ED
Ordenação (Selection Sort, Bubble Sort, Insertion Sort, MergeSort, QuickSort)

Profa. Célia Taniwaki

Ordenação

- Muitas vezes, é preciso ordenar os dados para poder manipular esses dados de uma forma organizada.
  - Por exemplo:
    - Nomes em ordem alfabética
    - Alunos por ordem do RA ou por Nota
- A busca de um elemento em uma lista ordenada é mais eficiente (Pesquisa binária)

Métodos de Ordenação Simples

- São 3:
  - Selection sort – ordenação por seleção
  - Bubble sort – ordenação por troca
  - Insertion sort – ordenação por inserção
- Características:
  - Fácil implementação

Selection sort

- Método simples de seleção
  - Ordena através de sucessivas seleções do elemento de menor valor (ou de maior valor) em um segmento não ordenado do vetor e seu posicionamento no final de um segmento ordenado
- Características
  - Realiza uma busca sequencial pelo menor valor (ou maior valor) no segmento não ordenado a cada iteração

Selection Sort - Algoritmo

```
SelectionSort (int[] v)
início
   inteiro i, j, min;
   para i de 0 até v.length-2 (inclusive) faça
   início
      min ← i;
      para j de i+1 até v.length-1 (inclusive) faça
         se v[j] < v[min]
         então min ← j;
      troca (v[i], v[min]);
   fim
fim
// onde está troca(v[i],v[min]) significa que os valores
// de v[i] e v[min] devem ser trocados (isso pode ser
// feito com 3 instruções e uma variável auxiliar)
```

Bubble Sort

- Método simples de troca
  - Ordena através de sucessivas trocas entre pares de elementos do vetor
- Características
  - Realiza varreduras no vetor, trocando pares adjacentes (vizinhos) de elementos, sempre que o próximo elemento for menor que o anterior
  - Após uma varredura, o maior elemento está corretamente posicionado no vetor e não precisa mais ser comparado.
- Veja simulação de funcionamento em:
  - http://i.giphy.com/26u6cLtm2nwTvUcgw.gif

Bubble Sort - Algoritmo

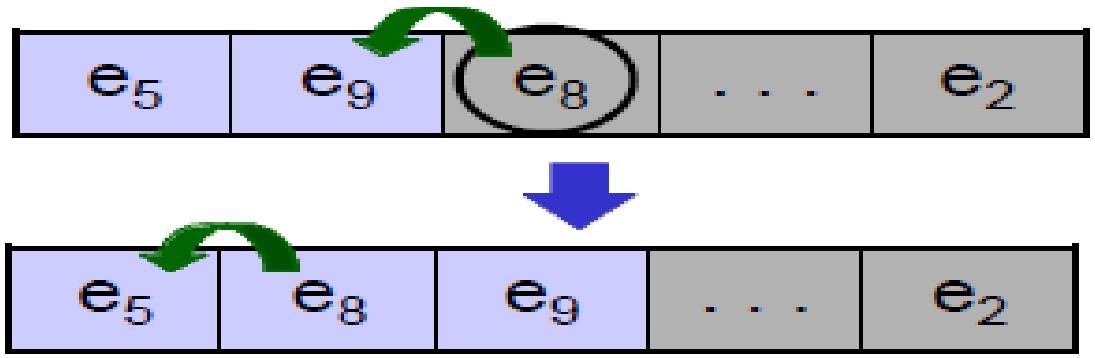
```
BubbleSort (int[] v)
início
  inteiro i, j;
  para i de 0 até v.length-2 (inclusive) faça
    para j de 1 até v.length-1-i (inclusive) faça
        se v[j-1] > v[j]
        então troca (v[j], v[j-1]);
fim
```

Insertion Sort

- Método simples de inserção
- Características
  - Considera 2 segmentos do vetor: ordenado (aumenta a cada varredura) e não ordenado (diminui)
  - Ordena através da inserção de um elemento por vez do segmento não ordenado no segmento ordenado, na sua posição correta
  - Inicialmente, o segmento ordenado contém apenas o primeiro elemento do vetor

Insertion Sort

- Realiza uma busca sequencial no segmento ordenado para inserir corretamente o novo elemento
- Realiza trocas entre elementos adjacentes, até encontrar o local onde deve inserir o novo elemento.



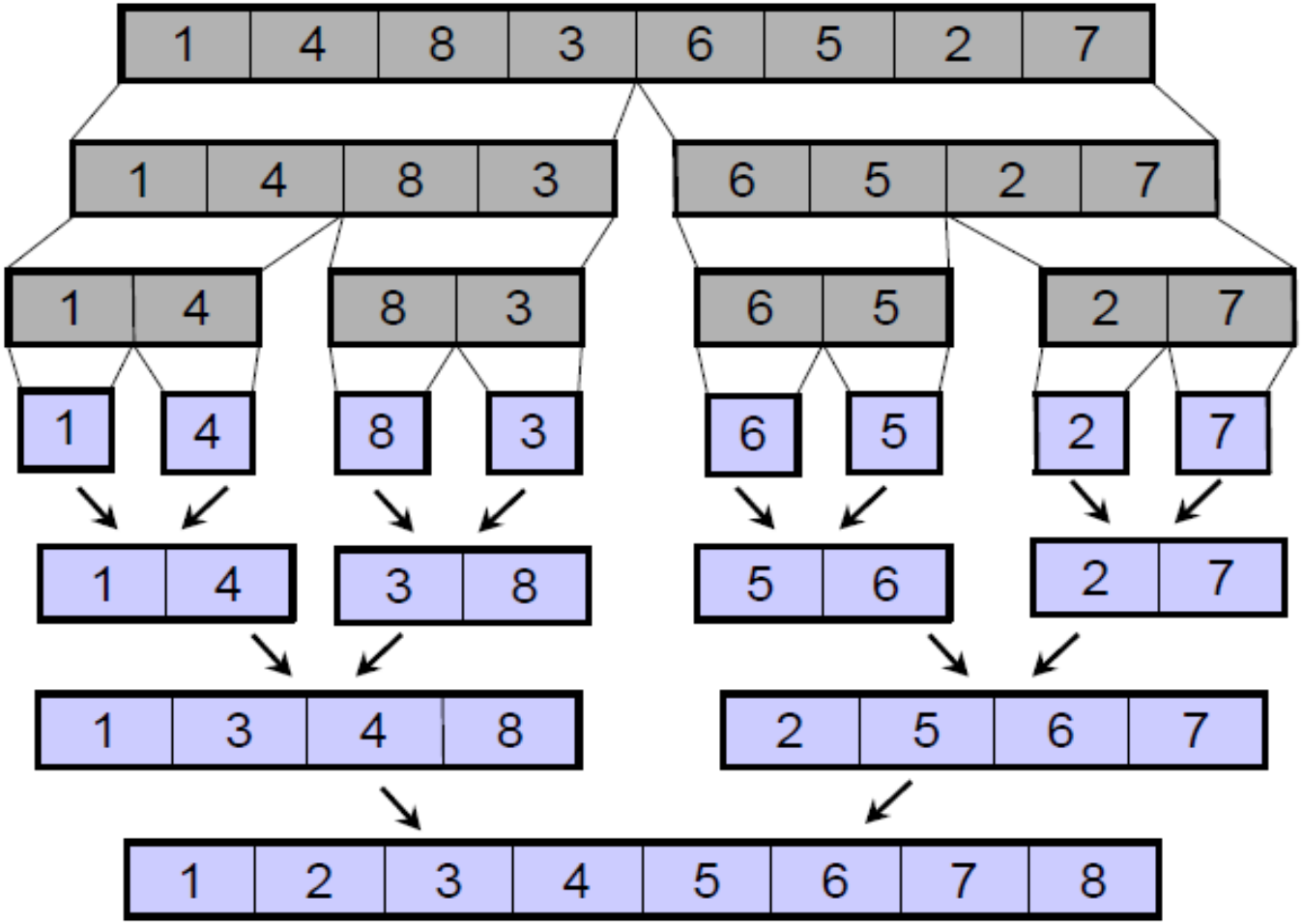
Insertion Sort - Algoritmo

```
InsertionSort (int v[], int n)
/* vetor v [0 ... n-1]  */
início
   inteiro i, j, x;
   para i de 1 até n-1 (inclusive) faça
   início
       x ← v[i];
       j ← i - 1;
       enquanto (j >= 0) e (v[j] > x) faça
       início
          v[j+1] ← v[j];
          j ← j - 1;
       fim
       v[j+1] ← x;
   fim
fim
```

MergeSort

- Método particular de ordenação
  - Baseia-se em intercalações sucessivas de 2 sequências ordenadas em uma única sequência ordenada
- Aplica o método "dividir para conquistar"
  - Divide o vetor de n elementos em 2 segmentos de comprimento n/2
  - Ordena recursivamente cada segmento
  - Intercala os 2 segmentos ordenados para obter o vetor ordenado completo

MergeSort - Exemplo



MergeSort – Algoritmo recursivo

```
Mergesort (int p, int r, int[] v)
/* p = índice inicial do vetor a ser ordenado */
/* r = índice final + 1 do vetor a ser ordenado */
/* vetor v [p ..... r-1]  */

se (p < r-1)
início
    int q ← (p + r) / 2;  /* q = índice do meio */
    Mergesort(p, q, v);    /* ordena 1a metade */
    Mergesort(q, r, v);    /* ordena 2a metade */
    Intercala(p, q, r, v); /* intercala 2 metades */
fim
```

Algoritmo Intercala (MergeSort)

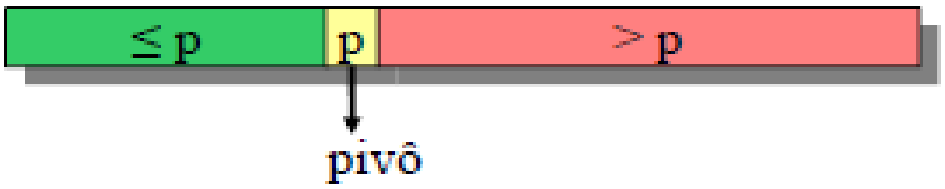
```
Intercala (int p, int q, int r, int[] v)
/*  1a metade do vetor  v[ p ... q-1] */
/*  2a metade do vetor  v[ q ... r-1] */

inteiro i, j, k, w[];
/* deve alocar vetor w de r-p elementos */

i ← p;  j ← q;  k ← 0;
enquanto (i < q) e (j < r) faça
   se (v[i] <= v[j])
   então  w[k++] ← v[i++];
   senão  w[k++] ← v[j++];

enquanto (i < q) faça   w[k++] ← v[i++];
enquanto (j < r) faça   w[k++] ← v[j++];
para i de p até r-1 faça  v[i] ← w[i - p];
```

Quicksort

- Proposto por Hoare em 1960 e publicado em 1962.
- Em geral, o algoritmo é muito mais rápido que os algoritmos elementares.
- Também é um algoritmo de troca: ordena através de sucessivas trocas entre pares de elementos do vetor.
- Aplica o método "dividir para conquistar":
  - A idéia 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: elementos $\leq$ x.
  - Parte direita: elementos $\geq$ 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*.

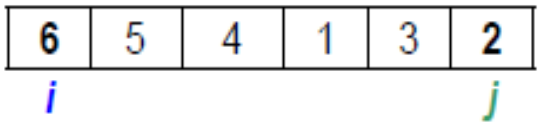
Escolha do pivô

- Particionamento pode ser feito de diferentes formas.
- Principal decisão é escolher o pivô
  - Primeiro elemento do vetor
  - Último elemento do vetor
  - Elemento do meio do vetor
  - Elemento que mais ocorre no vetor
  - Elemento mais próximo da média aritmética dos elementos do vetor

Quicksort – Exemplo (pivô – elem. do meio)

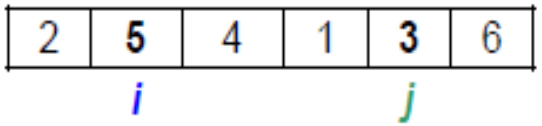
- Seja o vetor abaixo.
- Considerando que 4 seja o pivô, o primeiro passo do Quicksort rearranjaria o vetor da forma abaixo:

1º Passo



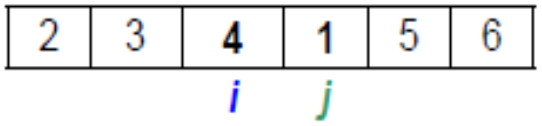
6>4 e 2<4, então, troca 6 com 2

Incrementa i e decrementa j



5>4 e 3<4, então, troca 5 com 3

Incrementa i e decrementa j



4=4 e 1<4, então, troca 4 com 1

Incrementa i e decrementa j

i > j → ***PARTICIONA***

i é diferente de final e ***j*** é diferente de início.

Faz a chamada recursiva para os dois casos.

Pivô
4

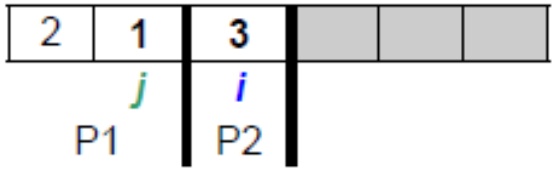
2º Passo

2<3 → incrementa i

Pivô
3

3=3 e 1<3, então, troca 3 com 1

Incrementa i e decrementa j



*i > j → **PARTICIONA***

Chama a recursão apenas para a Partição 1 (P1), pois P2 é de tamanho igual a 1 (i = fim)

3º Passo

2=2 e 1<2, então, troca 2 com 1

Incrementa i e decrementa j

Pivô
2

i > j → j = inicio e i= fim → Finaliza

Não particiona mais

Algoritmo

```
particiona (int[] v, int indInicio, int indFim)
início
   inteiro i, j, pivo;
   i ← indInicio;  j ← indFim;
   pivo ← v[(indInicio+indFim)/2];
   enquanto i <= j faça
    início
      enquanto i < indFim e v[i] < pivo
           i ← i + 1;
      enquanto j > indInicio e v[j] > pivo
           j ← j - 1;
      se i <= j
      entao inicio
                  troca (v[i], v[j]);
               i ← i + 1;
               j ← j - 1;
            fim
    fim
   se indInicio < j então particiona (v, indInicio, j);
   se i < indFim então particiona (v, i, indFim);
fim
```

Quicksort

- Quicksort

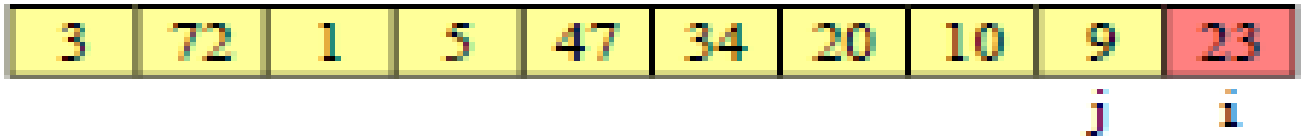
  particiona (v, 0, v.length-1);

- Veja simulação de funcionamento em:
  - http://i.giphy.com/26u6cLtm2nwTvUcgw.gif

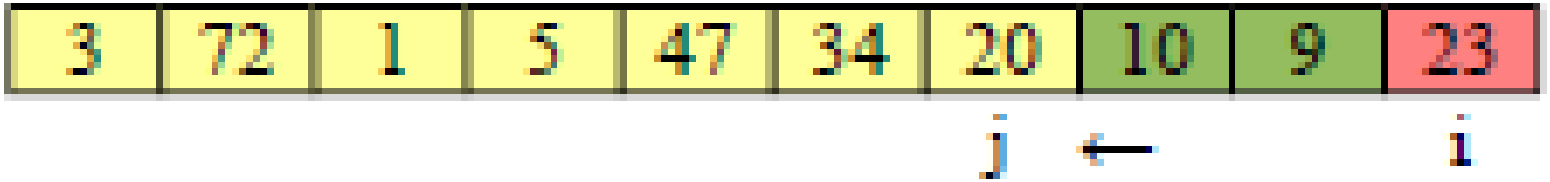
QuickSort (pivô – último elemento)

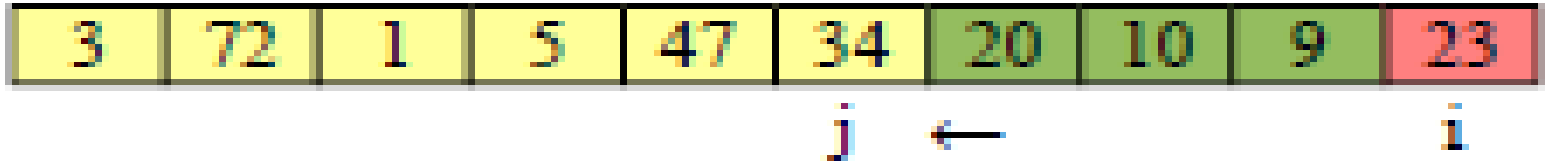
```
particiona (int[] v, int indInicio, int indFim)
início
   inteiro i, j, pivo;
   pivo ← v[indFim];
   i ← indFim;
   para j de indFim – 1 até indInicio faça
    início
      se v[j] > pivo
      então inicio
                i ← i - 1;
               troca (v[i], v[j]);
           fim
    fim
   troca (v[indFim], v[i]);
   se indInicio < i então particiona (v, indInicio, i-1);
   se i < indFim  então particiona (v, i+1, indFim);
fim
```

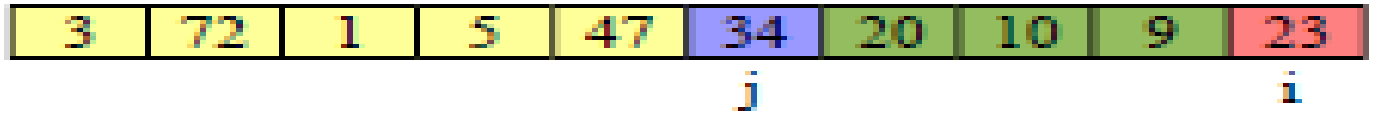
Exemplo

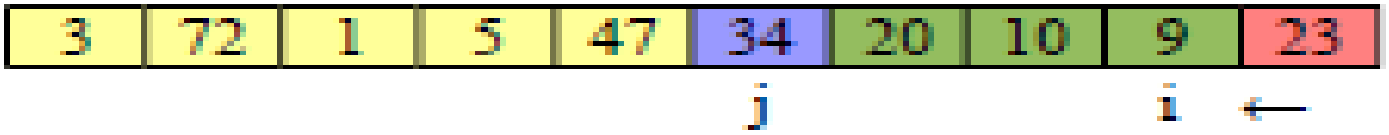



3
72
1
5
47
34
20
10
9
23
j
←
i


3
72
1
5
47
34
20
10
9
23
j
←
i


3
72
1
5
47
34
20
10
9
23
j
←
i


3
72
1
5
47
34
20
10
9
23
j
i


3
72
1
5
47
34
20
10
9
23
j
i

Exercícios

1. Sejam os valores: 5, 3, 4, 2, 8, 1, 6, 7

   Faça a simulação (a mão) dos 5 algoritmos dessa aula para a ordenação dos valores acima, exibindo os dados após cada iteração do algoritmo.
2. Implementar os 5 algoritmos de ordenação dessa aula em Java ou em C#. Acrescentar nos algoritmos a exibição dos elementos do vetor, após cada iteração da ordenação, e observe como é o mecanismo de cada algoritmo.