Recursividade

Projeto e Análise de Algoritmos

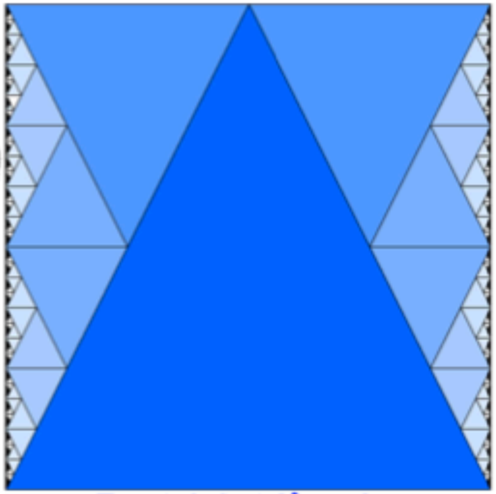
Felipe Cunha

Pontifícia Universidade Católica de Minas Gerais

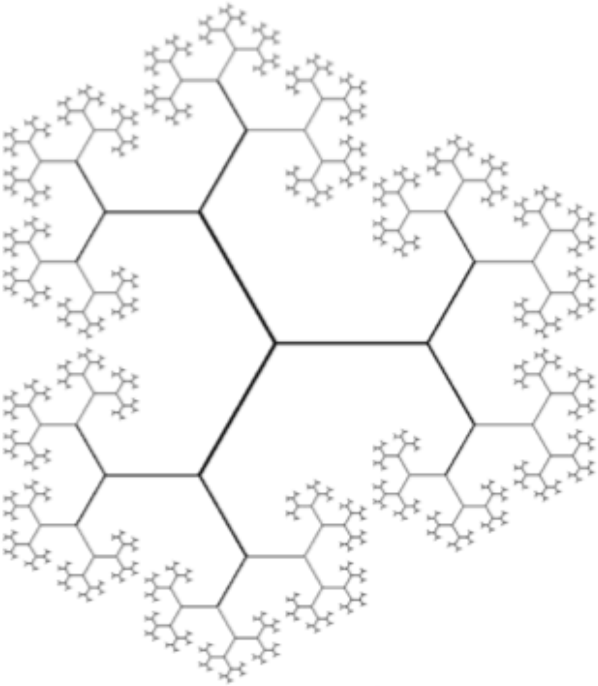
Recursividade

Fractal de Fern

Fractal espiral



Fractal de triângulos



Derivação binária

Recursividade

Foto recursiva

Imagem recursiva

Pensamento recursivo

Recursividade

- Um algoritmo recursivo é aquele que direta ou indiretamente chama a si próprio.
- Procedimentos recursivos permitem definir um número infinito de instruções através de uma rotina finita.
- Durante a execução de um procedimento recursivo, diversas ativações são empilhadas na pilha do sistema, ocupando memória e gastando tempo.
  - Isto é uma limitação para o uso da recursividade em programas.

Poder da Recursão

- Definir um conjunto infinito de objetos através de um comando finito

- Um problema recursivo P pode ser expresso como

$$P \equiv P[Si,P],$$

  - onde P é a composição de comandos Si e do próprio P

- **Importante**: constantes e variáveis locais a P são duplicadas a cada chamada recursiva

Problema de Terminação

- Definir um condição de terminação.

- Ideia:
  - Associar um parâmetro, por exemplo n, com P e chamar P recursivamente com $n - 1$ como parâmetro.
  - A condição $n > 0$ garante a terminação.
  - Exemplo:
    - $P(n) \equiv$ if $n > 0$ then $P[S_i; P(n - 1)]$.

- **Importante**: na prática é necessário:
  - Mostrar que o nível de recursão é finito, e
  - Tem que ser mantido pequeno! Por que?

Razões para Limitar a Recursão

- Memória necessária para acomodar variáveis a cada chamada.
- O estado corrente da computação tem que ser armazenado para permitir a volta da chamada recursiva.

Exemplo:

```
function F(i : integer) : integer;
begin
  if i > 0
  then F := i * F(i-1)
  else F := 1;
end;
```

$F(4) \rightarrow$	1	$4*F(3)$
	2	$3*F(2)$
	3	$2*F(1)$
	4	$1*F(0)$
	1	

Quando Empregar Recursão?

- O problema está definido de forma recursiva
- A profundidade da recursão é relativamente pequena
- A conversão do algoritmo para a forma iterativa é difícil (exigindo memória auxiliar)
- A rotina não constitui parte crítica do programa

Análise de Algoritmos Recursivos

1. Equações de Recorrência
2. Método da Expansão Telescópica
3. Árvores de Recorrência
4. Teorema Mestre

Equações de Recorrência

- Para cada procedimento recursivo é associada uma função de complexidade *f(n)* desconhecida, onde *n* mede o tamanho dos argumentos para o procedimento.
- Obtemos uma equação de recorrência para *f(n)*.
- **Equação de recorrência**: maneira de definir uma função por uma expressão envolvendo a mesma função.
- Técnicas de solução:
  - Expansão telescópica
  - Árvore de recorrência
  - Método de substituição
  - Teorema mestre

Exemplo:

ALGORITMO

```
void Pesquisa(n) {
(1)   if (n <= 1) {
(2)     inspecione_elemento(n);
        return; }
      else {
(3)     for (int i=0; i<n; i++)
          inspecione_elemento(i);
(4)     Pesquisa(n/3);
       }
}
```

Análise do Procedimento:

- Seja *T*(*n*) uma função de complexidade que represente o número de inspeções nos *n* elementos do conjunto.
- O custo de execução das linhas (1) e (2) é $\Theta(1)$.
- O custo de execução da linha (3) é exatamente n.

Exemplo:

ALGORITMO

```
void Pesquisa(n) {
(1)   if (n <= 1) {
(2)     inspecione_elemento(n);
        return; }
      else {
(3)     for (int i=0; i<n; i++)
          inspecione_elemento(i);
(4)     Pesquisa(n/3);
       }
}
```

Análise do Procedimento:

- Usa-se uma equação de recorrência para determinar o no de chamadas recursivas.
- O termo $T(n)$ é especificado em função dos termos anteriores $T(1), T(2), \ldots, T(n-1)$.
- $T(n) = n + T(n/3)$;
- $T(1) = 1$
  (para n = 1 faz-se 1 insp.)

Exemplo:

ALGORITMO

```
void Pesquisa(n) {
(1)   if (n <= 1) {
(2)     inspecione_elemento(n);
        return; }
      else {
(3)     for (int i=0; i<n; i++)
          inspecione_elemento(i);
(4)     Pesquisa(n/3);
       }
}
```

Análise do Procedimento:

- $T(n) = n + T(n/3)$;
- $T(1) = 1$
  (para n = 1 faz-se 1 insp.)
- Por exemplo:
  - $T(3) = T(3/3) + 3 = 4$
  - $T(9) = T(9/3) + 9 = 13$
  - e assim por diante...
- Para calcular o valor da função seguindo a definição são necessários
  k -1 passos para computar o valor de $T(3^k)$.

Método da Expansão Telescópica

- Substituem-se os termos $T(k)$, $k < n$, até que todos os termos $T(k)$, $k > 1$, tenham sido substituídos por fórmulas contendo apenas $T(1)$:

$$
\begin{aligned}
T(n) &= n + T(n/3) \\
T(n/3) &= n/3 + T(n/3/3) \\
T(n/3/3) &= n/3/3 + T(n/3/3/3) \\
&\dots \\
T(n/3/3 \dots /3) &= n/3/3 \dots /3 + T(n/3 \dots /3)
\end{aligned}
$$

Método da Expansão Telescópica

- Adicionando lado a lado, temos

$$T(n) = n + n\,(1/3) + n\,(1/3^2) + n\,(1/3^3) + \ldots + T(n/3/3\ \ldots\ /3)$$

- que representa a soma de uma série geométrica de razão 1/3, multiplicada por *n*, e adicionada de $T(n/3/3\ \ldots\ /3)$, que é menor ou igual a 1.

Método da Expansão Telescópica

$T(n) = n + n\,(1/3) + n\,(1/3^2) + n\,(1/3^3) + \ldots + T(n/3/3\ \ldots\ /3)$

- *$T(n/3/3\ \ldots\ /3)$* será igual a 1 quando *$n/3/3\ \ldots\ /3=1$*.
  - Se considerarmos *x* o número de subdivisões por 3 do tamanho do problema, então $n/3^x=1$, logo $x=\log_3 n$.
  - Lembrando que T(1)=1, temos que:

$$\begin{aligned} T(n) &= \sum_{i=0}^{x-1} \frac{n}{3^i} + T\left(\frac{n}{3^x}\right) \\ &= n \sum_{i=0}^{x-1} (1/3)^i + 1 \\ &= \frac{n(1-(\frac{1}{3})^x)}{(1-\frac{1}{3})} + 1 \\ &= \frac{3n}{2} - \frac{1}{2}. \end{aligned}$$

Método da Expansão Telescópica

$T(n) = n + n\,(1/3) + n\,(1/3^2) + n\,(1/3^3) + \ldots + T(n/3/3 \ldots /3)$

- $T(n/3/3 \ldots /3)$ será igual a 1 quando $n/3/3 \ldots /3=1$.
  - Se considerarmos x o número de subdivisões por 3 do tamanho do problema, então $n/3^x=1$, logo $x=\log_3 n$.
  - Lembrando que T(1)=1, temos que:

$$
\begin{aligned}
T(n) &= \sum_{i=0}^{x-1} \frac{n}{3^i} + T\left(\frac{n}{3^x}\right) \\
&= n \sum_{i=0}^{x-1} (1/3)^i + 1 \\
&= \frac{n(1 - (\frac{1}{3})^x)}{(1 - \frac{1}{3})} + 1 \\
&= \frac{3n}{2} - \frac{1}{2}.
\end{aligned}
$$

  - Logo, o programa do exemplo é $\Theta(n)$

Comentários Sobre Recursão

- Evitar o uso de recursividade quando existe uma solução óbvia por iteração!

- Exemplos:
  - Fatorial
  - Série de Fibonacci

Exemplo: MergeSort

ALGORITMO

```
void merge_sort(int A[],
                int p, int r)
{
   if (p < r) {
      q = (p+r)/2;
      merge_sort(A,p,q);
      merge_sort(A,q+1,r);
      merge(A,p,q,r);
  }
}
```

Análise do Pior Caso

- Invocação inicial: merge_sort(A,1,n); em que *A* contém *n* elementos.
-
- Qual a dimensão de cada sub-seqüência criada no passo de divisão?

Exemplo: MergeSort

- Simplificação da análise de “merge sort”: tamanho da entrada é uma potência de 2. Em cada divisão, as sub-sequências têm tamanho exatamente $n/2$.

- Seja $T(n)$ o tempo de execução (no pior caso) sobre uma entrada de tamanho n.

Exemplo: MergeSort

- Se $n = 1$, esse tempo é constante, que escrevemos:

$$T(n)=\Theta(1)$$

- Senão:
  - O cálculo da posição do meio do vetor é feita em tempo constante:

$$D(n) = \Theta(1)$$

  - São resolvidos dois problemas, cada um de tamanho $n/2$; o tempo total para isto é:

$$2T(n/2)$$

  - A função merge executa em tempo linear:

$$C(n) = \Theta(n)$$

Exemplo: MergeSort

- Desta forma:

  $$T(n) = \Theta(1), \text{ se } n = 1;$$
  $$= \Theta(1) + 2T(n/2) + \Theta(n), \text{ se } n > 1$$

- Considerando o número de comparações como operação crítica:

  $$T(n) = 0, \text{ se } n = 1;$$
  $$= 2T(n/2) + n, \text{ se } n > 1$$

Exemplo: MergeSort

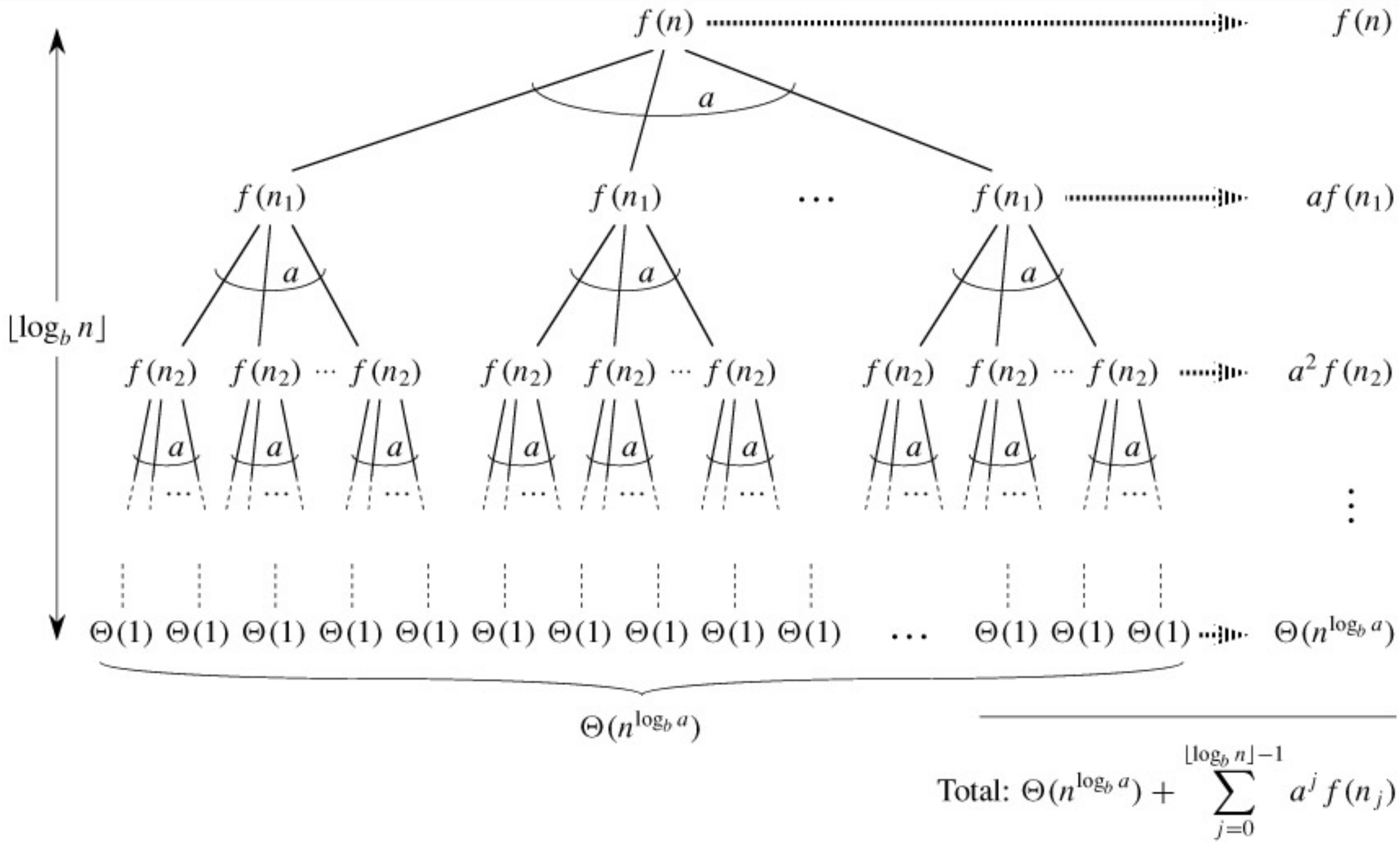
- Solução por Expansão Telescópica

$$T(n) = 0, \text{ se } n = 1;$$
$$= 2T(n/2) + n, \text{ se } n > 1$$

Árvore de Recursão

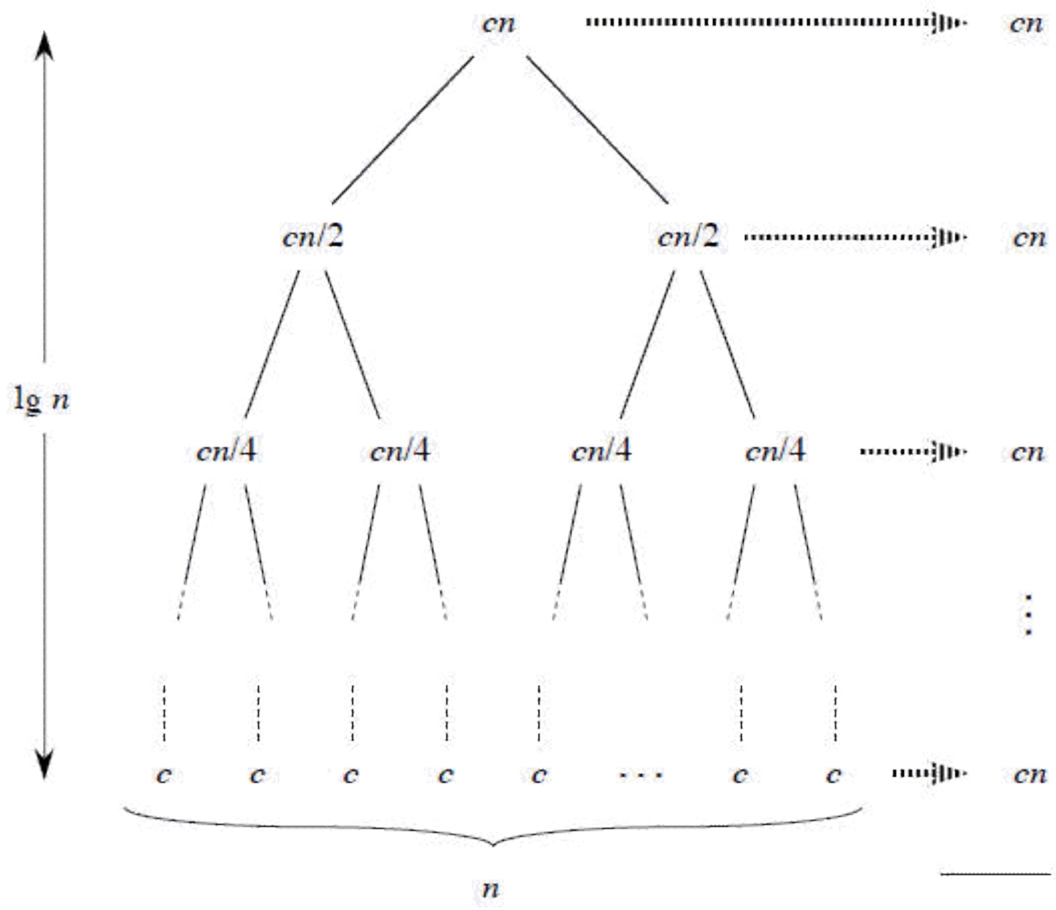
- Cada nodo da árvore contém o custo daquele nível de recursão.
- Chamadas recursivas são filhos do nodo.
- Nodos folha contêm o custo da condição de término da recorrência.
- O custo final é a soma dos custos dos nodos. A altura da árvore deve ser calculada.
- Pode ser usada para se supor uma função de custo através de exemplos.
  - A solução deve ser provada por indução.

Árvore de Recursão



Exemplo

- MergeSort: $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$



Exemplo

- MergeSort: $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$$

  PROVA POR INDUÇÃO:

  - Caso base: n=1
    - Implica que tem 1 nível: lg 1 + 1 = 0 + 1 = 1
  - Passo indutivo:
    - Nossa hipótese indutiva é que a árvore para um problema de tamanho 2i tem lg 2i + 1 = i + 1 níveis.
    - Porque nós assumimos que o problema é potência de 2, o próximo problema aumenta de 2i para 2i + 1.
    - Uma árvore para o problema de tamanho 2i + 1 tem um nível a mais que o problema de tamanho 2i implicando em i + 2 níveis.
    - Já que lg 2i + 1 = i + 2, indicamos a indução.

Exemplo

- MergeSort: $T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\,, \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\,. \end{cases}$

PROVA POR INDUÇÃO:

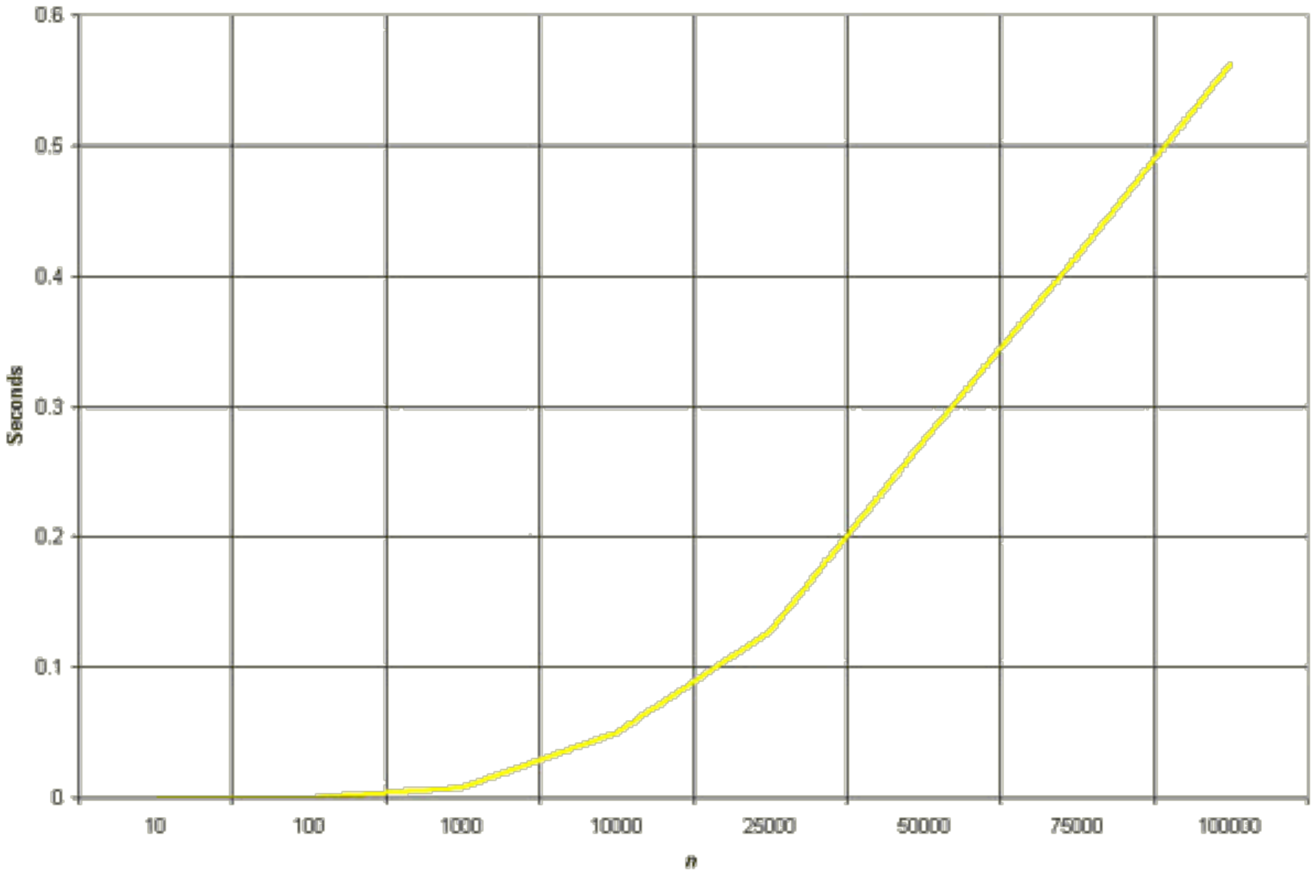
- Caso base: n=1
  - Implica que tem 1 nível: lg 1 + 1 = 0 + 1 = 1
- Passo indutivo:
  - O custo total é a soma dos custos de cada nível da árvore.
  - Já que temos lg n +1 níveis, cada um com um custo de cn, o custo total é:

$$cn \lg n + cn$$

  - Depois de provar que cn lg n + cn = Θ(n lg n), cqd!

Exemplo

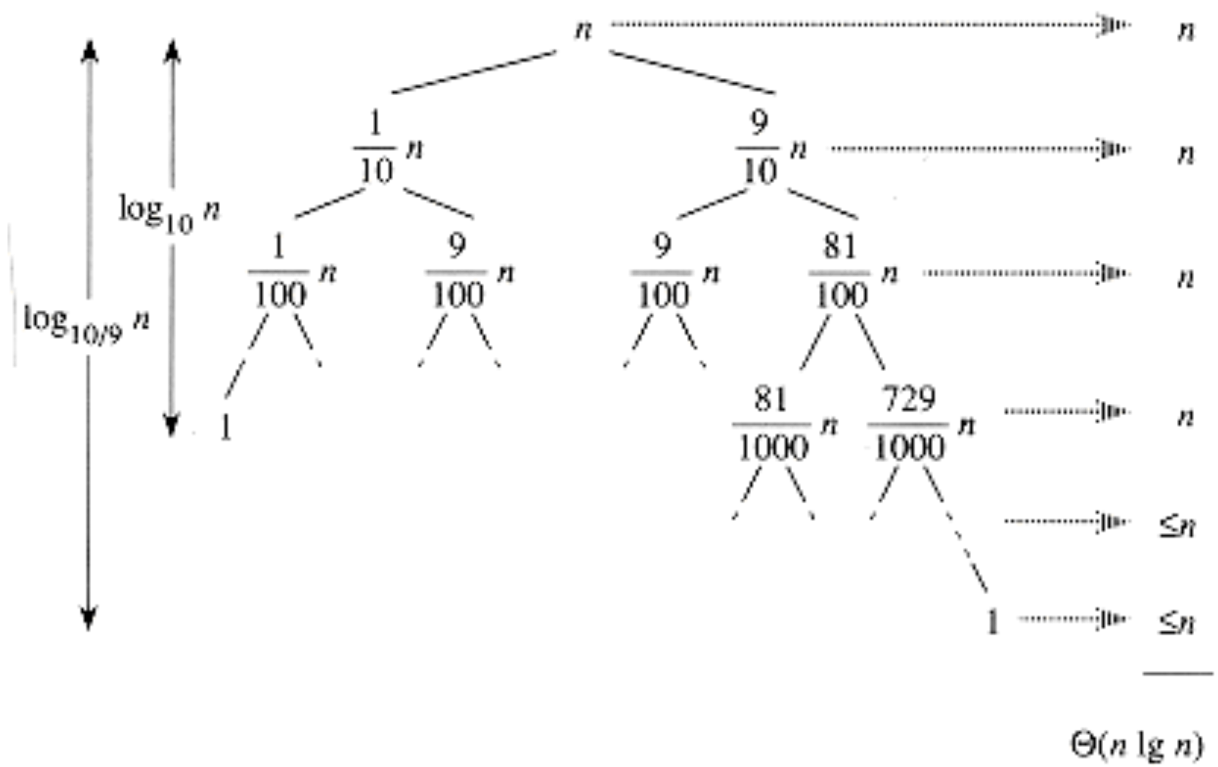
- MergeSort: $$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1\ , \\ 2T(n/2) + \Theta(n) & \text{if } n > 1\ . \end{cases}$$



Exemplo

- Quicksort: (caso médio)

$$T(n) = T(9n/10) + T(n/10) + n$$



Alguns Somatórios Úteis

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

$$\sum_{i=0}^{k} a^i = \frac{a^{k+1} - 1}{a - 1} (a \neq 1)$$

$$\sum_{i=0}^{k} 2^k = 2^{k+1} - 1$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=0}^{k} \frac{1}{2^i} = 2 - \frac{1}{2^k}$$

Teorema Mestre

- Recorrências da forma
  - $T(n) = aT(n/b) + f(n)$,

- onde $a \geq 1$ e $b > 1$ são constantes e $f(n)$ é uma função assintoticamente positiva podem ser resolvidas usando o Teorema Mestre.

- Note que neste caso não estamos achando a forma fechada da recorrência mas sim seu comportamento assintótico.

Teorema Mestre

- Sejam as constantes a ≥ 1 e b > 1 e f(n) uma função definida nos inteiros não-negativos pela recorrência:

$$T(n) = aT(n/b) + f(n),$$

- onde a fração n/b pode significar $\lfloor n/b \rfloor$ ou $\lceil n/b \rceil$.

Teorema Mestre

- A equação de recorrência T(n) pode ser limitada assintoticamente da seguinte forma:

1. Se $f(n) = O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$, então $\boxed{T(n) = \Theta(n^{\log_b a})}$.
2. Se $f(n) = \Theta(n^{\log_b a})$, então $\boxed{T(n) = \Theta(n^{\log_b a} \log n)}$.
3. Se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguma constante $\epsilon > 0$ e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e para n suficientemente grande, então $\boxed{T(n) = \Theta(f(n))}$.

Teorema Mestre

- Nos três casos estamos comparando a função f(n) com a função $n^{\log_b a}$. Intuitivamente, a solução da recorrência é determinada pela maior das duas funções.
- Por exemplo:
  - No primeiro caso a função nlogb a é a maior e a solução para a recorrência é $T(n) = \Theta(n^{\log_b a})$.
  - No terceiro caso, a função f(n) é a maior e a solução para a recorrência é $T(n) = \Theta(f(n))$.
  - No segundo caso, as duas funções são do mesmo "tamanho"
    - Neste caso, a solução fica multiplicada por um fator logarítmico e fica da forma:

$$T(n) = \Theta(n^{\log_b a} \log n) = \Theta(f(n) \log n).$$

Teorema Mestre

- No primeiro caso, a função f(n) deve ser não somente menor que $n^{\log_b a}$ mas ser polinomialmente menor. Ou seja, f(n) deve ser assintoticamente menor que $n^{\log_b a}$ por um fator de n^{ε}, para alguma constante $\varepsilon > 0$.

- No terceiro caso, a função f(n) deve ser não somente maior que $n^{\log_b a}$ mas ser polinomialmente maior e satisfazer a condição de “regularidade” que $af(n/b) \leq cf(n)$
  - Esta condição é satisfeita pela maior parte das funções polinomiais encontradas neste curso.

Exemplo

$$T(n) = 9T(n/3) + n$$

Temos que,

$$a = 9, b = 3, f(n) = n$$

Desta forma,

$$n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$$

Como $f(n) = O(n^{\log_3 9 - \epsilon})$, onde $\epsilon = 1$, podemos aplicar o caso 1 do teorema e concluir que a solução da recorrência é

$$T(n) = \Theta(n^2)$$

Exemplo

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

Temos que,

$$a = 1, b = 3/2, f(n) = 1$$

Desta forma,

$$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$$

O caso 2 se aplica já que $f(n) = \Theta(n^{\log_b a}) = \Theta(1)$. Temos, então, que a solução da recorrência é

$$T(n) = \Theta(\log n)$$

Exemplo

$$T(n) = 3T(n/4) + n \log n$$

Temos que,

$$a = 3, b = 4, f(n) = n \log n$$

Desta forma,

$$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$

Como $f(n) = \Omega(n^{\log_4 3+\epsilon})$, onde $\epsilon \approx 0.2$, o caso 3 se aplica se mostrarmos que a condição de regularidade é verdadeira para $f(n)$.

Exemplo

Para um valor suficientemente grande de n

$$af(n/b) = 3(n/4)\log(n/4) \leq (3/4)n\log n = cf(n)$$

para $c = 3/4$. Conseqüentemente, usando o caso 3, a solução para a recorrência é

$$T(n) = \Theta(n \log n)$$

Exemplo

$$T(n) = 2T(n/2) + n \log n$$

Temos que,

$$a = 2, b = 2, f(n) = n \log n$$

Desta forma,

$$n^{\log_b a} = n$$

Aparentemente o caso 3 deveria se aplicar já que $f(n) = n \log n$ é assintoticamente maior que $n^{\log_b a} = n$. Mas no entanto, não é polinomialmente maior. A fração $f(n)/n^{\log_b a} = (n \log n)/n = \log n$ que é assintoticamente menor que n^{ϵ} para toda constante positiva ϵ. Conseqüentemente, a recorrência cai na situação entre os casos 2 e 3 onde o teorema não pode ser aplicado.

Exercícios

1. Encontre e resolva a equação de recorrência do Quicksort para o pior caso.
2. Mostre o tempo de execução do Quicksort é $\Theta(n \log n)$ quando todos os elementos do vetor tem o mesmo valor.
3. Resolva a seguinte equação de recorrência pelo método de expansão: $T(n) = c + T(n-1)$, sendo c uma constante.
4. Resolva pelo teorema mestre:
   1. $T(n) = 4T(n/2) + n$
   2. $T(n) = 4T(n/2) + n^2$
   3. $T(n) = 4T(n/2) + n^3$
5. Encontre e resolva a equação de recorrência do problema da Torre de Hanoi.

Exercícios

1. Encontre e resolva a equação de recorrência do Quicksort para o pior caso.

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

   - Para avaliar essa recorrência, observe que $T(1) = (1)$.
   - Então, itere:

$$\begin{aligned} T(n) &= T(n-1) + \Theta(n) \\ &= \sum_{k=1}^{n} \Theta(k) \\ &= \Theta\left(\sum_{k=1}^{n} k\right) \\ &= \Theta(n^2) . \end{aligned}$$

Exercícios

1. Encontre e resolva a equação de recorrência do Quicksort para o pior caso.
2. Mostre o tempo de execução do Quicksort é $\Theta(n \log n)$ quando todos os elementos do vetor tem o mesmo valor.
3. Resolva a seguinte equação de recorrência pelo método de expansão: T(n) = c + T(n-1), sendo c uma constante.

Esta equação de recorrência pode ser expressa da seguinte forma:

$$\begin{aligned}
T(n) &= c + T(n-1) \\
&= c + (c + T(n-2)) \\
&= c + c + (c + T(n-3)) \\
\vdots &= \vdots \\
&= c + c + \cdots + (c + T(1)) \\
&= \underbrace{c + c + \cdots + c}_{n-1} + d
\end{aligned}$$

Em cada passo, o valor do termo T é substituído pela sua definição (ou seja, esta recorrência está sendo resolvida pelo método da expansão). A última equação mostra que depois da expansão existem $n-1$ c's, correspondentes aos valores de 2 até n. Desta forma, a recorrência pode ser expressa como:

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