 2$

$= 7n\lg n - 7n + 3n - 2 + 7n$

$= 7n\lg n + 3n - 2$ *e está correta!*

Método da Substituição

Embora o método possa fornecer uma prova de que uma solução para uma recorrência é correta, torna-se difícil apresentar uma boa suposição.

Método da Árvore de recursão

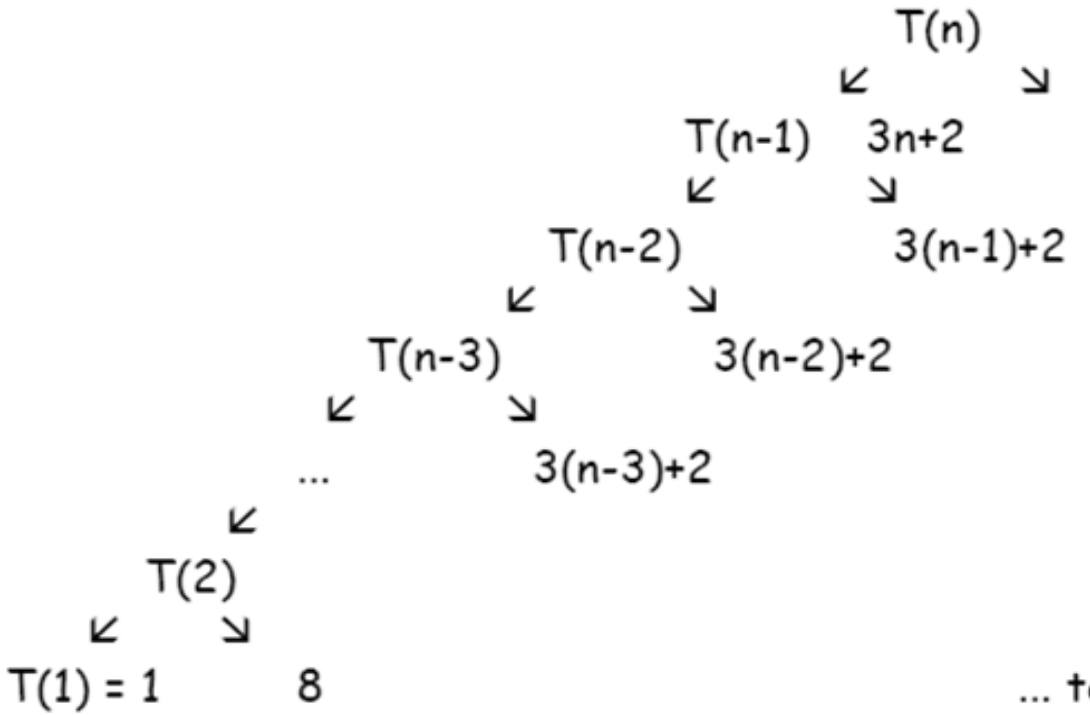
Cada nó representa o custo de um único subproblema em algum lugar do conjunto de chamadas recursivas.

No exemplo-1 com a recorrência:

$T(1) = 1$

$T(n) = T(n-1)+3n+2$ para $n = 2,3,4,\ldots$

Expandindo T(n), temos a árvore:



... temos 1 + (n-1) níveis,

Método da Árvore de recursão

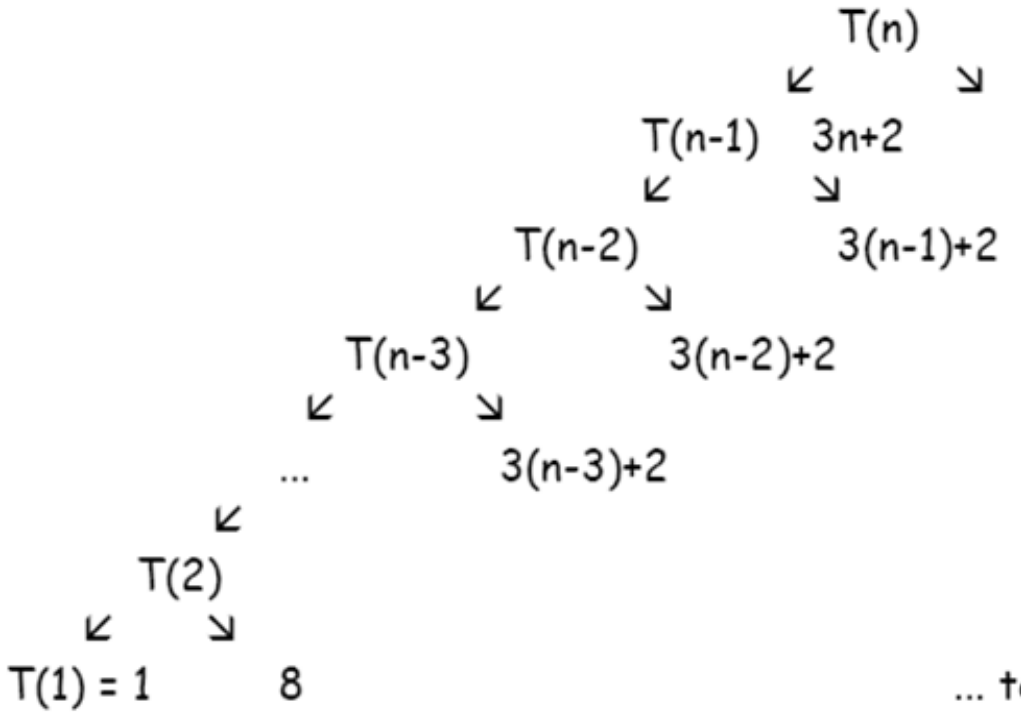
Cada nó representa o custo de um único subproblema em algum lugar do conjunto de chamadas recursivas.

No exemplo-1 com a recorrência:

$T(1) = 1$

$T(n) = T(n-1)+3n+2$ para $n = 2,3,4,\ldots$

Expandindo T(n), temos a árvore:



... temos 1 + (n-1) níveis,

Método da Árvore de recursão

exemplo-1 (cont.)

Portanto:

$$T(n) = [3n+2] + [3(n-1) + 2] +$$

$$+ [3(n-2) + 2] + [3(n-3)+2] +$$

$$+ \ldots + 8 + T(1) =$$

$$= 3[n + (n-1) + (n-2) + \ldots + 2] + 2(n-1) + 1 =$$

$$= (3/2)n^2 + (7/2)n - 4$$

... e obtemos a fórmula fechada!

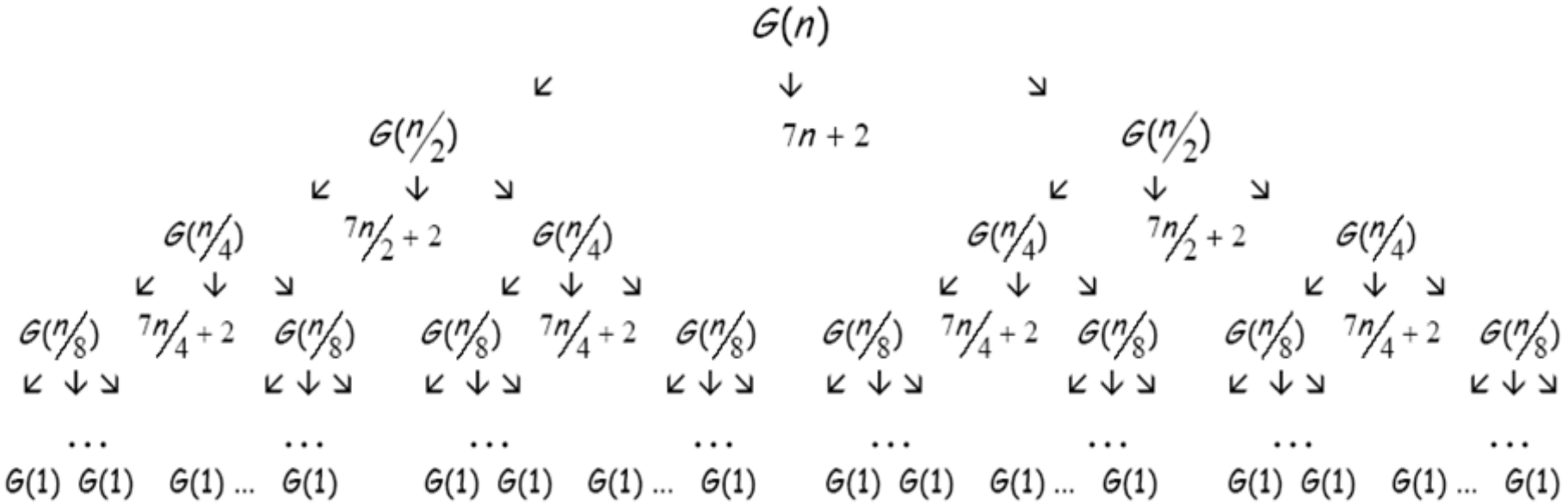
Método da Árvore de recursão

No exemplo - 2 a recorrência:

$G(1) = 1$

$G(n) = 2G(n/2)+7n+2$ para $n = 2,4,\ldots,2^i, \ldots$

Apresenta a expansão de G(n):



Método da Árvore de recursão

exemplo-2 (cont.)

Temos 1+lgn níveis, com as somas:

nível	soma no nível
0	$7n+2$
1	$7n+4$
2	$7n+8$
...	
k-1	$7n+2^k$
k	$2^kG(1)$

Método da Árvore de recursão

exemplo-2 (cont.)

Sendo $n = 2^k$ temos,

$$G(n) = 7n+2^1 + 7n+2^2 + \ldots + 7n + 2^{\lg n} + 2^{\lg n}\, G(1)$$

$$= 7n\lg n + (2^1 + 2^2 + \ldots + 2^{\lg n}) + 2^{\lg n}$$

$$= 7n\lg n + 2 \cdot 2^{\lg n} - n$$

$$= 7n\lg n + 3n - 2$$

Obs: para resolver usamos o cálculo $x^0+\ldots+x^k = (x^{k+1}-1)/(x-1)$

Exercícios

1. Determine as funções que complexidade dos exemplos de algoritmos recursivos apresentados na aula sobre "Recorrências".

2. Resolva a recorrência

$$T(1) = 1$$
$$T(2) = 1$$
$$T(n) = T(n - 2) + 2n + 1 \text{ para } n = 3, 4, 5, \ldots$$

3. Resolva a recorrência

$$T(1) = 1$$
$$T(n) = T(\lfloor n/2 \rfloor) + 1 \text{ para } n = 2, 3, 4, 5, \ldots$$

SIN110 Algoritmos e Grafos

Algoritmos Recursivos

Algoritmos Recursivos

Análise da classificação por inserção recursiva, onde $n = d - e + 1$:

```
Inserção_Rec (A, e, d)                    contagem
1  se e < d                                   1
2     então Inserção_Rec(A,e,d-1)          f(n-1)
3  x ← A[d]                                   1
4  i ← d-1                                    1
5  enquanto i>0 e A[i]>x faça                 n
6           A[i+1] ← A[i]                    n-1
7           i ← i - 1                        n-1
8  A[i+1] ← x                                 1
```

Trabalhando novamente com a hipótese do pior caso,
o algoritmo consome um tempo $f(n)$, que terá a seguinte expressão:

$f(1) = 1$ se $n = 1$,

$f(n) = f(n-1) + 3n + 2$ se temos $n = 2, 3, 4$, etc.

Algoritmos Recursivos

Tentando uma fórmula "fechada":

$f(n) = f(n-1) + [3n + 2]$

$f(n) = f(n-2) + [3(n-1) + 2] + [3n + 2]$

$f(n) = f(n-3) + [3(n-2) + 2] + [3(n-1) + 2] + [3n + 2]$

$f(n) = f(n-4) + [3(n-3) + 2] + [3(n-2) + 2] + [3(n-1) + 2] + [3n + 2]$

. . .

$f(n) = f(2) + [3(3) + 2] + [3(4) + 2] + \ldots + [3(n-1) + 2] + [3n + 2]$

$f(n) = f(1) + [3(2) + 2] + [3(3) + 2] + \ldots + [3(n-1) + 2] + [3n + 2]$

assim obtemos:

$f(n) = 1 + 2(n-1) + 3[2 + 3 + \ldots + (n-1) + n]$, que resulta

$f(n) = 1 + 2n - 2 + (3/2)[(n+2)(n-1)]$

e finalmente:

$f(n) = (3/2)n^2 + (7/2)n - 4$ com consumo de tempo na ordem $O(n^2)$.

Exercício

1. Quantas vezes a comparação "$A[d] \neq 0$" é executada?

   Defina esse número por meio de uma recorrência.

```
Limpa(A,e,d)
1  se e = d
2    então devolve d
3    senão m ← Limpa(A,e,d-1)
4          se A[d] ≠ 0
5            então m ← m+1
6                  A[m] ← A[d]
7  devolve m
```

Dê uma fórmula exata par a função definida pela recorrência.

Em que *classe* Θ está a função definida pela recorrência?