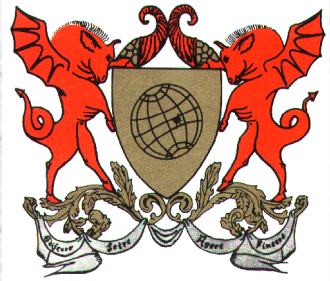
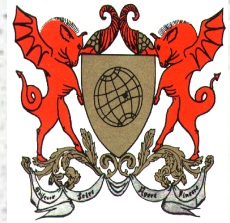


Universidade Federal de Viçosa - Campus Florestal

Algoritmos e Estruturas de Dados I (CCF 211)

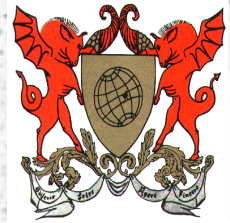
Quicksort

Profa. Thais R. M. Braga Silva
<thais.braga@ufv.br>



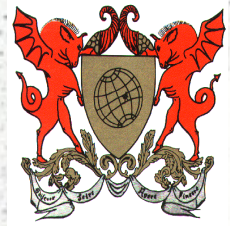
Quicksort

- Proposto por Hoare em 1960 e publicado em 1962.
- É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações.
- Provavelmente é o mais utilizado.
- A ideia básica é dividir o problema de ordenar um conjunto com n itens em dois problemas menores.
  - Os problemas menores são ordenados independentemente.
  - Os resultados são combinados para produzir a solução final.



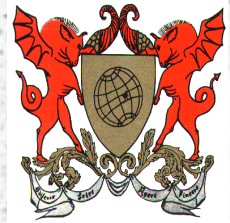
Quicksort

- A parte mais delicada do método é o processo de partição.
- O vetor A [Esq..Dir] é rearranjado por meio da escolha arbitrária de um **pivô** x.
- O vetor A é particionado em duas partes:
  - Parte esquerda: chaves $\leq$ x.
  - Parte direita: chaves $\geq$ x.



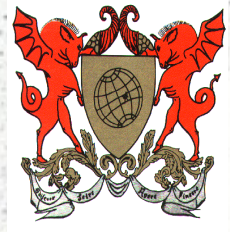
Quicksort - Partição

- Algoritmo para o particionamento:
  1. Escolha arbitrariamente um **pivô** x.
  2. Percorra o vetor a partir da esquerda até que $A[i] \geq x$.
  3. Percorra o vetor a partir da direita até que $A[j] \leq x$.
  4. Troque A[i] com A[j].
  5. Continue este processo até os apontadores i e j se cruzarem.



Quicksort – Após a Partição

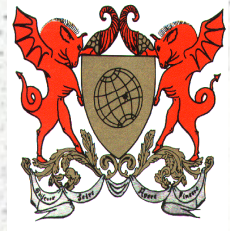
- Ao final, do algoritmo de partição:
  - o vetor A[Esq..Dir] está particionado de tal forma que:
    - Os itens em A[Esq], A[Esq + 1], ..., A[j] são menores ou iguais a *x*;
    - Os itens em A[i], A[i + 1], ..., A[Dir] são maiores ou iguais a *x*.



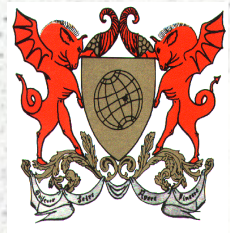
Quicksort - Exemplo

- O pivô x é escolhido como sendo:
  - O elemento central: A[(i + j) / 2].
- Exemplo:

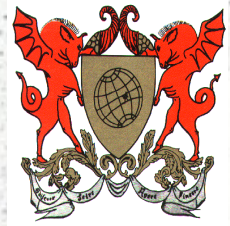
3 6 4 5 1 7 2



Quicksort - Exemplo

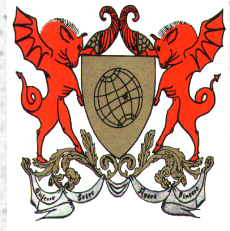


Quicksort - Exemplo



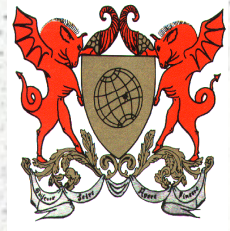
Quicksort - Partição

```
void Particao(int Esq, int Dir,
              int *i, int *j, Item *A)
{
  Item pivo, aux;
  *i = Esq; *j = Dir;
  pivo = A[(*i + *j)/2]; /* obtem o pivo x */
  do
  {
    while (pivo.Chave > A[*i].Chave) (*i)++;
    while (pivo.Chave < A[*j].Chave) (*j)--;
    if (*i <= *j)
    {
      aux = A[*i]; A[*i] = A[*j]; A[*j] = aux;
      (*i)++; (*j)--;
    }
  } while (*i <= *j);
}
```



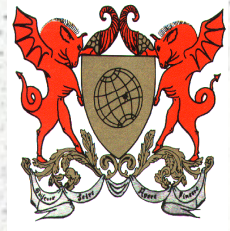
Quicksort

- O anel interno da função Particao é extremamente simples.
- Razão pela qual o algoritmo Quicksort é tão rápido.



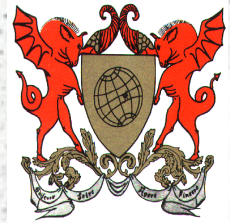
Quicksort - Função

```
/* Entra aqui o procedimento Particao */
void Ordena(int Esq, int Dir, Item *A)
{
  int i,j;
  Particao(Esq, Dir, &i, &j, A);
  if (Esq < j) Ordena(Esq, j, A);
  if (i < Dir) Ordena(i, Dir, A);
}
void QuickSort(Item *A, int n)
{
  Ordena(0, n-1, A);
}
```



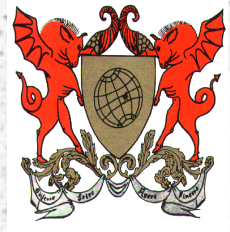
Quicksort

- **Características**
  - Qual o pior caso para o Quicksort?
    - Por que?
    - Qual sua ordem de complexidade?
  - Qual o melhor caso?
  - O algoritmo é estável?



Quicksort

- **Análise**
  - Seja C(n) a função que conta o número de comparações.
  - Pior caso: $C(n) = O(n^2)$
    - O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo sempre o maior ou o menor elemento.
    - Isto faz com que o procedimento Ordena seja chamado recursivamente n vezes, eliminando apenas um item em cada chamada.
    - O pior caso pode ser evitado empregando pequenas modificações no algoritmo.
    - Para isso basta escolher três itens quaisquer do vetor e usar a **mediana dos três** como pivô.



Quicksort

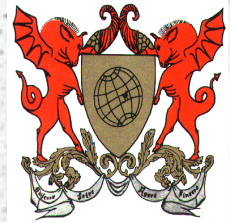
- **Análise**
  - Melhor caso:

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

    - Esta situação ocorre quando cada partição divide o arquivo em duas partes iguais.
  - Caso médio, de acordo com Sedgewick e Flajolet (1996, p. 17):

    $C(n) \approx 1{,}386n \log n - 0{,}846n,$

    - Isso significa que em média o tempo de execução do Quicksort é O(n log n).



Quicksort

- Vantagens:
  - É extremamente eficiente para ordenar arquivos de dados.
  - Necessita de apenas uma pequena pilha como memória auxiliar.
  - Requer cerca de n log n comparações em média para ordenar n itens.
- Desvantagens:
  - Tem um pior caso $O(n^2)$ comparações.
  - Sua implementação é muito delicada e difícil:
    - Um pequeno engano pode levar a efeitos inesperados para algumas entradas de dados.
  - O método não é **estável**.