Algoritmos e Estrutura de Dados II
(ESTCMP011)
2do PERIODO 2018

UEA
UNIVERSIDADE DO ESTADO DO AMAZONAS

Algoritmos de ordenação:
Quick_sort.

Prof. Luis Cuevas Rodríguez, PhD
E-mail: lcuevasrodriguez@gmail.com / lrodriguez@uea.edu.br
Celular: 9298154648

EST
Escola Superior de Tecnologia da UEA

NUCOMP
NÚCLEO DE COMPUTAÇÃO

Conteúdo

- Especificado o algoritmo de ordenação por:
  - Quicksort
- Analisar seu tempo de execução.

Quicksort

Quicksort (ordenação rápida)

- O arranjo $A[p..r]$ é particionado em dois subarranjos $A[p..q\text{-}1]$ e $A[q+1..r]$
- Os elemento de $A[p..q\text{-}1]$ é menor que $A[q]$, por sua vez, é igual ou menor a cada elemento de $A[q+1..r]$.
- O índice q é calculado como parte do procedimento

- Complexidade
  - No pior caso é $\Theta(n2)$
  - Na prática a eficiência na média é muito boa
  - O tempo de execução esperado $\Theta(n \log n)$

Quicksort

- Conquistar
  - Os dois subarranjos $A[p..q-1]$ e $A[q+1..r]$ são ordenados por chamadas recursivas
- Combinar
  - Os arranjos são ordenados localmente, não há a necessidade de combinação

Quicksort

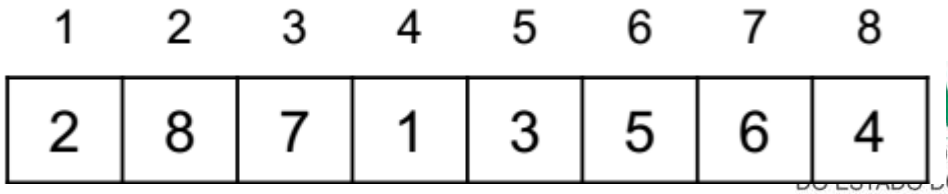
```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```

- Para ordenar um arranjo *A*, a chamada inicial seria `QUICKSORT(A,1,n)`

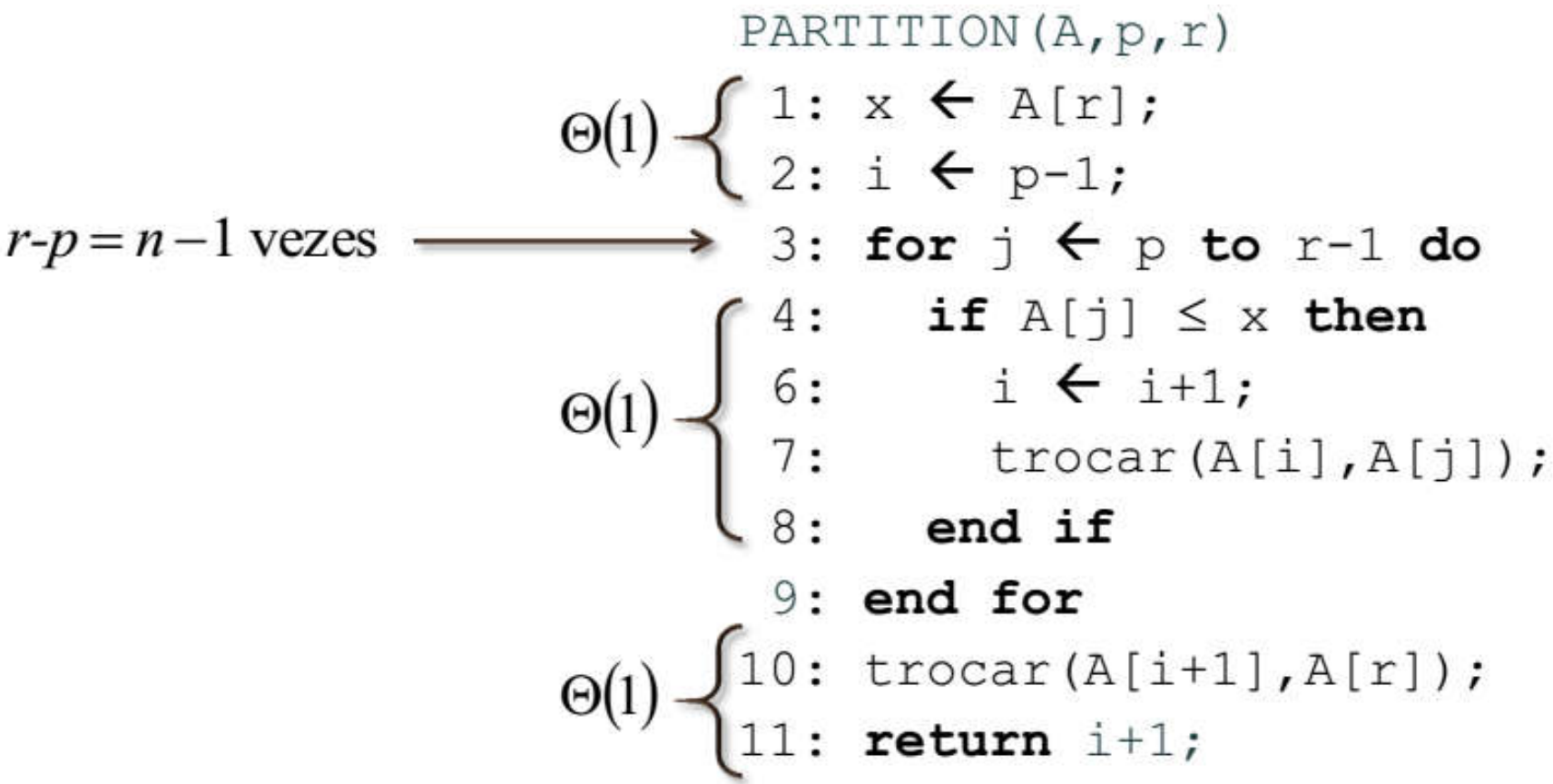
Quicksort

```
PARTITION(A,p,r)
 1: x ← A[r];
 2: i ← p-1;
 3: for j ← p to r-1 do
 4:    if A[j] ≤ x then
 6:       i ← i+1;
 7:       trocar(A[i],A[j]);
 8:    end if
 9: end for
10: trocar(A[i+1],A[r]);
11: return i+1;
```

- O PARTITION sempre seleciona um elemento $x=A[r]$ como um pivô.
- O particionamento do arranjo $A[p..r]$ é feito em torno do pivô x



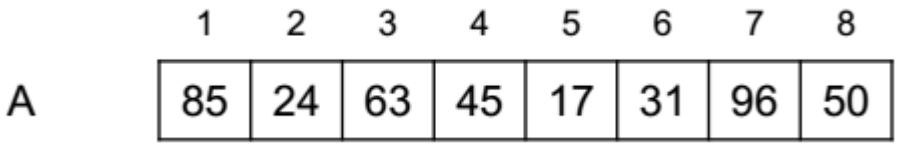
Quicksort



$$T(n) = \Theta(1) + (n-1)\Theta(1) + \Theta(1) = \Theta(n) \quad \Rightarrow \quad T(n) = \Theta(n)$$

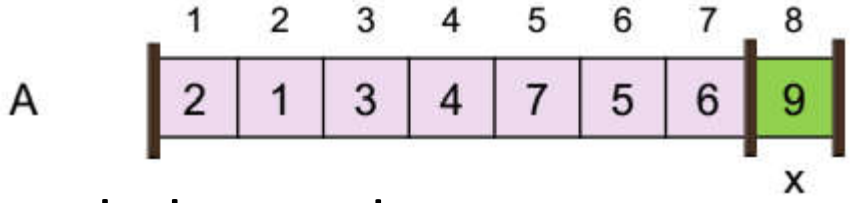
Quicksort

```
QUICKSORT(A,p,r)
1: if p < r then
2:    q ← PARTITION(A,p,r);
3:    QUICKSORT(A,p,q-1);
4:    QUICKSORT(A,q+1,r);
6: end if
```



Quicksort

- O desempenho do Quicksort depende da qualidade do particionamento
  - Pior caso: totalmente desbalanceado



  - Melhor caso: totalmente balanceado
- Particionamento balanceado
  - Custo de mesma ordem que o Mergesort
- Particionamento desbalanceado
  - Custo de mesma ordem que o Insertionsort

Exercícios

1. Ilustre a operação Quicksort sobre os vetores
   a) C=(27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4,8,9,0)
   b) A=(5, 3, 17, 10,84, 19,6,22,9)
   c) B=(5, 13, 2,25,7,17,20,8,4)

Implementação

```
int particao (int vetor[], int inicio, int fim)
{
    int temp;
    int pivot = vetor[fim];
    int i = (inicio - 1);

    for (int j = inicio; j <= fim-1; j++)
    {
        if (vetor[j] <= pivot)
        {
            i++;
            temp = vetor[i];
            vetor[i]=vetor[j];
            vetor[j]=temp;
        }
    }
    temp = vetor[i + 1];
    vetor[i + 1]=vetor[fim];
    vetor[fim]=temp;
    return (i + 1);
}
```

```
void quickSort(int vetor[], int inicio, int fim)
{
    if (inicio < fim)
    {
        int q = particao(vetor, inicio, fim);
        quickSort(vetor, inicio, q - 1);
        quickSort(vetor, q + 1, fim);
    }
}
```

Implementação

```
int main()
{
    int lista[TAM]={10, 80, 30, 90, 40, 50, 70};
    imprime_vetor(lista);
    quickSort(lista,0,6);
    cout << endl;
    imprime_vetor(lista);
    return 0;
}
```