Atividades
Divisão e Conquista
Projeto de indução

Exemplo: cálculo de x^n

- Resolução iterativa com *n* multiplicações: $T(n) = O(n)$
- Resolução mais eficiente, com divisão e conquista:

Exemplo: cálculo de x^n

- Resolução iterativa com *n* multiplicações: T(n) = O(n)
- Resolução mais eficiente, com divisão e conquista:

$$x^n = \begin{cases} 1, \text{ se } n = 0 \\ x, \text{ se } n = 1 \\ x^{\frac{n}{2}} \times x^{\frac{n}{2}}, \text{ se } n \text{ par} > 1 \\ x \times x^{\frac{n-1}{2}} \times x^{\frac{n-1}{2}}, \text{ se } n \text{ ímpar} > 1 \end{cases}$$

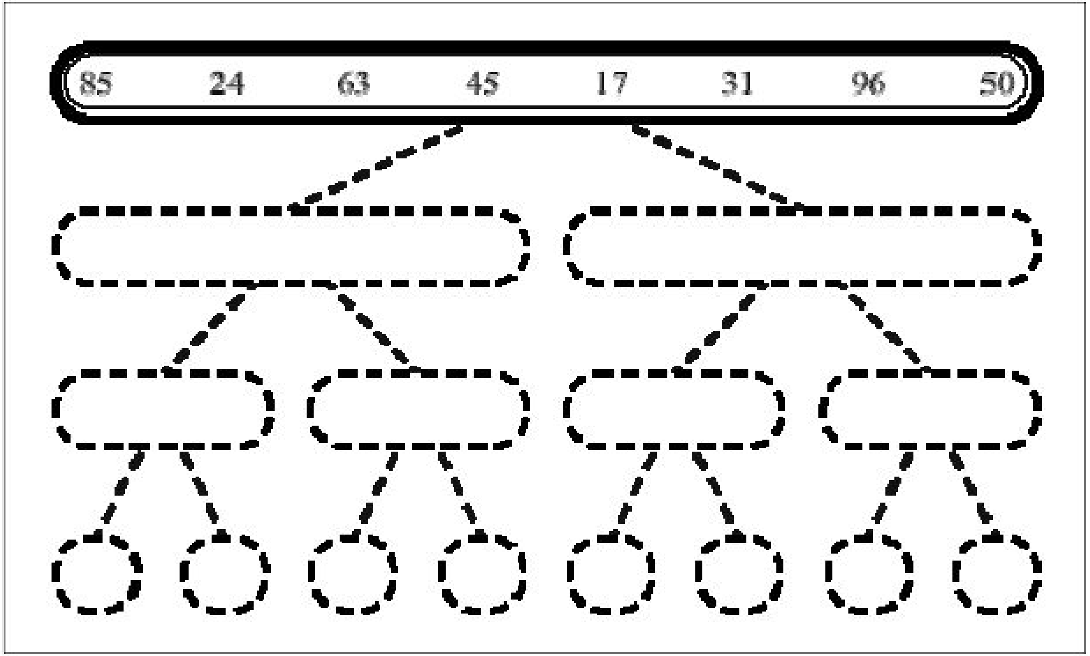
```
double power(double x, int n) {
  if (n == 0) return 1;
  if (n == 1) return x;
  double p = power(x, n / 2);
  if (n % 2 == 0) return p * p;
  else return x * p * p;
}
```

- Divisão em subproblemas iguais, junção em tempo O(1)
- Nº de multiplicações reduzido para aprox. $\log_2 n$
- T(n) = O(log n) .... mas S(n) = O(log n) (espaço)

Exemplo: *Mergesort*

- Seja **S** = {s_1, …, s_n} um conjunto que se pretenda ordenar. Se **S** = ∅ ou **S** = {s}, então nada é necessário!
- Dividir: remover todos os elementos de S e colocá-los em duas subsequências: $\mathbf{S}_1$ e $\mathbf{S}_2$, cada uma com ~n/2 elementos
- Conquistar: consiste em ordenar $\mathbf{S}_1$ e $\mathbf{S}_2$, utilizando mergesort
- Combinar: colocar os elementos de volta em S, unindo as sequências ordenadas $\mathbf{S}_1$ e $\mathbf{S}_2$ numa sequência ordenada única.

Exemplo: *Mergesort*



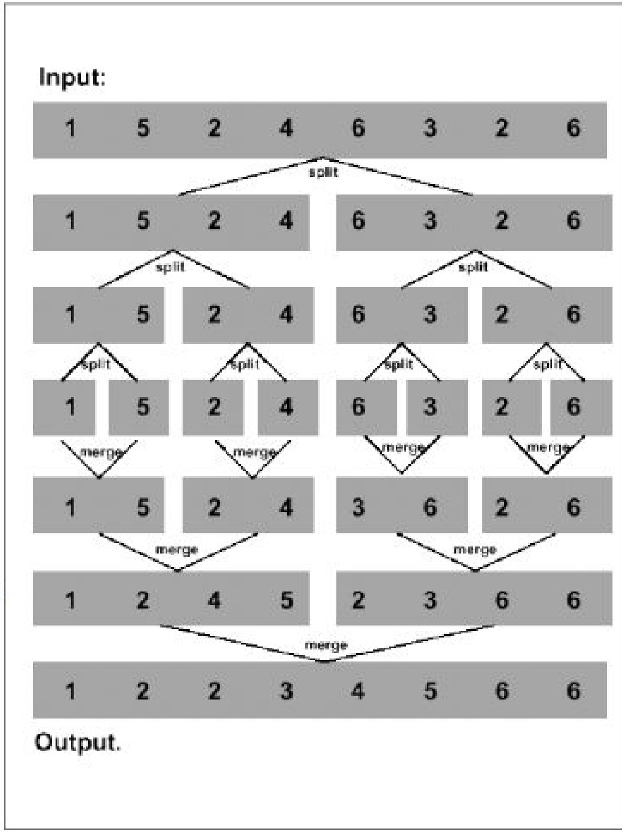
Exemplo: *Mergesort*

```
Merge-Sort(A, p, r)
    if p < r then
        q←(p+r)/2
        Merge-Sort(A, p, q)
        Merge-Sort(A, q+1, r)
        Merge(A, p, q, r)
```

```
Merge(A, p, q, r)
    Take the smallest of the two topmost elements of
sequences A[p..q] and A[q+1..r] and put into the
resulting sequence. Repeat this, until both sequences
are empty. Copy the resulting sequence into A[p..r].
```

Exemplo: *Mergesort*

- P/ ordenar *n* elementos:
  - Se $n = 1$, está feito!
  - Recursivamente ordenar 2 listas de $\lfloor n/2 \rfloor$ elementos
  - Combinar as duas listas ordenadas em tempo $\Theta(n)$
- Estratégia:
  - Dividir o problema em sub-problemas menores
  - Resolver recursivamente
  - Combinar as subsoluções



A estrutura heap

Esta página estuda as propriedades da estrutura de dados *binary heap* inventada por J.W.J. Williams. A estrutura está no coração do algoritmo Heapsort e é muito útil na a construção de filas de prioridades.

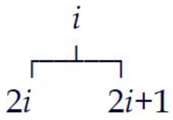
Esta página é inspirada no capítulo 6 de CLRS. Veja também o verbete *Heapsort* na Wikipedia.

Árvore binária armazenada em um vetor

Suponha dado um vetor $A[1..n]$. Por enquanto, os valores armazenados no vetor não nos interessam; só interessam certas relações entre índices. Para todo índice i, diremos que

- $\lfloor i/2 \rfloor$ é o *pai* do índice i (veja definição de *piso*),
- $2i$ é o *filho esquerdo* de i ,
- $2i+1$ é o *filho direito* de i .

É claro que isso deve ser entendido com cautela: o índice 1 não tem pai; um índice i só tem filho esquerdo se $2i \leq n$; e i só tem filho direito se $2i+1 \leq n$.



Com essa história de pais e filhos, o vetor adquire uma estrutura de *árvore binária quase completa* e os elementos do vetor, identificados pelos índices 1 a n, passam a ser chamados *nós*.

A figura abaixo sugere uma maneira de encarar o vetor. Cada caixa é um nó da árvore binária quase completa $A[1..55]$. (O número dentro de cada caixa é i e não $A[i]$.)