Projeto e Análise de Algoritmos

Quicksort

Diane Castonguay
diane@inf.ufg.br

Instituto de Informática
Universidade Federal de Goiás

```
QUICKSORT(A, p, r)
1. se p < r
2.      então q ← PARTITION(A, p, r)    //Dividir
3.            QUICKSORT(A, p, q – 1)    //Conquistar
4.            QUICKSORT(A, q + 1, r)    //Conquistar
5.  fim-se
```

```
/*
 * entrada: um vetor A[p..r]
 * saída: um valor q tal que A[p..q-1] ≤ A[r] e A[q..r-1] > A[r]
 * O vetor A é modificado
 */
PARTITION(A, p, r)
1.    chave ← A[r]
2.    i ← p – 1
3.    para j ← p até r – 1 faça
4.          se A[j] ≤ chave
5.          então i ← i + 1
6.                trocar A[i] ↔ A[j]
7.          fim-se
8.    fim-para
9.    trocar A[i+1] ↔ A[r]
10.   retorna i+1
```

Loop invariante do para de PARTITION

No inicio de cada iteração j, temos que

$i < j$

$A[p..i] \leq chave$

$A[i+1..j-1] > chave$

Conclusão do loop invariante

No final do loop para (j = r), temos que

$i < r$

$A[p..i] \leq chave$

$A[i+1..r-1] > chave$

```
RANDOMIZED-PARTITION(A, p, r)
1.    i ← RANDOM(p, r)
2.    trocar A[i] ↔ A[r]
3.    retorna PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)
1. se p < r
2.    então q ← RANDOMIZED-PARTITION(A, p, r)
3.          RANDOMIZED-QUICKSORT(A, p, q – 1)
4.          RANDOMIZED-QUICKSORT(A, q + 1, r)
5.  fim-se
```