UFSC-CTC-INE

INE5384 - Estruturas de Dados

Ordenação de Dados (IV)

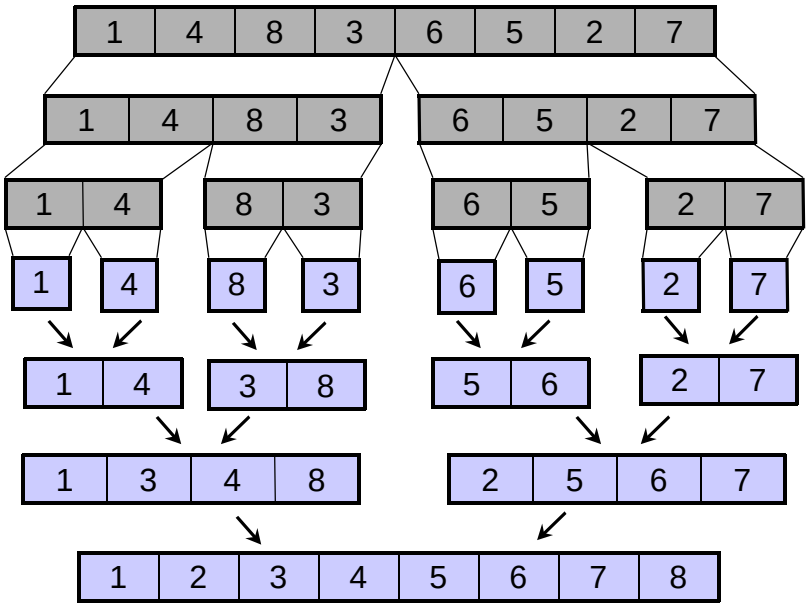
Prof. Ronaldo S. Mello

2002/2

MergeSort

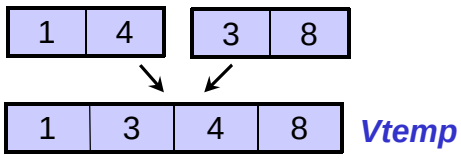
- *MergeSort* é um método particular de ordenação
  - baseia-se em junções sucessivas (*merge*) de 2 seqüências ordenadas em uma única seqüência ordenada
- Aplica um método "dividir para conquistar"
  - divide o vetor em 2 segmentos (sub-vetores) de comprimento $\lfloor n/2 \rfloor$ e $\lceil n/2 \rceil$
  - ordena recursivamente cada sub-vetor (dividindo novamente, quando possível)
  - faz o *merge* dos 2 sub-vetores ordenados para obter o vetor ordenado completo

MergeSort - Exemplo



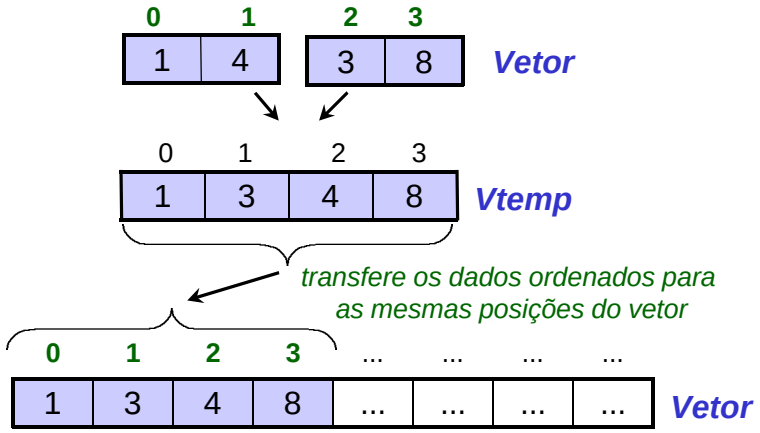
MergeSort - Junção ou *Merge*

- Utiliza um vetor temporário (*Vtemp*) para manter o resultado da ordenação dos 2 sub-vetores
  - *o método **não** é "in-place"*
  - *Vtemp* deve ser um atributo da classe *OrdenadorMergeSort*



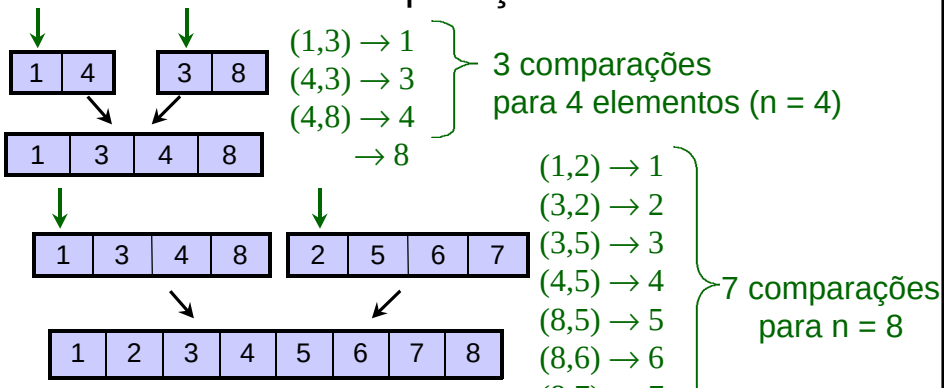
MergeSort - Junção ou *Merge*

- Após a ordenação, o conteúdo de *Vtemp* é transferido para o vetor



MergeSort - Junção ou *Merge*

- Número de comparações?



  – máximo de comparações: n - 1

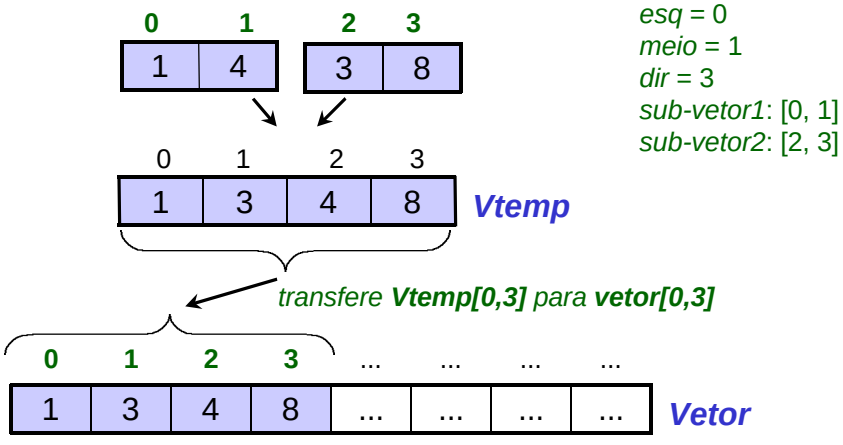
- Complexidade: ***O(n)***

MergeSort - Método *Merge*

- Recebe 3 parâmetros: *esq*, *meio*, *dir*
  - esq, meio e dir são posições do vetor que delimitam 2 sub-vetores contíguos ordenados
  - sub-vetor 1: [esq, meio]
  - sub-vetor 2: [meio+1, dir]
- Ordena os 2 vetores em *Vtemp*
  - o resultado fica no intervalo [esq, dir] de *Vtemp*
    - *Vtemp* deve ter o mesmo comprimento do vetor
- Transfere o intervalo [esq, dir] de *Vtemp* para o intervalo [esq, dir] do vetor

MergeSort - Método *Merge*

- Exemplo: *Merge(0, 1, 3)*

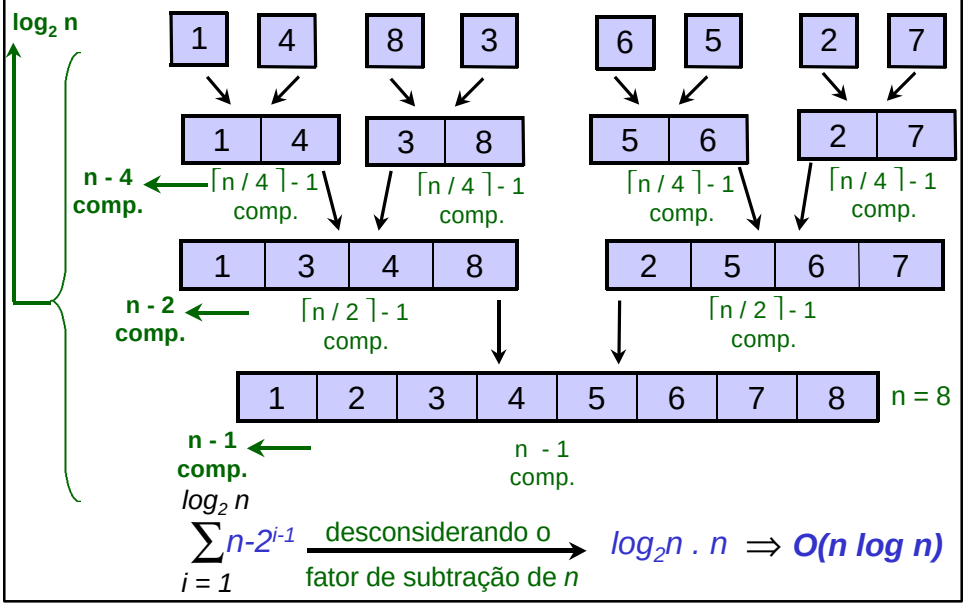


esq = 0
meio = 1
dir = 3
sub-vetor1: [0, 1]
sub-vetor2: [2, 3]

MergeSort - Método *Ordena*

- É um método recursivo
  - recebe como parâmetro os delimitadores do vetor a ser ordenado (esq, dir)
    - no início: *ordena(0, n-1)*
  - se o vetor tiver pelo menos 2 elementos
    - determina o meio do vetor (*meio*)
    - chama recursivamente *ordena* para os 2 sub-vetores: *ordena(esq, meio)* e *ordena(meio+1,dir)*
    - realiza o *merge* dos 2 sub-vetores ordenados

MergeSort - Complexidade



MergeSort

- Simulação de funcionamento

http://math.hws.edu/TMCM/java/xSortLab

MergeSort - Comparação

	QuickSort	*HeapSort*	*MergeSort*
Pior caso	$O(n^2)$	$O(n \log n)$	$O(n \log n)$
Caso médio	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Melhor caso	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$

- Classificando por melhor desempenho médio:
  1º) *MergeSort* (algoritmo mais simples)
  2º) *QuickSort*
  3º) *HeapSort*

Ordenação por Distribuição

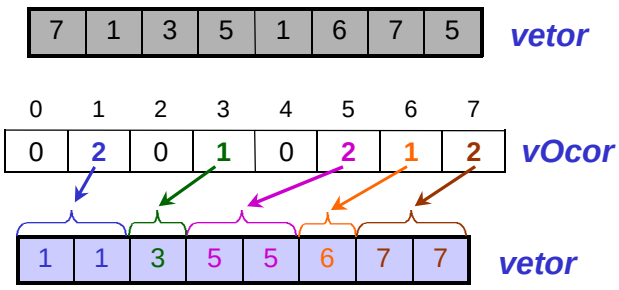
- Não faz comparações entre elementos para realizar a ordenação (!)
- Não pode ser aplicado para a ordenação de qualquer conjunto de dados
  - algumas restrições devem ser atendidas
- Ordenação em tempo linear (*O(n)*) (!)
- Exemplos:
  - *BucketSort*
  - *RadixSort*

BucketSort

- Restrições
  - ordena somente elementos que são números inteiros
  - considera um pequeno conjunto de dados que não ultrapassa um valor máximo pequeno ***m***
    - valor dos elementos encontra-se no intervalo [0, m]
- Exemplo
  - ordenação dos 80 empregados da empresa pelo seu tempo de serviço (em anos)
    - n = 80
    - m = 50

BucketSort - Funcionamento

- Utiliza um vetor auxiliar (*vOcor*) que mantém o número de ocorrências de cada valor de elemento
- Distribui os valores dos elementos ordenadamente no vetor com base nos números de ocorrências em *vOcor*



BucketSort - Implementação

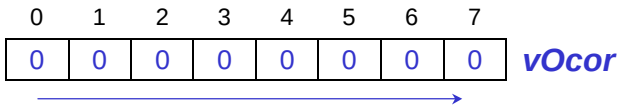
```
Classe OrdenadorBucketSort
SubClasse de Ordenador
início
   m inteiro;
   vOcor inteiro[ ];

      construtor OrdenadorBucketSort (m inteiro);
      início
            this.m ← m;
            vOcor ← NOVO inteiro[m];
      fim;
. . .
fim;
```

BucketSort – Método Ordena

- Inicializa o vetor de ocorrências (*vOcor*) com zero
- Varre o vetor e contabiliza o número de ocorrências de cada elemento em *vOcor*
- Ordena o vetor com base no número de ocorrências em *vOcor*

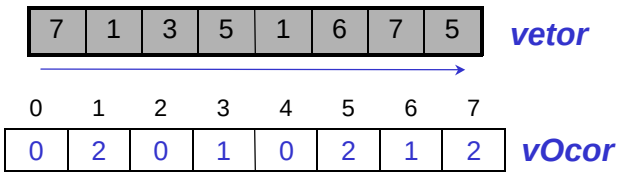
BucketSort – Etapa 1

- Inicializa o vetor de ocorrências (*vOcor*) com zero
  - complexidade: ***O(m)***



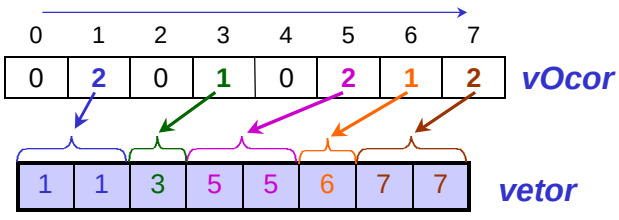
BucketSort – Etapa 2

- Varre o vetor e contabiliza o número de ocorrências de cada elemento em *vOcor*
  - complexidade: ***O(n)***



BucketSort – Etapa 3

- Ordena o vetor com base no número de ocorrências em *vOcor*
  - complexidade: ***O(m + n)***
  - *"+n"* se refere ao *loop* que conta quantas ocorrências existem em cada posição de *vOcor* e insere o elemento no vetor (o total desta contagem é *n*)



BucketSort - Complexidade

- Complexidades envolvidas:
  - *O(m)*, *O(n)* e *O(m+n)*
- Considerando que *m* deve ser pequeno, sua complexidade é assumida como linear no número de dados (*m = O(n)*)
- Complexidade do *BucketSort*: ***O(n)***

Exercícios

- Implementar os seguintes métodos para a classe *OrdenadorMergeSort*:
  - *ordena*(*esq* inteiro, *dir* inteiro)
  - *merge(esq* inteiro*, meio* inteiro*, dir* inteiro*)*
- *Implementar* para a classe *OrdenadorBucketSort*:
  - *ordena*()