Merge Sort

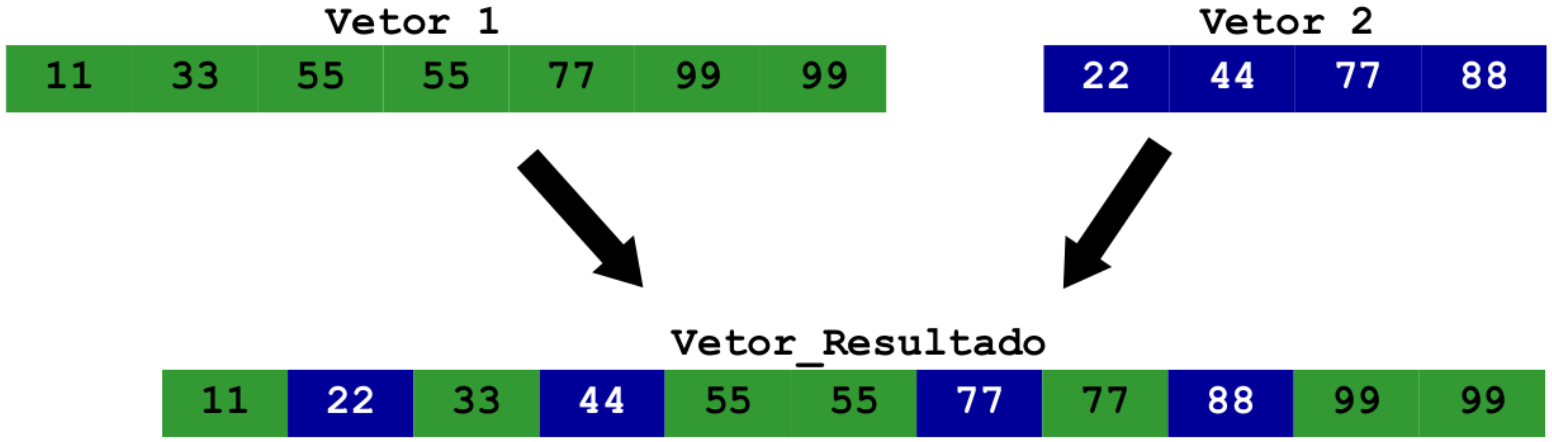
Ciência da Computação
Laboratório de Ordenação e Pesquisa
Prof. M.Sc. Elias Gonçalves

Ideia

- Para entender a ideia do Merge Sort, antes precisamos entender a **intercalação**:
  - Intercalação é o processo através do qual diversos arquivos sequenciais classificados por um mesmo critério são mesclados gerando um único arquivo sequencial.
  - Considera-se cada arquivo como uma pilha, e o registro em memória seu topo.
  - Em cada iteração do algoritmo e leitura dos registros, o topo da pilha com menor chave é gravado, e substituído pelo seu sucessor.

Ideia

➔ Outra forma de fazer **intercalação** é mesclar dois vetores ordenados de modo a obter somente **um** vetor ordenado, com os elementos dos **dois** vetores usados na mesclagem.

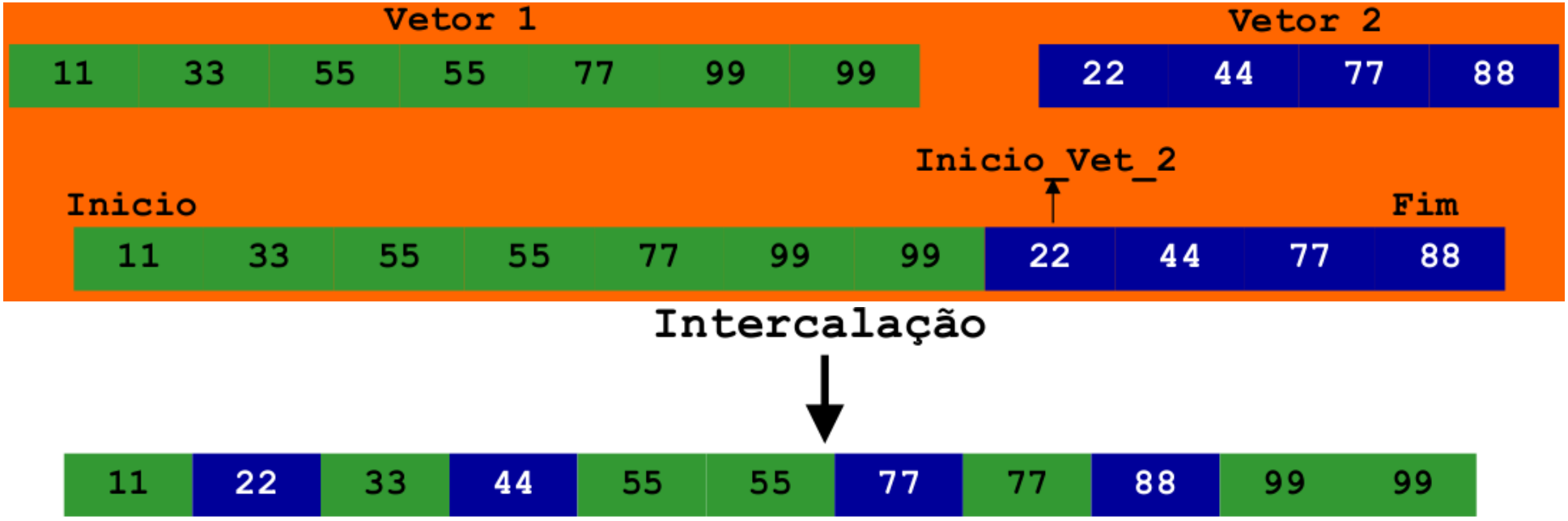


Fonte: Prof. Jairo Francisco

Ideia

➔ Para que aconteça a intercalação inicial, os elementos dos dois vetores devem ser copiados para apenas um vetor.

➔ Para execução do algoritmo de intercalação, o índice de **início** do segundo vetor e o **tamanho** do novo vetor devem ser encontrados.



Fonte: Prof. Jairo Francisco

Ideia

➔ Merge Sort é um algoritmo de **divisão e conquista**.

➔ Ele divide um vetor de entrada em duas partes.

➔ Usa a ideia de intercalação para ordenação dos dados.

Ideia

➔ A ideia central é unir dois vetores (**A** e **B**) já ordenados e criar um terceiro vetor (**C**) que contenha os elementos de **A** e **B** já ordenados na ordem correta.

➔ O foco inicial é no processo de junção dos vetores e posteriormente trabalha-se o processo de ordenação.

Dinâmica

→ Considere que temos dois vetores já ordenados (crescente) **A** e **B** que não precisam ser do mesmo tamanho onde **A** possui 4 elementos e **B** possui 6 elementos. Eles serão unidos para a criação de um vetor **C** com 10 elementos ao final do processo de união.

→ Olharemos para os valores na posição 0 dos vetores **A** e **B** como o topo de uma pilha. Assim podemos comparar cada elemento do dois topos. O menor deles é inserido no vetor **C**.

A

23	47	81	95
0	1	2	3

B

7	14	39	55	62	74
0	1	2	3	4	5

C

7	14	23	39	47	55	62	74	81	95
0	1	2	3	4	5	6	7	8	9

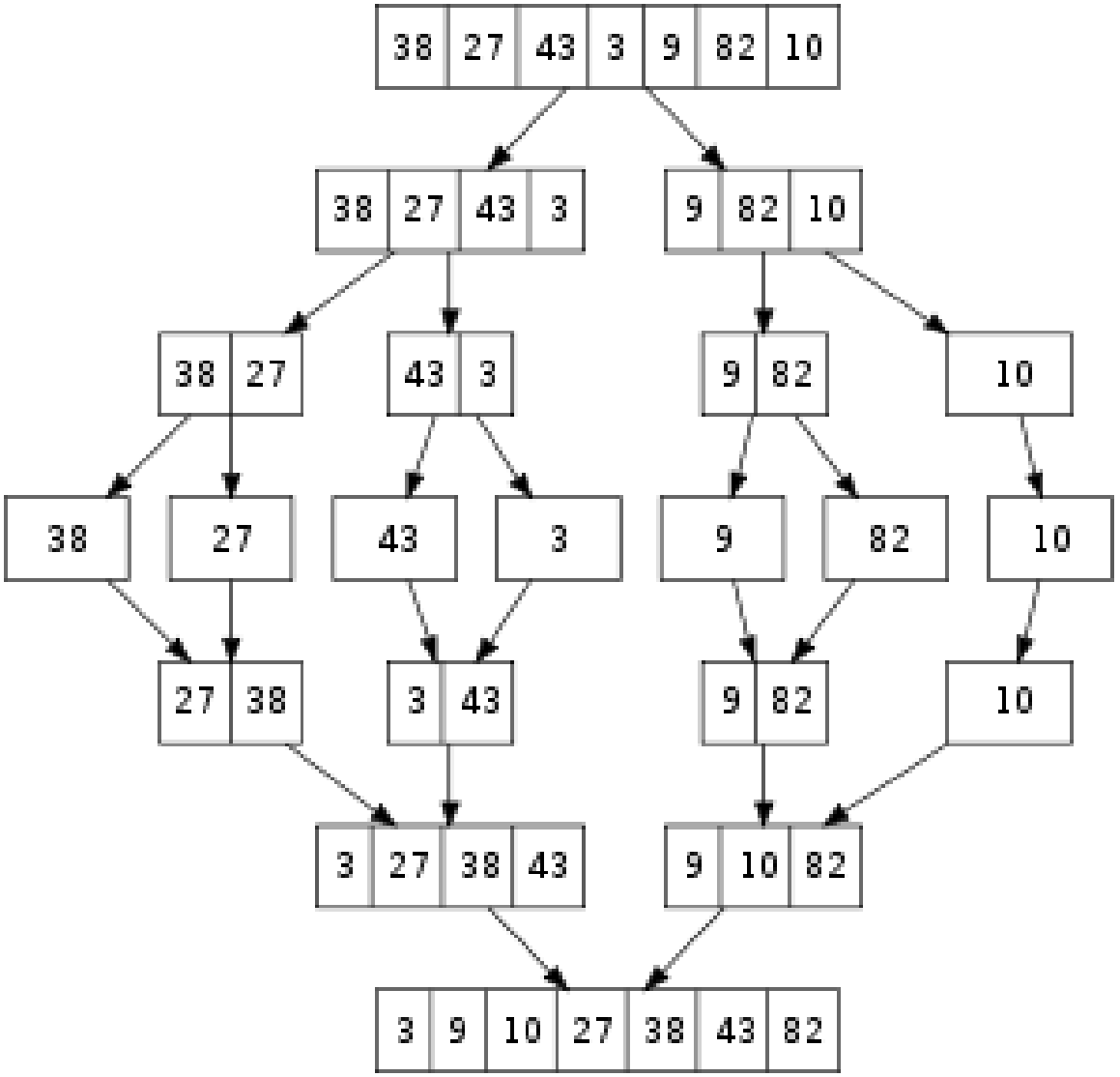
23 < 7? Não. Insere o 7
23 < 14? Não. Insere o 14
23 < 39? Sim. Insere o 23
47 < 39? Não. Insere o 39
47 < 55? Sim. Insere o 47
81 < 55? Não. Insere o 55
81 < 62? Não. Insere o 62
81 < 74? Não. Insere o 74
Como o vetor B terminou e o vetor A já está ordenado, insere o 81 e o 95.

Ideia

➔ **Resumindo**: O Merge Sort divide o vetor de entrada ao meio, ordena cada metade e depois junta estas duas metades novamente formando o vetor original, porém ordenado.

➔ Mas como fazer essa **divisão** e **ordenação** para que as metades possam ser **unidas**?

Recursividade



O Algoritmo

- O algoritmo divide a sequência original em pares de dados, classificando e intercalando os elementos das partições;
- Três passos são fundamentais para a execução do Merge Sort:
  - **Dividir:** Dividir os dados em subsequências pequenas;
  - **Conquistar:** Classificar as duas metades recursivamente aplicando o Merge Sort;
  - **Combinar:** Juntar as duas metades em um único conjunto já classificado.

Código 1

- **Dividir para conquistar:**
  - Executado de forma recursiva;
  - Ordena a primeira metade;
  - Ordena a segunda metade;

```
void merge_sort(int v[], int inicio, int fim) {
    if(inicio < fim){
        int meio= inicio+(fim-inicio)/2;
        merge_sort(v, inicio, meio);
        merge_sort(v, meio+1, fim);
        combinar(v, inicio, meio, fim);
    }
}
```

Código 1 (Continua)

```c
void combinar(int v[], int inicio, int meio, int fim) {
    int i, j, k;
    int tam_inicio = meio - inicio + 1;
    int tam_fim = fim - meio;

    /* Cria vetores auxiliares */
    int aux_inicio[tam_inicio], aux_fim[tam_fim];

    /* Copia dados para os vetores auxiliares */
    for (i = 0; i < tam_inicio; i++)
        aux_inicio[i] = v[inicio + i];

    for (j = 0; j < tam_fim; j++)
        aux_fim[j] = v[meio + 1 + j];

    /* Junta os vetores auxiliares no vetor principal v[inicio...fim]*/
    i = 0;          // Posição inicial do sub vetor inicio
    j = 0;          // Posição inicial do sub vetor fim
    k = inicio;     // Posição inicial do vetor combinado
```

Código 1 (Continuação)

```
    while(i < tam_inicio && j < tam_fim) {
        if (aux_inicio[i] <= aux_fim[j]) {
            v[k] = aux_inicio[i];
            i++;
        }
        else{
            v[k] = aux_fim[j];
            j++;
        }
        k++;
    }

    /* Se ainda houver elementos em aux_inicio, copia eles de volta */
    while (i < tam_inicio) {
        v[k] = aux_inicio[i];
        i++;
        k++;
    }

    /* Se ainda houver elementos em aux_fim, copia eles de volta */
    while (j < tam_fim) {
        v[k] = aux_fim[j];
        j++;
        k++;
    }
}
```

Código 2

- **Dividir para conquistar:**
  - Executado de forma recursiva;
  - Ordena a primeira metade;
  - Ordena a segunda metade;

```
void merge_sort(int v[], int inicio, int fim) {
    if(inicio < fim){
        int meio= inicio+(fim-inicio)/2;
        merge_sort(v, inicio, meio);
        merge_sort(v, meio+1, fim);
        intercalar(v, inicio, meio, fim);
    }
}
```

Código 2 (Continua)

```
void intercalar(int *v, int inicio, int meio, int fim){
    int *aux, pos_inicio, pos_meio, pos_aux;
    pos_inicio = inicio;
    pos_meio = meio+1;
    pos_aux = 0;

    aux = malloc(fim * sizeof(v));

    while( (pos_inicio < meio+1) || (pos_meio < fim+1) ){
        if(pos_inicio == meio+1){
            aux[pos_aux] = v[pos_meio];
            pos_meio++;
            pos_aux++;
        }
        else if(pos_meio == fim+1){
            aux[pos_aux] = v[pos_inicio];
            pos_inicio++;
            pos_aux++;
        }
```

Código 2 (Continuação)

```
        else if(v[pos_inicio] <= v[pos_meio]){
            aux[pos_aux] = v[pos_inicio];
            pos_inicio++;
            pos_aux++;
        }
        else{
            aux[pos_aux] = v[pos_meio];
            pos_meio++;
            pos_aux++;
        }
    }

    for(pos_inicio = inicio; pos_inicio <= fim; pos_inicio++)
        v[pos_inicio] = aux[ (pos_inicio-inicio) ];

    free(aux);

}
```

Estabilidade

- O algoritmo é considerado estável, pois não há a possibilidade de elementos iguais mudar de posição no processo de ordenação.
- A fase de divisão do algoritmo não altera a posição de nenhuma chave.
- Ordenação é feita pelo algoritmo de intercalação:
  - A intercalação é feita verificando, sequencialmente, os elementos de acordo com sua posição no vetor.
  - Dessa forma, elementos com mesma chave não terão a sua posição relativa alterada.

Atividades

- Agora que você já conhece um pouco sobre o Merge Sort, faça sua implementação nas atividades já distribuídas.
- Analise o resultado obtido com a implementação do Merge Sort.

Biblioteca Virtual

CELES, Waldemar; CERQUEIRA, Renato; RANGEL, José Lucas. **Introdução a Estruturas de Dados com Técnicas de Programação em C** (Capítulo 11);

DROZDEK, Adam. **Estrutura de dados e algoritmos em c++** (Capítulo 9);

MARKENZON, Lilian; SZWARCFITER, Jorge Luiz. **Estruturas de Dados e seus Algoritmos** (Capítulos: 7, 11 e 12);

PINTO, Rafael Albuquerque. **Estrutura de Dados** (Páginas 155 a 177).