Algoritmos e Estruturas de Dados

Vetores: Ordenação

Prof. Maiquel de Brito
maiquel.b@ufsc.br

Ordenação de Vetores

Dado um vetor (*v*) com **N** elementos, rearranjar esses elementos por ordem crescente (ou melhor, por ordem não decrescente, porque podem existir valores repetidos)

- Entrada: vetor com elementos a serem ordenados
- Saída: mesmo vetor com elementos na ordem especificada

- Ordenação:
  - Pode ser aplicado a qualquer dado com ordem bem definida
  - Vetores com dados complexos (structs):
    - Chave de ordenação escolhida entre os campos
    - Elemento do vetor contém apenas um ponteiro para os dados
    - Troca da ordem entre 2 elem = troca de ponteiros

Algoritmos de Ordenação

Facilidade de codificação X complexidade algoritmica

Algoritmos:

- **Ordenação por Inserção (InsertionSort)**
- **Ordenação por Seleção (SelectionSort)**
- BubbleSort
- ShellSort
- **MergeSort**
- **QuickSort**
- HeapSort

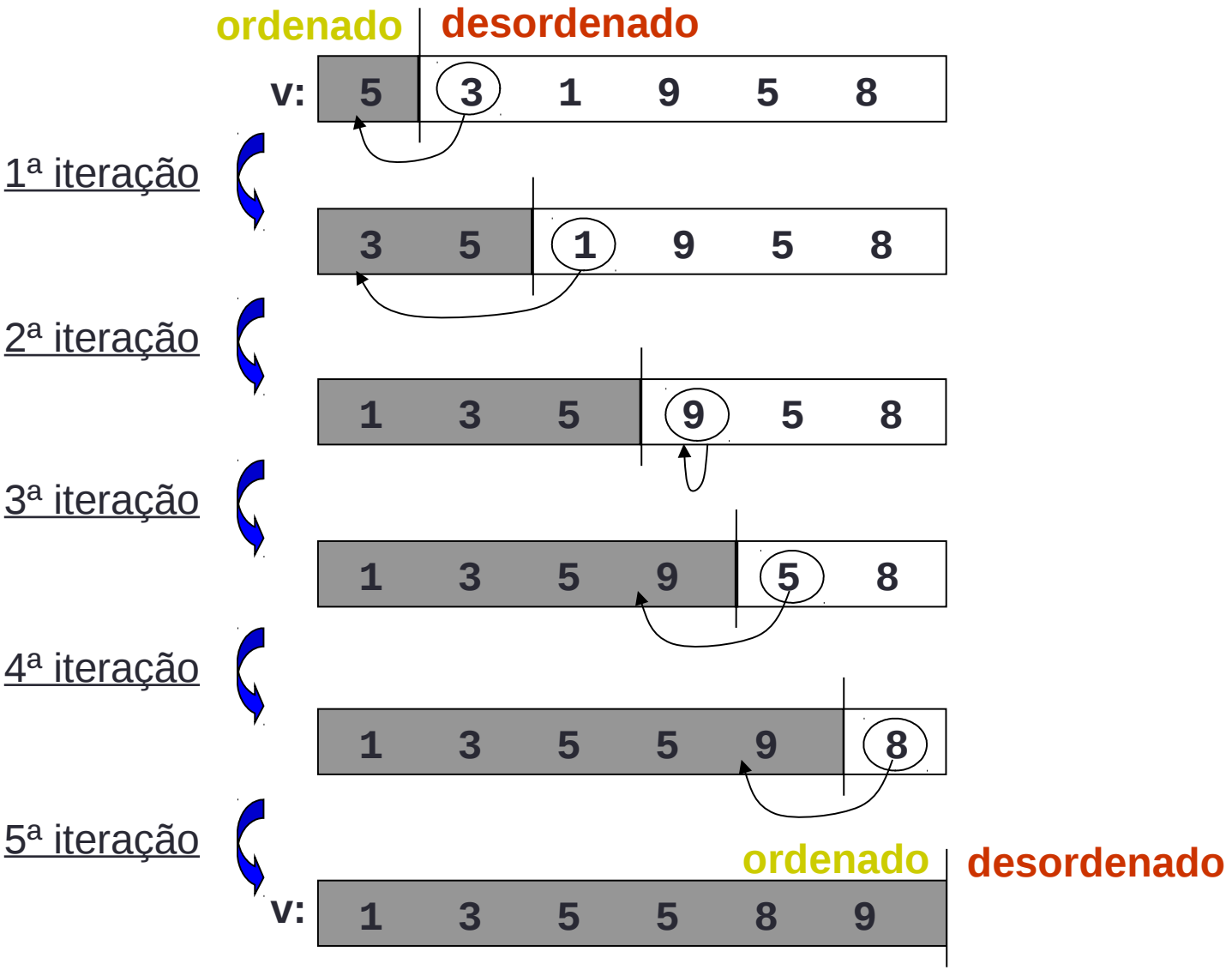
Ordenação por Inserção

Percorre-se um vetor de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados

Algoritmo

- Considera-se o vetor dividido em dois sub-vetores (esquerdo e direito), com o da esquerda ordenado e o da direita desordenado
- Começa-se com um elemento apenas no sub-vetor da esquerda
- Move-se um elemento de cada vez do sub-vetor da direita para o sub-vetor da esquerda, inserindo-o na posição correta de forma a manter o sub-vetor da esquerda ordenado
- Termina-se quando o sub-vetor da direita fica vazio

Ordenação por Inserção



Ordenação por Inserção (implementação em C)

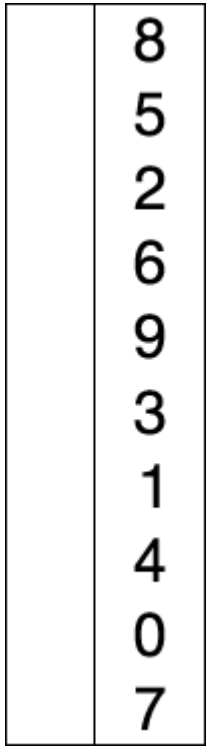
```c
/* Ordena elementos do vetor v de inteiros. */

void insertionsort(int *v, int tamanho){
   int i, j, tmp;
   for (i = 1; i < tamanho; i++){
       tmp = v[i];
       for (j = i; j>0 && tmp<v[j-1]; j--){
          v[j] = v[j-1];
     }
       v[j] = tmp;
   }
}
```

Ordenação por Seleção

Estratégia: seleciona o menor elemento do vetor, depois o segundo menor, depois o terceiro menor, e assim por diante

Em cada etapa F:

- Procura-se (sequencialmente) a posição M com o menor elemento guardado nas posições de F a N;
- Troca-se o valor guardado na posição F com o valor guardado na posição M (excepto se M for igual a F)



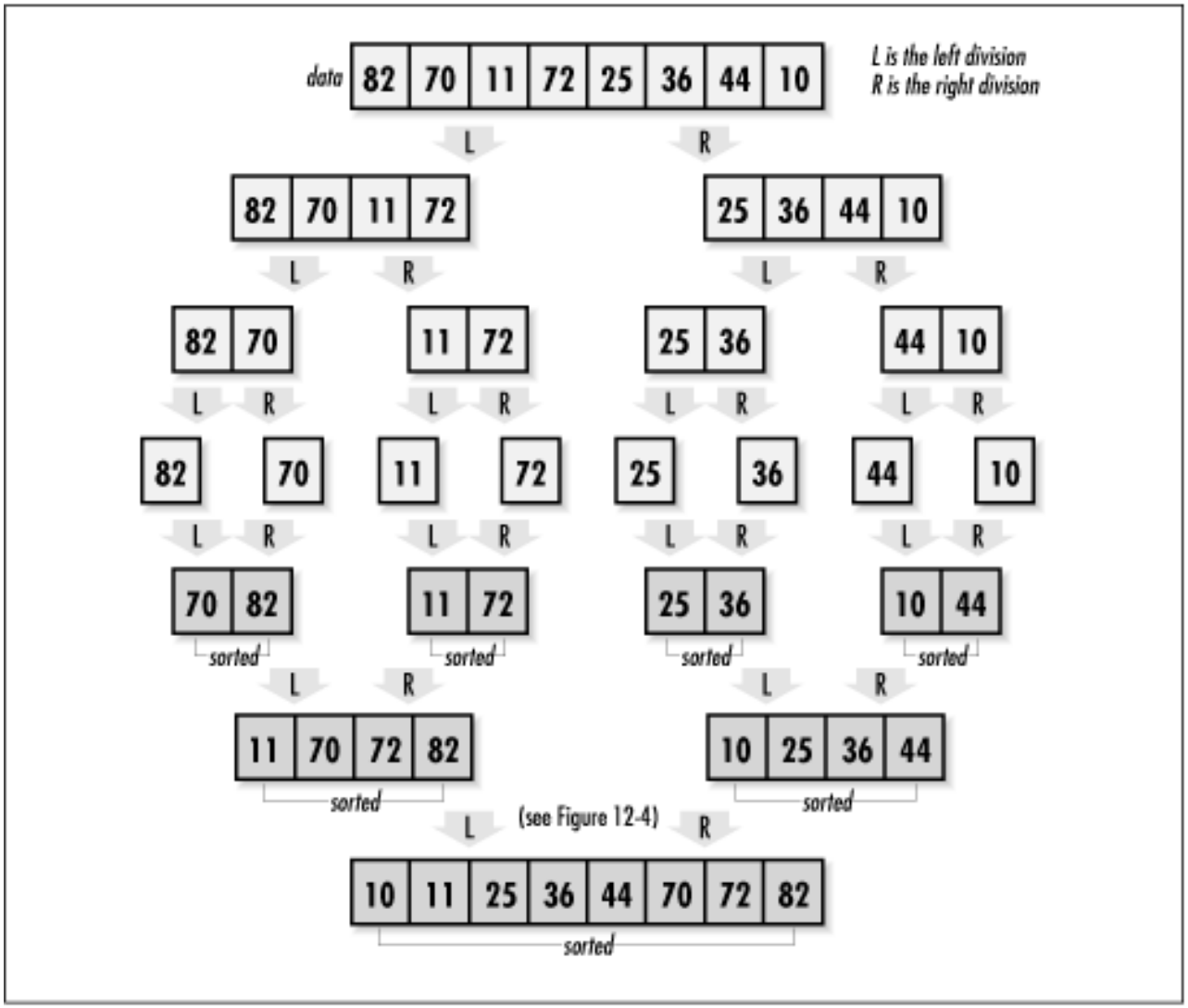
Ordenação por Seleção (vetor na vertical)

Índice	início	**passo 1**	**passo 2**	**passo 3**	**passo 4**	**passo 5**	**passo 6**	**passo 7**
0	7	7	2	2	2	2	2	2
1	21	21	21	7	7	7	7	7
2	10	10	10	10	10	10	10	10
3	15	15	15	15	15	11	11	11
4	2	2	7	21	21	21	13	13
5	13	13	13	13	13	13	21	15
6	11	11	11	11	11	15	15	21

Ordenação por Seleção (implementação em C)

```
void selectionsort(int *v, int n){
    int passo = 0, imin, i, aux;
    while (passo < n – 1) {
        imin = passo;
        i = passo + 1;
      while (i < n){
          if (v[i] < v[imin]){
                imin = i;
            }
          i++;
        }
        if (imin != passo){
            aux = v[passo];
            v[passo] = v[imin];
            v[imin] = aux;
        }
       passo++;
    }
}
```

Merge Sort


data
82 70 11 72 25 36 44 10
L is the left division
R is the right division
L
R
82 70 11 72
25 36 44 10
82 70
11 72
25 36
44 10
82
70
11
72
25
36
44
10
70 82
11 72
25 36
10 44
sorted
11 70 72 82
10 25 36 44
(see Figure 12-4)
10 11 25 36 44 70 72 82

Fonte: http://c2.com/cgi/wiki?SortingAlgorithms

Comparativo

	Worst case	Average case	Best case	Extra space	Stable
BubbleSort	O(n^2)	O(n^2)?	O(n)	O(1)	yes
SelectionSort	O(n^2)	O(n^2)	O(n^2)	O(1)	No (i
InsertionSort	O(n^2)	O(n^2)	O(n)	O(1)	yes
BitonicSort	O(n log^2 n)	O(n log^2 n)?	?	O(1)?	?
ShellSort	O(n^2)	O(n log n)?	O(n)	O(1)	no
QuickSort	O(n^2)	O(n log n)	O(n log n)	O(log n)	no
HeapSort	O(n log n)	O(n log n)	O(n log n)	O(1)	no
SmoothSort	O(n log n)	O(n log n)?	O(n)	O(1)	no
MergeSort	O(n log n)	O(n log n)	O(n log n)	O(n)	yes
TimSort	O(n log n)	O(n log n)?	O(n)	O(n)	yes
CountingSort	O(n+k)	O(n+k)	O(n+k)	O(n+k)	yes
RadixSort	O(n+k)	O(n+k)	O(n+k)	O(n+k)	yes
BucketSort	O(n^2)	O(n+k)	?????	O(n*k) or O(n+k)	?
BogoSort	unbounded	O(n!)	O(n)	O(1)	no
SlowSort	O(n^(log n))	O(n^(log n))	O(n^(log n))	O(1)	yes
QuantumBogoSort	O(1)	O(1)	O(1)	O(0)	no