DCC
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UFMG
UNIVERSIDADE FEDERAL DE MINAS GERAIS

# Estrutura de Dados

## Ordenação: MergeSort

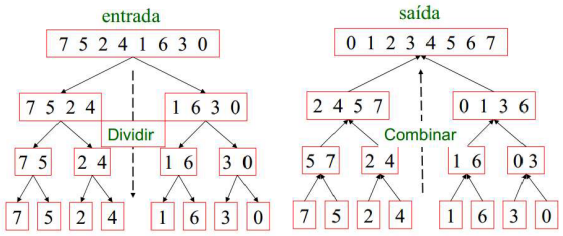
Professores: Luiz Chaimowicz e Raquel Prates

## Introdução

- Método dividir para conquistar baseado em *merging* (ou intercalação)
  - Combinação de dois vetores ordenados em um vetor maior que também esteja ordenado
- Quicksort *x* Mergesort
  - Quicksort:
    - divide o vetor em vetores independentes
    - indexação da posição do pivô + duas chamadas recursivas
  - Mergesort:
    - une dois vetores para criar um único
    - duas chamadas recursivas para dividir + procedimento para unir vetores



## Merge Sort – Visão Geral




Ideia: Dividir → Ordena os pares → Junta



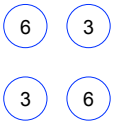
## Merge Sort - Pseudocódigo

```
MergeSort(e,d)
{
  if (e < d )
  {
     meio = (e+d)/2;
     MergeSort(e, meio);
     MergeSort(meio+1, d);
     Merge(e, meio, d);
  }
}
```



## Como juntar?

- Quando acabo de dividir o vetor, chego na situação em que cada vetor tem apenas 1 elemento, então começo a juntar
- Como juntar 2 elementos?

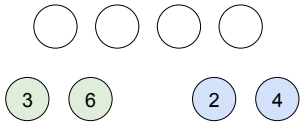



```
if(values[j] < values[j-1])
  swap(values, j-1, j);
```



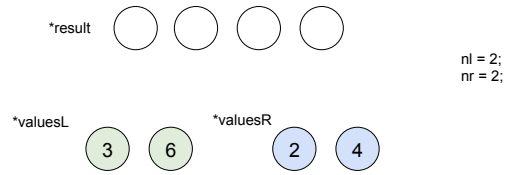
## Merge

- Se eu tiver 2 vetores de 2 elementos ordenados, como criar um **novo** vetor de 4 elementos ainda ordenado?



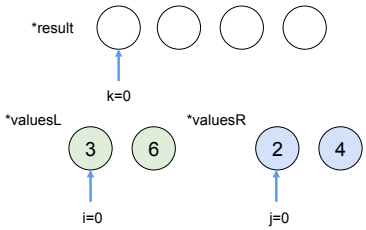
# Merge

```
int *merge(int *valuesL, int *valuesR, int nl  int nr) {
  int *result = (int *) malloc((nl+nr) * sizeof(int));
  //...
  return result
}
```



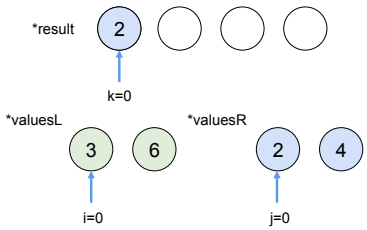



# Merge



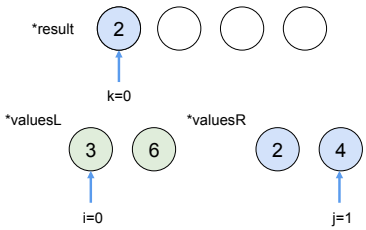



# Merge



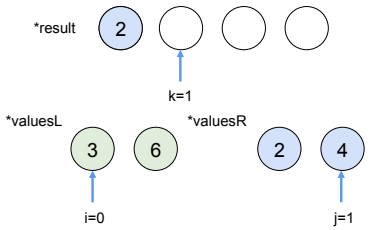



# Merge



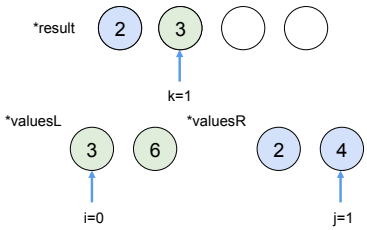



# Merge

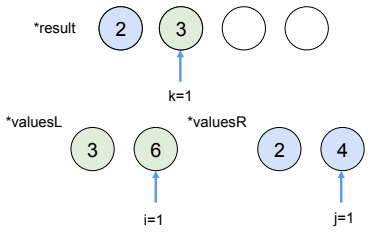





# Merge

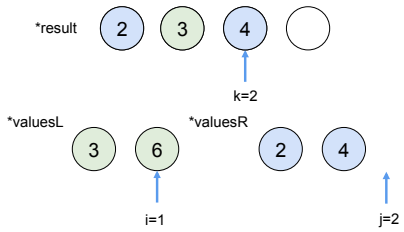


# Merge



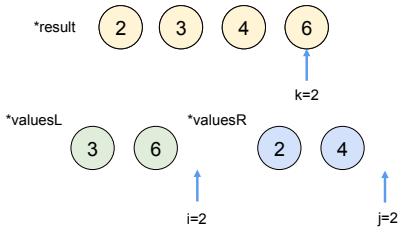



# Merge






# Merge





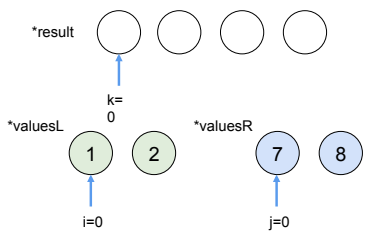

# Merge - Código

- Copia o menor de cada lado
- Até não ter mais de onde copiar de 1 dos lados
- Podemos ficar com elementos restantes

```
while (i < nl && j < nr) {
  if (valuesL[i] < valuesR[j]) {
     result[k] = valuesL[i];
     i++;
  } else {
     result[k] = valuesR[j];
     j++;
  }
  k++;
}
```

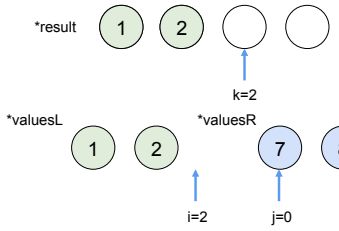


# Merge – E se sobrar elementos de um lado?



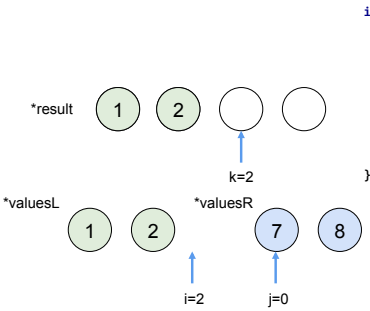



# Merge – E se sobrar elementos de um lado?




```
if (k < nl + nr) {
  for(; i < nl; i++) {
    result[k] = valuesL[i];
    k++;
  }
  for(; j < nr; j++) {
    result[k] = valuesR[j];
    k++;
  }
}
```

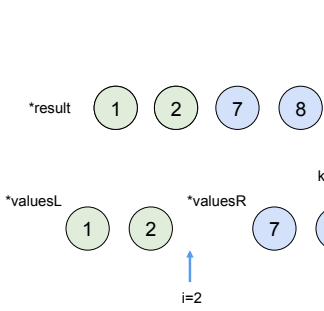
## Merge – E se sobrar elementos de um lado?




```
if (k < nl + nr) {
  for(; i < nl; i++) {
    result[k] = valuesL[i];
    k++;
  }
  for(; j < nr; j++) {
    result[k] = valuesR[j];
    k++;
  }
}
```



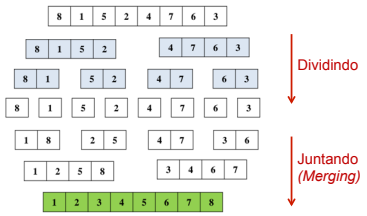
## Merge – E se sobrar elementos de um lado?




```
if (k < nl + nr) {
  for(; i < nl; i++) {
    result[k] = valuesL[i];
    k++;
  }
  for(; j < nr; j++) {
    result[k] = valuesR[j];
    k++;
  }
}
```



## MergeSort






## Reescrevendo a função Merge

- Dois vetores a e b ordenados para um vetor c
- Escolhe para c, o menor de dos elementos que ainda não foram escolhidos dos vetores a e b.

```
merge(Item c[], Item a[], int aTam, Item b[], int bTam ) {
   int i, j, k;
   for (i = 0, j = 0, k = 0; k < aTam+bTam; k++) {
     if (i = = aTam) { c[k] = b[j]; j++; continue; }
     if (j = = bTam) { c[k] = a[i]; i++; continue; }
     if (a[i].chave < b[j].chave) {
       c[k] = a[i];
       i++
     } else {
       c[k] = b[j];
       j++;
     }
   }
}
```



## Merge

- **Problema**
  - Há dois testes no laço interno.
  - Dois vetores separados são passados (a e b)
- **Solução?**
  - Para evitá-los, copia um dos vetores em ordem reversa e o percorre da direita para esquerda.
  - Passa vetor único, indicando o índice do último elemento do vetor da esquerda (variável m).



## Merge Revisto

```
Item aux[maxN];

merge(Item a[], int e, int m, int d){
  int i, j, k;
  /* copia a e b (reverso) para vetor auxiliar */
  for (i = 0; i <= m; i++)
    aux[i] = a[i];
  for (j = m+1; j <= d; j++)
    aux[d-j+m+1] = a[j];
  i = e; j = d;
  for (k = e; k <= d; k++)
    if (aux[i].chave <= aux[j].chave)
       a[k] = aux[i++];
    else
       a[k] = aux[j--];
}
```

## MergeSort

```
void mergesort(Item a[], int e, int d) {
  int m = (d+e)/2;
  if (e < d) {
     mergesort(a, e, m);
     mergesort(a, m+1, d);
     merge(a, e, m, d);
  }
}
```



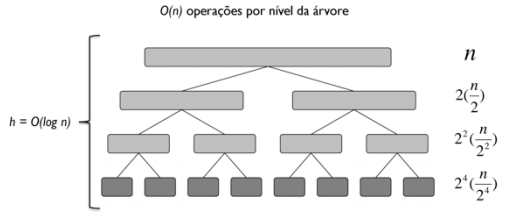
## MergeSort – Análise de Complexidade

- **MergeSort**
  - Para n = 1, T(1) = 0
  - Para n > 1,
    - Uma vez para n/2 elementos
    - Uma vez para n/2 elementos

    } **2 vezes**
- **Operação de Merge**
  - Custo linear: O(n)
  - Dica para análise: contar número de movimentações por iteração e não de comparações por elemento
- Logo: T(n) = 2T(n/2) + n = *O(n log n)*



## Análise de Complexidade






## Com o Teorema Mestre

- Formato da Equação de Recorrência:

$$T(n) = aT(n/b) + f(n)$$

onde $a \geq 1, b > 1$ e $f(n)$ positiva

- Para: T(n) = 2T(n/2) + n

Qual caso se aplica?

1. $T(n) = \Theta(n^{\log_b a})$, se $f(n) = O(n^{\log_b a - \epsilon})$ para alguma constante $\epsilon > 0$,
2. $T(n) = \Theta(n^{\log_b a} \log n)$, se $f(n) = \Theta(n^{\log_b a})$,
3. $T(n) = \Theta(f(n))$, se $f(n) = \Omega(n^{\log_b a + \epsilon})$ para alguma constante $\epsilon > 0$, e se $af(n/b) \leq cf(n)$ para alguma constante $c < 1$ e todo $n$ a partir de um valor suficientemente grande.



## Com o Teorema Mestre

- Formato da Equação de Recorrência:

$$T(n) = aT(n/b) + f(n)$$

onde $a \geq 1, b > 1$ e $f(n)$ positiva

- Para: T(n) = 2T(n/2) + n

Temos: $a = 2$ ,$b = 2$ , $f(n) = n$ e $n^{\log_b a} = n^{\log_2 2} = n$

Caso 2:

$T(n) = \Theta(n^{\log_b a} \log n)$, se $f(n) = \Theta(n^{\log_b a})$

O caso 2 se aplica porque, $f(n) = \Theta(n^{\log_b a}) = \Theta(n)$, *assim,* $T(n) = \Theta(n \log n)$



## MergeSort - Análise

- O algoritmo é **estável**?
  - Sim, pois se os elementos forem iguais eles não são trocados de ordem
- O algoritmo é **adaptável**?
  - Não, ele ordena vetor com n elementos e tempo proporcional a n log n, não importa a entrada.

## Considerações Finais

- Vantagens
  - Deve ser considerado quando alto custo de pior caso não pode ser tolerável.
- Desvantagens
  - Requer espaço extra proporcional a n.
- Comumente adaptado para ordenação em memória secundária



## Exercício

- Execute o mergesort no vetor abaixo indicando a ordem em que as ações serão feitas de acordo com as chamadas recursivas

| 65 | 77 | 51 | 25 | 03 | 84 | 48 | 21 | 05 |
|---|---|---|---|---|---|---|---|---|