Fontes principais

1. Cormem T. H.; Leiserson C. E.; Rivest R.: Stein C.. *Introduction to Algorithms*, 3^a edição, MIT Press, 2009
2. Análise de algoritmo - IME/USP (prof. Paulo Feofiloff) `http://www.ime.usp.br/~pf/analise_de_algoritmos`

Algoritmo Heap-Sort

Algoritmo Heap-Sort

Heap: Estrutura de dados útil quando se quer remover o máximo de um conjunto sucessivas vezes.

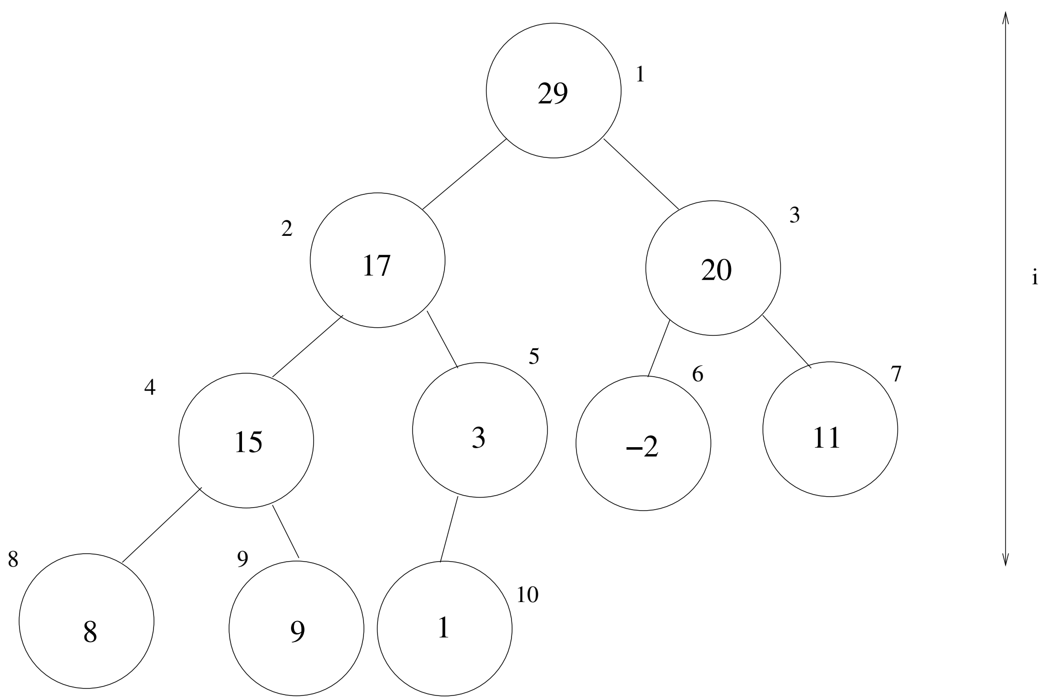
Algoritmo Heap-Sort

Formalmente, um heap é uma árvore binária em que cada nó contém um elemento do conjunto e vale:

1. A árvore é completa até o penúltimo nível
2. No último nível todos os nós estão o mais a esquerda possível
3. Cada nó interno armazena um valor maior que seus filhos

Algoritmo Heap-Sort

Exemplo: $S = \{20, 15, 17, -2, 8, 11, 3, 9, 29, 1\}$



Algoritmo Heap-Sort

Um heap pode ser armazenado em um vetor de tal forma que:

$\triangleright$ O pai do elemento da posição i ($i > 1$) sempre está na posição $\lfloor i/2 \rfloor$
$\triangleright$ E os filhos, nas posições $2i$ e $2i + 1$ (se tais índices forem menores ou iguais a n).

Note que o valor máximo do conjunto representado pelo heap sempre está na raiz.

Algoritmo Heap-Sort

Algoritmos de manipulação do heap consome tempo proporcional à altura do heap. A altura do heap com n elementos: $\Theta(\log n)$

Desce-Heap(A, n, i)

enquanto $(2i \leq n)$
    faça $f = 2i$
        se $(f < n)$ e $(A[f+1] > A[f])$
            então $f = f + 1$
        se $(A[i] < A[f])$
            então $A[i] \leftrightarrow A[f]$
                $i = f$
            senão
                $i = n + 1$

Remoção do máximo heap

Extrai-Maximo-Heap(A, n)

```
max = A[1]
A[1] = A[n]
Desce-Heap(A, n − 1, 1)
retorne max
```

Construção de um Heap

Podemos utilizar o Desce-Heap()

Constroi-Heap(A)

para i de $\lfloor n/2 \rfloor$ **até** 1
  faça Desce-Heap(A, n, i)

Custo do Constroi-Heap(): $\Theta(n)$

Agora podemos definir o algoritmo Heap-Sort

Heap-Sort

Heap-Sort(A, n)

Constroi-Heap(A)
para $i = n$ **até** 2 $\qquad \triangleright\ O(n \log n)$
    faça $A[1] \leftrightarrow A[i]$
        Desce-Heap(A, i-1, 1)

Complexidade do Heap-Sort

Pior caso: Vetor em ordem decrescente $O(n \log n)$

Melhor caso: Vetor em ordem crescente $O(n \log n)$

Caso aleatório: Vetor aleatório $O(n \log n)$

Complexidade de expaço: $\Theta(n)$

Algoritmo Quick-Sort

Algoritmo Quick-Sort

Estratégia de divisão e conquista

Quick-Sort(A, p, r)

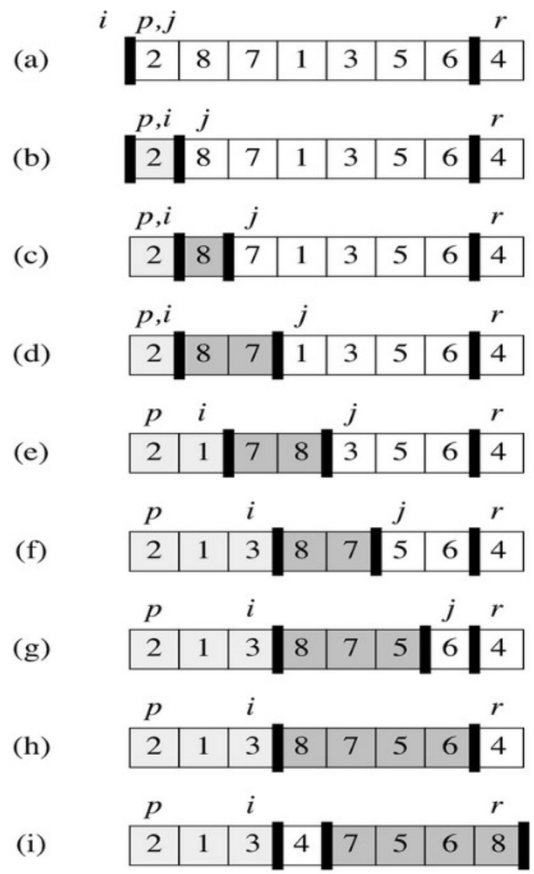
```
se p < r
    então q = particiona(A, p, r)        ▷ p ≤ q < r
          Quick-Sort(A, p, q)
          Quick-Sort(A, q + 1, r)
```

Particionamento

particiona(A, p, r)	custo
$x = A[r]$	$\Theta(1)$
$i = p - 1$	$\Theta(1)$
para $j = p$ **até** $r - 1$ **faça**	$\Theta(n)$
se $A[j] \leq x$ **então**	$\Theta(n)$
$i = i + 1$	$\Theta(n)$
$A[i] \leftrightarrow A[j]$	$\Theta(n)$
$A[i + 1] \leftrightarrow A[r]$	$\Theta(1)$
retorne $i + 1$	$\Theta(1)$
	Total: $\Theta(n)$

Portanto, a complexidade do algoritmo é $\Theta(n)$

Particionamento



Complexidade do Quick-Sort

$$T(n) = \begin{cases} \Theta(1) & \text{, se } n = 0 \\ \max\{T(m) + T(n-m-1)\} + \Theta(n) & \text{, se } n > 0 \text{ e } 0 \leq m < n \end{cases}$$

Resolvendo a recorrência temos, que:

$$T(n) = O(n^2)$$

Dica: utilize indução para verificar que a solução é de fato esta.

Complexidade do Quick-Sort

Pior caso: partições não balanceadas

$$T(n) = T(n-1) + \Theta(n)$$
$$T(n) = \Theta(n^2)$$

Melhor caso: partições balanceadas

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

Caso médio:

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

Quick-Sort aleatorizado

```
particiona-aleatorizado(A, p, r)
    i = aleatorio(p, r)
    A[r] ↔ A[i]
    retorne particiona(A, p, r)
```

```
Quick-Sort-Aleatorizado(A, p, r)
    se p < r
        então q = particiona-aleatorizado(A, p, r)
              Quick-Sort-Aleatorizado(A, p, q)
              Quick-Sort-Aleatorizado(A, q + 1, r)
```

Análise do Quick-Sort aleatorizado

Recorrência para o tempo esperado $E(T(n))$:

$$T(n) = \begin{cases} \Theta(1) & \text{, se } n = 1 \\ \frac{1}{n}\sum_{k=1}^{n-1}(T(k) + T(n-k)) + \Theta(n) & \text{, se } n > 1 \end{cases}$$

Podemos manipular o somatório de forma adequada:

$$T(n) = \frac{2}{n}\sum_{k=1}^{n-1} T(k) + cn$$

Análise do Quick-Sort aleatorizado

Podemos provar que $T(n) = O(n \log n)$. Suponha que $T(n) \leq an \log n + b$, para $n \geq n_0$, com $a, b > 0$ constantes.

$$\begin{aligned} T(n) &= \frac{2}{n}\sum_{k=1}^{n-1} T(k) + cn \\ &\leq \frac{2}{n}\sum_{k=1}^{n-1} (ak \log k + b) + cn \\ &= \frac{2a}{n}\sum_{k=1}^{n-1} k \log k + \frac{2b}{n}(n-1) + cn \end{aligned}$$

Análise do Quick-Sort aleatorizado

Sabemos que $\sum_{k=1}^{n-1} k \log k \leq \frac{1}{2}n^2 \log n - \frac{1}{8}n^2$

$$
\begin{aligned}
T(n) &= \frac{2a}{n}\sum_{k=1}^{n-1} k \log k + \frac{2b}{n}(n-1) + cn \\
&\leq \frac{2a}{n}(\frac{1}{2}n^2 \log n - \frac{1}{8}n^2) + \frac{2b}{n}(n-1) + cn \\
&\leq an \log n - \frac{a}{4}n + 2b + cn \\
&= an \log n + b + (cn + b - \frac{a}{4}n) \\
&\leq an \log n + b
\end{aligned}
$$

Para $\frac{a}{4}n \geq cn + b$

Obrigado