Análise e Síntese de Algoritmos

Revisão [CLRS, Cap. 4-6]

2010/2011

Contexto

- Revisão [CLRS, Cap.1-13]
  - Fundamentos; notação; exemplos
- Algoritmos em Grafos [CLRS, Cap.21-26]
  - Algoritmos elementares
  - Árvores abrangentes
  - Caminhos mais curtos
  - Fluxos máximos
- Programação Linear [CLRS, Cap.29]
  - Algoritmos e modelação de problemas com restrições lineares
- Técnicas de Síntese de Algoritmos [CLRS, Cap.15-16]
  - Programação dinâmica
  - Algoritmos greedy
- Tópicos Adicionais [CLRS, Cap.32-35]
  - Emparelhamento de Cadeias de Caracteres
  - Complexidade Computacional
  - Algoritmos de Aproximação

Resumo

1. Recorrências
   - Teorema Mestre
   - Exemplos de Recorrências

2. Amontoados (Heaps)
   - Operações sobre Amontoados
   - Algoritmo Heap-Sort
   - Outras Operações

Motivação

Merge-Sort(A, p, r)

if $p < r$
  then $q \leftarrow \lfloor (p + r)/2 \rfloor$
    Merge-Sort(A, p, q)
    Merge-Sort$(A, q + 1, r)$
    Merge(A, p, q, r)

- Qual o tempo execução no pior caso?
  - Admitindo que tempo de execução de Merge cresce com n
- Qual a recorrência para este procedimento?

Recorrências

- Utilizadas para exprimir e calcular do tempo de execução de **procedimentos recursivos**

Resolução de Recorrências

- Por Substituição
  - Sugestão de solução
  - Provar formalmente a solução utilizando indução
- Por Iteração
  - Expandir recorrência com o intuito de a expressar em termos apenas de n e das condições iniciais
- Teorema Mestre

Teorema Mestre

Permite resolver recorrências da forma

$$T(n) = aT(n/b) + f(n) \quad , \quad a \geq 1 \, , \; b > 1$$

Problema é dividido em a subproblemas, cada um com dimensão n/b

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a} \; lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

Teorema Mestre

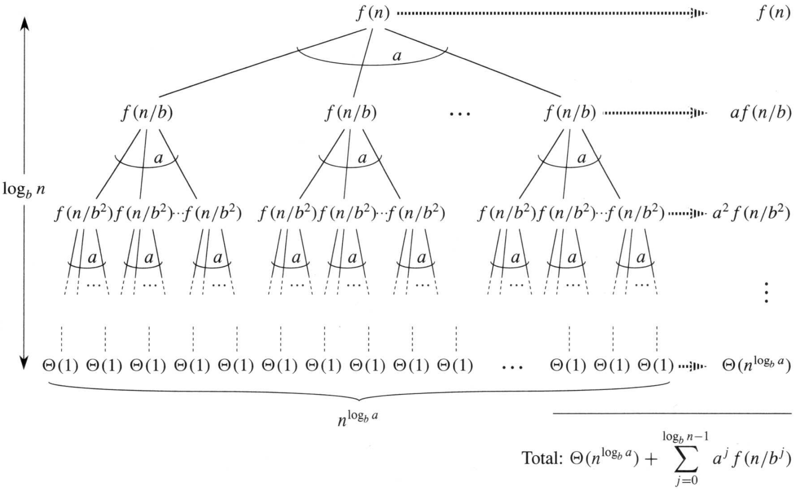
Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

- Em cada um dos 3 casos estamos a comparar $f(n)$ com $n^{log_b a}$
- A solução da recorrência é determinada pela maior das duas funções

Teorema Mestre



Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

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

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

- $a = 9, b = 3, f(n) = n$
- $f(n) = O(n^{2-\varepsilon})$
- $T(n) = \Theta(n^2)$ (caso 1 do Teorema Mestre)

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

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

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

- $a = 1, b = 3/2, f(n) = 1$
- $f(n) = \Theta(n^0)$
- $T(n) = \Theta(lgn)$ (caso 2 do Teorema Mestre)

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

- $a = 3, b = 3, f(n) = n^2$

Teorema Mestre: Exemplos de Aplicação

Definition (Teorema Mestre)

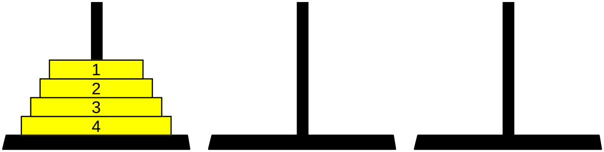
Sejam $a \geq 1, b > 1$ constantes, seja $f(n)$ uma função, e seja $T(n)$ definido por $T(n) = aT(n/b) + f(n)$. Então $T(n)$ é limitado assimptoticamente da seguinte forma:

1. Se $f(n) = O(n^{log_b a - \varepsilon})$ para $\varepsilon > 0$, então $T(n) = \Theta(n^{log_b a})$
2. Se $f(n) = \Theta(n^{log_b a})$, então $T(n) = \Theta(n^{log_b a}\ lgn)$
3. Se $f(n) = \Omega(n^{log_b a + \varepsilon})$ para $\varepsilon > 0$, e $a \cdot f(n/b) < c \cdot f(n)$ para alguma constante $c < 1$ e n suficientemente grande, então $T(n) = \Theta(f(n))$

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

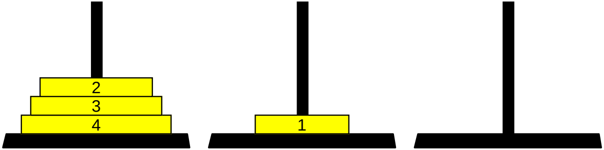
- $a = 3, b = 3, f(n) = n^2$
- $f(n) = \Omega(n^{1+\varepsilon})$
- $T(n) = \Theta(n^2)$ (caso 3 do Teorema Mestre)

Torres de Hanoi



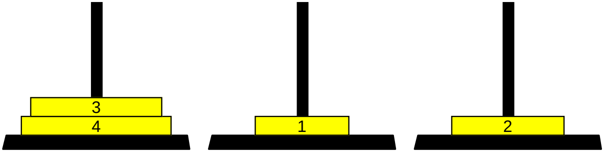
- 3 suportes
- *n* discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os *n* discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os *n* discos de um suporte para outro ?

Torres de Hanoi



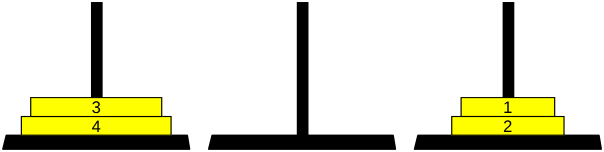
- 3 suportes
- *n* discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os *n* discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os *n* discos de um suporte para outro ?

Torres de Hanoi



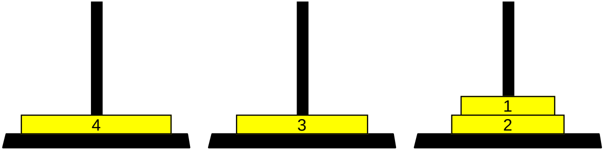
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



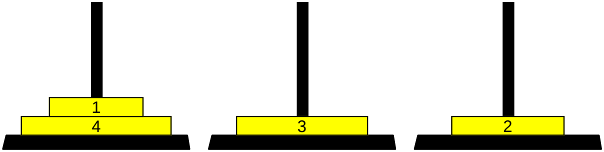
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



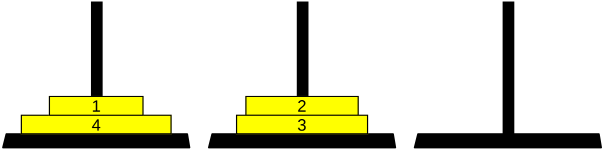
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



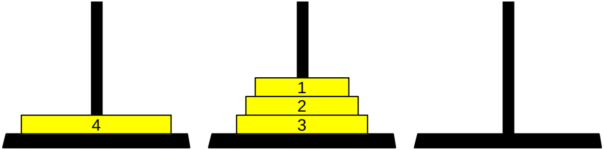
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



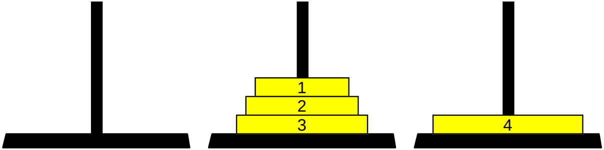
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



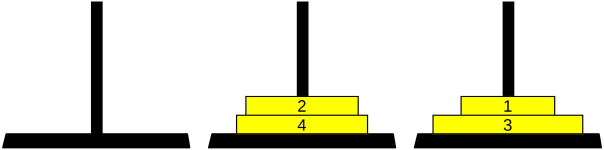
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



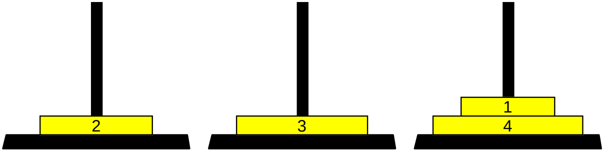
- 3 suportes
- *n* discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os *n* discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os *n* discos de um suporte para outro ?

Torres de Hanoi



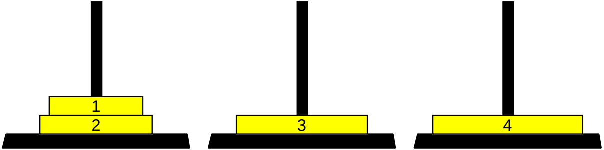
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



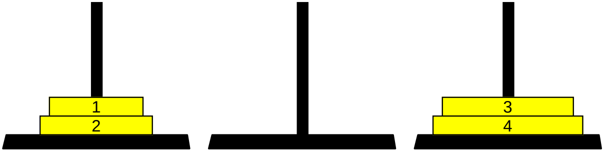
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



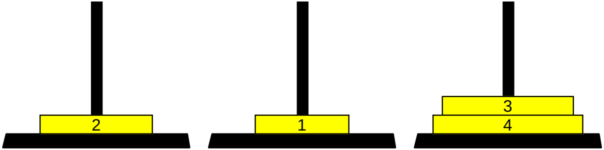
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



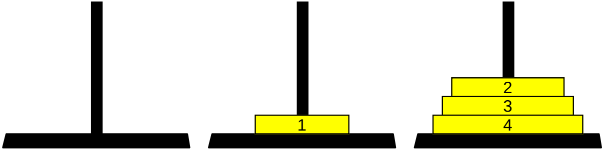
- 3 suportes
- *n* discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os *n* discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os *n* discos de um suporte para outro ?

Torres de Hanoi



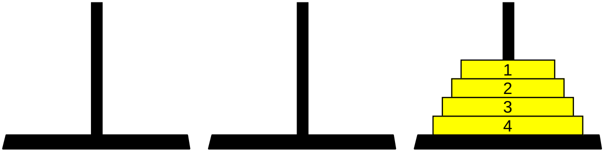
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



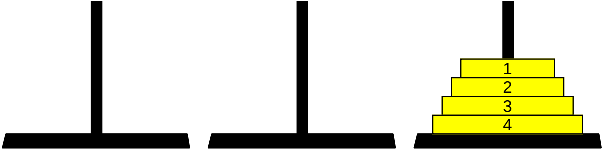
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ?

Torres de Hanoi



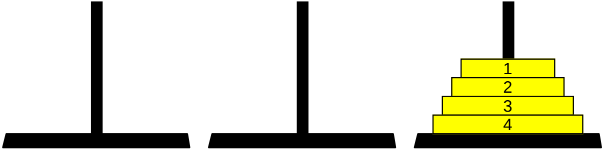
- 3 suportes
- n discos colocados num suporte, por ordem decrescente de tamanho
- Transferir os n discos para outro suporte, mantendo sempre a relação de ordem entre eles
- Qual o menor número de movimentos de disco individuais que é necessário e suficiente para deslocar os n discos de um suporte para outro ? Para $n = 4$, foram necessários 15 movimentos

Torres de Hanoi



- T_n - menor número de movimentos de disco individuais, necessário e suficiente para transferir n discos, de um suporte para outro
- $T_0 = 0, \; T_1 = 1$
- T_n ?
  - Transferir $n - 1$ discos para outro suporte (T_{n-1} movimentos)
  - Deslocar maior disco para suporte final
  - Transferir $n - 1$ discos para o suporte final (T_{n-1} movimentos)

Torres de Hanoi



- T_n - menor número de movimentos de disco individuais, necessário e suficiente para transferir n discos, de um suporte para outro
- $T_0 = 0, \; T_1 = 1$
- T_n ?
  - Transferir $n-1$ discos para outro suporte (T_{n-1} movimentos)
  - Deslocar maior disco para suporte final
  - Transferir $n-1$ discos para o suporte final (T_{n-1} movimentos)
- $T_n = 2T_{n-1} + 1$, para $n > 0$

Torres de Hanoi

Recorrência

- $T_0 = 0$
- $T_n = 2T_{n-1} + 1$, para $n > 0$

Solução

- Por substituição
- Tentativa: $T_n = 2^n - 1$
  - Substituir por $T_n = 2^n$ não resulta. Expressão tem que ser correcta.
- Substituindo $T_{n-1} = 2^{n-1} - 1$ em $T_n = 2T_{n-1} + 1$, obtemos $T_n = 2\left(2^{n-1} - 1\right) + 1 = 2^n - 1$!

Multiplicação de Matrizes [CLRS, Sec. 28.2]

Matrix-Multiplication(A, B)

```
n ← número de linhas de A
for i ← 1 to n
    do for j ← 1 to n
        do c_ij ← 0
           for k ← 1 to n
               do c_ij ← c_ij + a_ik b_kj
return C
```

Multiplicação de Matrizes [CLRS, Sec. 28.2]

Algoritmo usual (3 ciclos): $\Theta(n^3)$

Algoritmo usual (recursivo): $\Theta(n^3)$ $T(n) = 8T(n/2) + \Theta(n^2)$

Algoritmo de Strassen (recursivo)

- Matrizes $n \times n$
- $T(n) = 7T(n/2) + \Theta(n^2)$
- Caso 1 do Teorema Mestre com $a = 7, b = 2$ e $f(n) = n^2$
- $\Theta(n^{lg 7}) = O(n^{2.81})$

Amontoados: Propriedades

Vector de valores interpretado como uma árvore binária (essencialmente completa)

- $length[A]$: tamanho do vector
- $heap\text{-}size[A]$: número de elementos no amontoado
- $A[1]$: Raíz da árvore

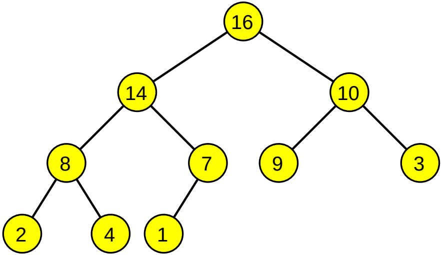
Relações entre nós da árvore

- $\text{Parent}(i) = \lfloor i/2 \rfloor$
- $\text{Left}(i) = 2i$
- $\text{Right}(i) = 2i + 1$

Propriedade de amontoado

- $A[\text{Parent}(i)] \geq A[i]$

Amontoados: Exemplo



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Árvores Binárias

Árvore Binária Completa

- Cada nó tem 2 (ou 0) filhos
- Qualquer nó folha (i.e. nó sem descendentes) tem uma mesma profundidade d
  - Profundidade: número de nós entre a raiz e um dado nó
- Todos os nós internos tem grau 2
  - Cada nó interno tem exactamente 2 filhos

Árvore Binária Essencialmente Completa

- Todos os nós internos têm grau 2, com 1 possível excepção
  - Existe nó à profundidade $d - 1$ com filho esquerdo, mas sem filho direito
- Nós folha posicionados à profundidade d ou $d - 1$
  - Qualquer nó interno à profundidade $d - 1$, posicionado à esquerda de qualquer nó folha à mesma profundidade

Operação Max-Heapify

Max-Heapify

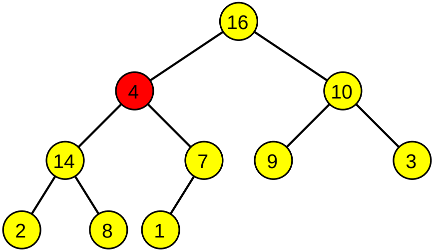
- Transforma a árvore com raiz em i num amontoado, assumindo que as árvores com raiz em Left(i) e Right(i) são amontoados
- Troca recursivamente valores entre elementos que não verifiquem a propriedade do amontoado
- Complexidade:
  - Altura da Heap: $h = \lfloor lg\ n \rfloor$
  - Complexidade de Max-Heapify: $O(h) = O(lg\ n)$

Operação Max-Heapify

Max-Heapify(A, i)

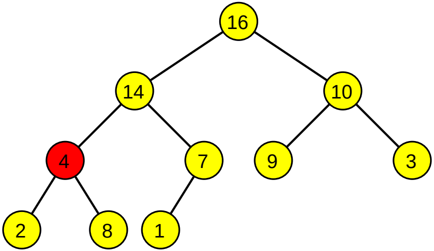
```
l ← Left(i)
r ← Right(i)
if l ≤ heap-size[A] and A[l] > A[i]
    then largest ← l
    else largest ← i
if r ≤ heap-size[A] and A[r] > A[largest]
    then largest ← r
if largest ≠ i
    then exchange A[i] ↔ A[largest]
         Max-Heapify(A, largest)
```

Operação Max-Heapify: Exemplo



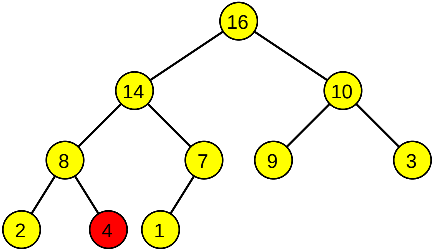
1	2	3	4	5	6	7	8	9	10
16	4	10	14	7	9	3	2	8	1

Operação Max-Heapify: Exemplo



1	2	3	4	5	6	7	8	9	10
16	14	10	4	7	9	3	2	8	1

Operação Max-Heapify: Exemplo



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

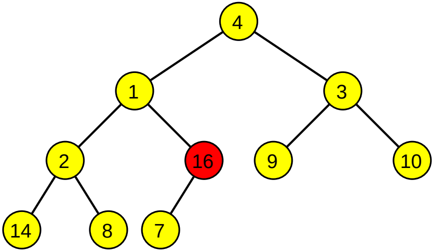
Operação Build-Max-Heap

- Construir amontoado a partir de um vector arbitrário
- Chamada selectiva de Max-Heapify

Build-Max-Heap(A)

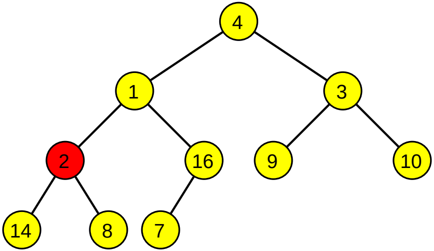
$\textit{heap-size}[A] \leftarrow \textit{length}[A]$
for $i \leftarrow \lfloor \textit{length}[A]/2 \rfloor$ **downto** 1
    do Max-Heapify(A, i)

Operação Build-Max-Heap: Exemplo



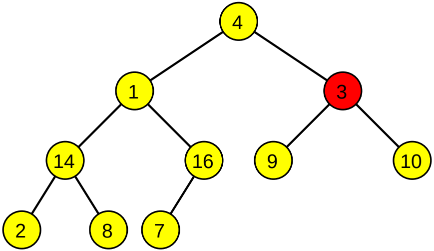
1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

Operação Build-Max-Heap: Exemplo



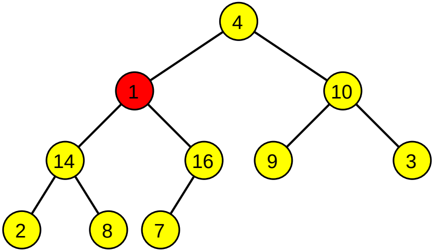
1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

Operação Build-Max-Heap: Exemplo



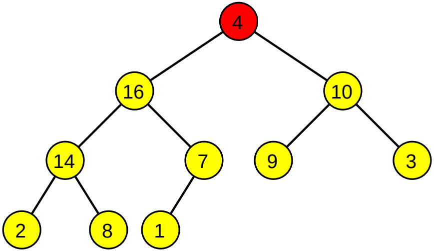
1	2	3	4	5	6	7	8	9	10
4	1	3	14	16	9	10	2	8	7

Operação Build-Max-Heap: Exemplo



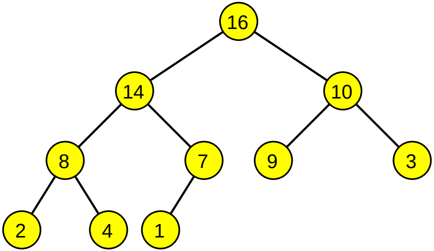
1	2	3	4	5	6	7	8	9	10
4	1	10	14	16	9	3	2	8	7

Operação Build-Max-Heap: Exemplo



1	2	3	4	5	6	7	8	9	10
4	16	10	14	7	9	3	2	8	1

Operação Build-Max-Heap: Exemplo



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Operação Build-Max-Heap

- Construir amontoado a partir de um vector arbitrário
- Chamada selectiva de Max-Heapify

Build-Max-Heap(A)

$\textit{heap-size}[A] \leftarrow \textit{length}[A]$
for $i \leftarrow \lfloor \textit{length}[A]/2 \rfloor$ **downto** 1
    do Max-Heapify(A, i)

Operação Build-Max-Heap

- Construir amontoado a partir de um vector arbitrário
- Chamada selectiva de Max-Heapify

Build-Max-Heap(A)

1 $\textit{heap-size}[A] \leftarrow \textit{length}[A]$
2 **for** $i \leftarrow \lfloor \textit{length}[A]/2 \rfloor$ **downto** 1
3 **do** Max-Heapify(A, i)

Complexidade ?

Operação Build-Max-Heap

- Construir amontoado a partir de um vector arbitrário
- Chamada selectiva de Max-Heapify

Build-Max-Heap(A)

1 $\quad$ *heap-size*$[A] \leftarrow$ *length*$[A]$
2 $\quad$ **for** $i \leftarrow \lfloor length[A]/2 \rfloor$ **downto** 1
3 $\qquad\quad$ **do** Max-Heapify(A, i)

Complexidade ? $O(n\ lg\ n)$

Operação Build-Max-Heap

- Construir amontoado a partir de um vector arbitrário
- Chamada selectiva de Max-Heapify

Build-Max-Heap(A)

1 $heap\text{-}size[A] \leftarrow length[A]$
2 **for** $i \leftarrow \lfloor length[A]/2 \rfloor$ **downto** 1
3 **do** Max-Heapify(A, i)

Complexidade ? $O(n\ lg\ n)$, mas é possível provar $O(n)$

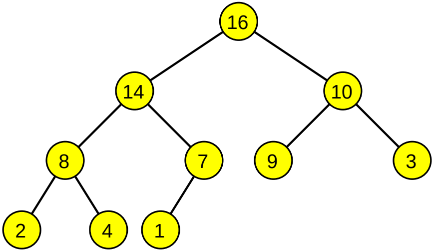
Ordenação: Heap-Sort

- Intuição
  - Extrair consecutivamente o elemento máximo de uma heap
  - Colocar esse elemento na posição (certa) do vector

Heap-Sort(A)

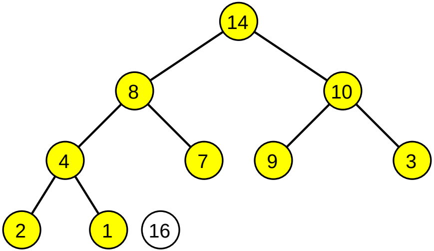
```
Build-Max-Heap(A)
for i ← length[A] downto 2
     do exchange A[1] ↔ A[i]
        heap-size[A] ← heap-size[A] − 1
        Max-Heapify(A, 1)
```

Operação Heap-Sort: Exemplo



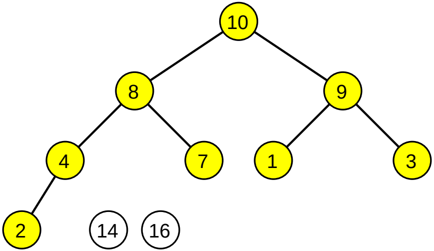
1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Operação Heap-Sort: Exemplo



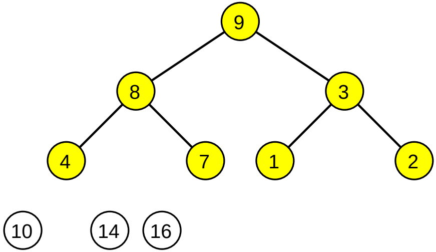
1	2	3	4	5	6	7	8	9	10
14	8	10	4	7	9	3	2	1	16

Operação Heap-Sort: Exemplo



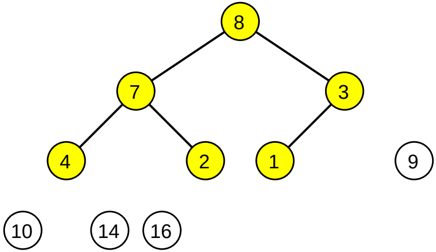
1	2	3	4	5	6	7	8	9	10
10	8	9	4	7	1	3	2	14	16

Operação Heap-Sort: Exemplo



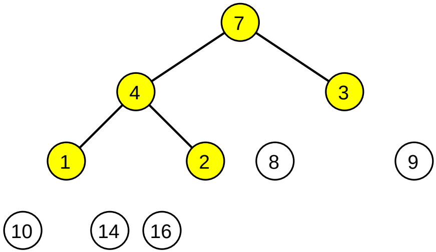
1	2	3	4	5	6	7	8	9	10
9	8	3	4	7	1	2	10	14	16

Operação Heap-Sort: Exemplo



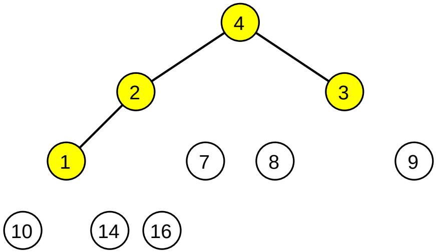
1	2	3	4	5	6	7	8	9	10
8	7	3	4	2	1	9	10	14	16

Operação Heap-Sort: Exemplo



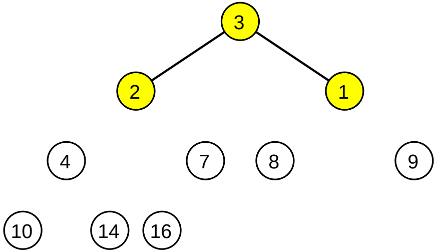
1	2	3	4	5	6	7	8	9	10
7	4	3	1	2	8	9	10	14	16

Operação Heap-Sort: Exemplo



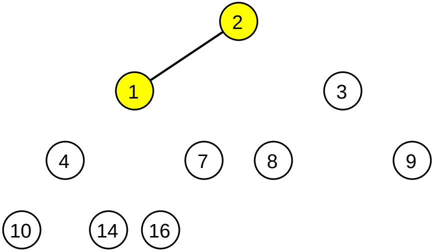
1	2	3	4	5	6	7	8	9	10
4	2	3	1	7	8	9	10	14	16

Operação Heap-Sort: Exemplo



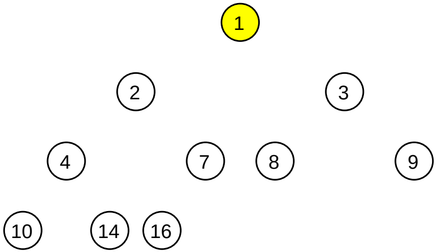
1	2	3	4	5	6	7	8	9	10
3	2	1	4	7	8	9	10	14	16

Operação Heap-Sort: Exemplo



1	2	3	4	5	6	7	8	9	10
2	1	3	4	7	8	9	10	14	16

Operação Heap-Sort: Exemplo



1	2	3	4	5	6	7	8	9	10
1	2	3	4	7	8	9	10	14	16

Ordenação: Heap-Sort

- Intuição
  - Extrair consecutivamente o elemento máximo de uma heap
  - Colocar esse elemento na posição (certa) do vector

Heap-Sort(A)

```
Build-Max-Heap(A)
for i ← length[A] downto 2
    do exchange A[1] ↔ A[i]
       heap-size[A] ← heap-size[A] − 1
       Max-Heapify(A, 1)
```

Ordenação: Heap-Sort

- Intuição
  - Extrair consecutivamente o elemento máximo de uma heap
  - Colocar esse elemento na posição (certa) do vector

Heap-Sort(A)

```
Build-Max-Heap(A)
for i ← length[A] downto 2
    do exchange A[1] ↔ A[i]
        heap-size[A] ← heap-size[A] − 1
        Max-Heapify(A, 1)
```

Complexidade ?

Ordenação: Heap-Sort

- Intuição
  - Extrair consecutivamente o elemento máximo de uma heap
  - Colocar esse elemento na posição (certa) do vector

Heap-Sort(A)

```
Build-Max-Heap(A)
for i ← length[A] downto 2
    do exchange A[1] ↔ A[i]
       heap-size[A] ← heap-size[A] − 1
       Max-Heapify(A, 1)
```

Complexidade ? $O(n\ lg\ n)$

Heap-Max e Heap-Extract-Max

Heap-Maximum(A)

1 **return** $A[1]$

Heap-Max e Heap-Extract-Max

Heap-Maximum(A)

1 **return** $A[1]$

Complexidade ?

Heap-Max e Heap-Extract-Max

Heap-Maximum(A)

1 **return** $A[1]$

Complexidade ? $O(1)$

Heap-Extract-Max(A)

1 $max \leftarrow A[1]$
2 $A[1] \leftarrow A[heap\text{-}size[A]]$
3 $heap\text{-}size[A] \leftarrow heap\text{-}size[A] - 1$
4 Max-Heapify$(A, 1)$
5 **return** max

Heap-Max e Heap-Extract-Max

Heap-Maximum(A)

return $A[1]$

Complexidade ? $O(1)$

Heap-Extract-Max(A)

$max \leftarrow A[1]$
$A[1] \leftarrow A[heap\text{-}size[A]]$
$heap\text{-}size[A] \leftarrow heap\text{-}size[A] - 1$
Max-Heapify$(A, 1)$
return max

Complexidade ?

Heap-Max e Heap-Extract-Max

Heap-Maximum(A)

1 **return** $A[1]$

Complexidade ? $O(1)$

Heap-Extract-Max(A)

1 $max \leftarrow A[1]$
2 $A[1] \leftarrow A[heap\text{-}size[A]]$
3 $heap\text{-}size[A] \leftarrow heap\text{-}size[A] - 1$
4 Max-Heapify$(A, 1)$
5 **return** max

Complexidade ? $O(lg\ n)$

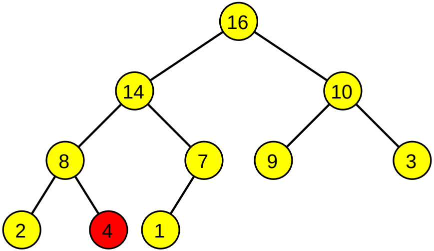
Min e Extract-Min ?

Heap-Increase-Key e Max-Heap-Insert

Heap-Increase-Key(A, i, key)

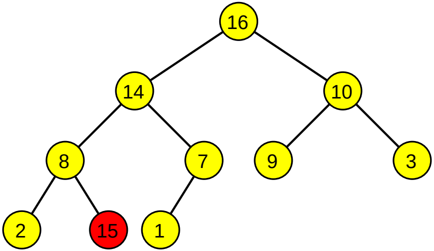
$A[i] \leftarrow key$
while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    do exchange $A[i] \leftrightarrow [\text{Parent}(i)]$
        $i \leftarrow \text{Parent}(i)$

Operação Heap-Increase-Key: Exemplo



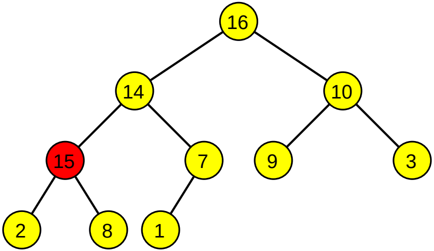
1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

Operação Heap-Increase-Key: Exemplo



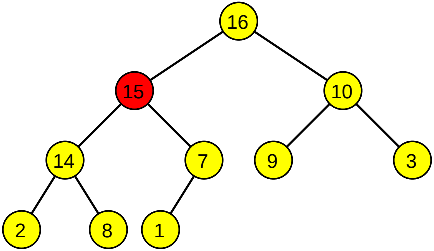
1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	15	1

Operação Heap-Increase-Key: Exemplo



1	2	3	4	5	6	7	8	9	10
16	14	10	15	7	9	3	2	8	1

Operação Heap-Increase-Key: Exemplo



1	2	3	4	5	6	7	8	9	10
16	15	10	14	7	9	3	2	8	1

Heap-Increase-Key e Max-Heap-Insert

Heap-Increase-Key(A, i, key)

$A[i] \leftarrow key$
while $i > 1$ and $A[\text{Parent}(i) < A[i]$
  do exchange $A[i] \leftrightarrow [\text{Parent}(i)]$
    $i \leftarrow \text{Parent}(i)$

Heap-Increase-Key e Max-Heap-Insert

Heap-Increase-Key(A, i, key)

$A[i] \leftarrow key$
while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    do exchange $A[i] \leftrightarrow [\text{Parent}(i)]$
        $i \leftarrow \text{Parent}(i)$

Complexidade ?

Heap-Increase-Key e Max-Heap-Insert

Heap-Increase-Key(A, i, key)

$A[i] \leftarrow key$
while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    do exchange $A[i] \leftrightarrow [\text{Parent}(i)]$
        $i \leftarrow \text{Parent}(i)$

Complexidade ? $O(lg\ n)$

Max-Heap-Insert(A, key)

$heap\text{-}size[A] \leftarrow heap\text{-}size[A] + 1$
$A[heap\text{-}size] \leftarrow -\infty$
Heap-Increase-Key$(A, heap\text{-}size[A], key)$

Heap-Increase-Key e Max-Heap-Insert

Heap-Increase-Key(A, i, key)

$A[i] \leftarrow key$
while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    do exchange $A[i] \leftrightarrow [\text{Parent}(i)]$
        $i \leftarrow \text{Parent}(i)$

Complexidade ? $O(lg\ n)$

Max-Heap-Insert(A, key)

$heap\text{-}size[A] \leftarrow heap\text{-}size[A] + 1$
$A[heap\text{-}size] \leftarrow -\infty$
Heap-Increase-Key$(A, heap\text{-}size[A], key)$

Complexidade ?

Heap-Increase-Key e Max-Heap-Insert

Heap-Increase-Key(A, i, key)

$A[i] \leftarrow key$
while $i > 1$ and $A[\text{Parent}(i)] < A[i]$
    do exchange $A[i] \leftrightarrow [\text{Parent}(i)]$
        $i \leftarrow \text{Parent}(i)$

Complexidade ? $O(lg\ n)$

Max-Heap-Insert(A, key)

$\textit{heap-size}[A] \leftarrow \textit{heap-size}[A] + 1$
$A[\textit{heap-size}] \leftarrow -\infty$
Heap-Increase-Key$(A, \textit{heap-size}[A], key)$

Complexidade ? $O(lg\ n)$