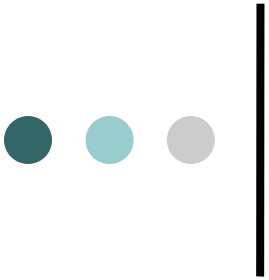


Capítulo 7

Processamento Co-sequencial e Ordenação

Parte 2

Programa

- Introdução
- Operações básicas sobre arquivos
- Armazenagem secundária
- Conceitos básicos sobre estrutura de arquivo
- Organizando arquivos para desempenho
- Indexação
- Processamento co-seqüencial e ordenação
- B-Tree e outras organizações em árvores
- B+Tree e acesso seqüencial indexado
- Hashing
- Hashing estendido

Objetivos

- Descrever uma atividade de processamento, frequentemente utilizada, chamada de processamento co-sequencial.
- Apresentar um modelo genérico para implementar todas as variações de processamento co-sequencial.
- Ilustrar o uso do modelo para resolver diferentes problemas de processamento co-sequencial.
- Apresentar o algoritmo “K-way merge” (intercalação múltipla).
- Apresentar o heapsort como uma abordagem para sorting em RAM
- Mostrar como merging (intercalação) fornece a base para ordenação de arquivos grandes.

Cosequencial Múltiplo

- Quando a entrada é composta por mais de dois arquivos, temos a seguinte situação:
  - Interseção (matching):

    Faça o processamento utilizando somente dois arquivos de entrada, gerando o de saída, repetindo o processo com os arquivos de saída até gerar um único arquivo.
  - Intercalação (merging):

    Para fazer a intercalação de *n* arquivos, precisamos modificar o algoritmo, inserindo uma função que dado *n* entradas, retorna o menor valor dentre os *n* elementos.

Intercalação Múltipla

Algoritmo K-way Merge: intercalar K listas ordenadas para criar uma lista ordenada de saída.

A parte mais cara deste processo são os testes:

- para verificar em quais arquivos a menor chave ocorreu.
  - para saber quais arquivos devem ser lidos a seguir.

A condição de parada

- Pode-se criar uma variável contador
  - que inicia como 0 e é incrementada cada vez que um arquivo chega ao fim.
- A condição de parada se torna verdadeira se ocorrer (EOF) e contador atingir o número de arquivos.

Ou seja, quando o último arquivo sendo lido chegou ao fim.

Intercalação Múltipla

Algortimo K-way Merge:

- Vamos adaptar o 2-way merge visto anteriormente.
  - Ao invés de **arq1** e **arq2** temos um array de arquivos: **arq[1], arq[2], ..., arq[k].**
  - Ao invés de **chave1** e **chave2** temos um array de chaves: **chave[1], chave[2], ..., chave[k].**

Intercalação Múltipla

```
Program Intercalação;
    Const narq = 20 ;
    type Tchave = ... ;
          Treg = record chave: Tchave, ... end;
    var reg: array [1..narq] of Treg;
        chave: array [1..narq] of Tchave;
        arq: array [1..narq] of file of Treg;
        cfimArq: integer ;      existe_Registro: boolean;

procedure Leia_lista (arq; var reg; var chave) {
        read (arq, reg);
         if ( eof (arq)) then
        {    reg.chave = valor_maximo;
             cfimArq = cfimArq + 1;
             if  cfimArq = narq then existe_registro = false
        }
         else if reg.chave < chave then
         {     print "arquivo" + arq + " fora de ordem"; pare };
         chave = reg.chave
};
```

Arrays de registros, chaves e arquivos.

Chegou ao final do último arquivo. Todos foram lidos!

Reconhecimento de erro.

Intercalação Múltipla

- Procedimento de inicialização:

```
{
    for i = 1 to narq do
        abra arq[i];
    crie arqSaída;
    for i = 1 to narq do
        chave[i] =valor_minimo;
    existe_registro = true;
    cfimArq = 0;
    for i =1 to narq
        Leia_lista (arq[i], reg[i], chave[i]);
```

Abre os arquivos/listas.

As chaves para todos os arquivos são inicializadas com o valor_minimo. Valor_minimo é uma constante. É o menor valor que a chave poderia assumir.

Lê o primeiro registro/item de cada arquivo/lista.

Intercalação Múltipla

- Procedimento de sincronização dos arquivos:

```
        while existe_registro do
        {
              k = 1;
              for i = 2 to narq do
                    if reg[i].chave <reg[k].chave then k = i;
              escreve (arqsaída, reg[k]);
              chavex = chave[k] ;
              for i =1 to narq do
                    if chavex = reg[i].chave then
                    Leia (arq[i], reg[i], chave[i]);
        };
}.
```

Seleciona a menor chave dentre os k registros lidos.

Grava o registro de menor chave no arquivo de saída.

Lê o próximo registro de todos os arquivos que tem chave igual a chave gravado no arq saída.

Intercalação Múltipla

A sequência de if's, dada abaixo, é necessária pois mais de um arquivo pode ter chave igual a menor chave encontrada.

```
chavex = chave[k] ;
for i =1 to narq do
        if chavex = reg[i].chave then
        Leia_lista (arq[i], reg[i], chave[i]);
```

Se não houverem itens repetidos nas listas, então apenas o arquivo contendo a menor chave deverá ser lido.

```
Leia_lista (arq[k], reg[k], chave[k]);
```

Intercalação Múltipla

Algoritmo de comparação para achar o arquivo que contém a menor chave

```
k = 1;
for i = 2 to narq do
        if reg[i].chave <reg[k].chave then k = i;
escreve (arqsaída, reg[k]);
```

PROBLEMA: Quanto maior for o valor de **narq**, mais caro será encontrar o menor.

ALTERNATIVA: Substituir o loop de comparações por uma árvore de seleção.

Intercalação Múltipla

- Árvore de seleção:
  - Cada nó em um nível mais alto representa o “vencedor” da comparação entre dois nós descendentes.
  - O valor mínimo está sempre no nó raíz da árvore.
  - Se cada chave tem uma referência associada à lista de onde veio, é apenas uma questão de obter a chave que está na raiz, e ler o próximo elemento da lista associada, e então rodar o algoritmo novamente.
  - É uma árvore binária, portanto:
    - O número de comparações para estabelecer um *novo* “vencedor” é relativo a sua profundidade, ou seja $\log_2 K$ (e não uma função linear de K).

Intercalação Múltipla:

Árvore de Seleção (Exemplo)

A1: ... 61, 52, 41

A2: ... 19, 14, 12

A3: ... 23, 17, 08

A4: ... 29, 25, 14

A5: ... 55, 48, 39

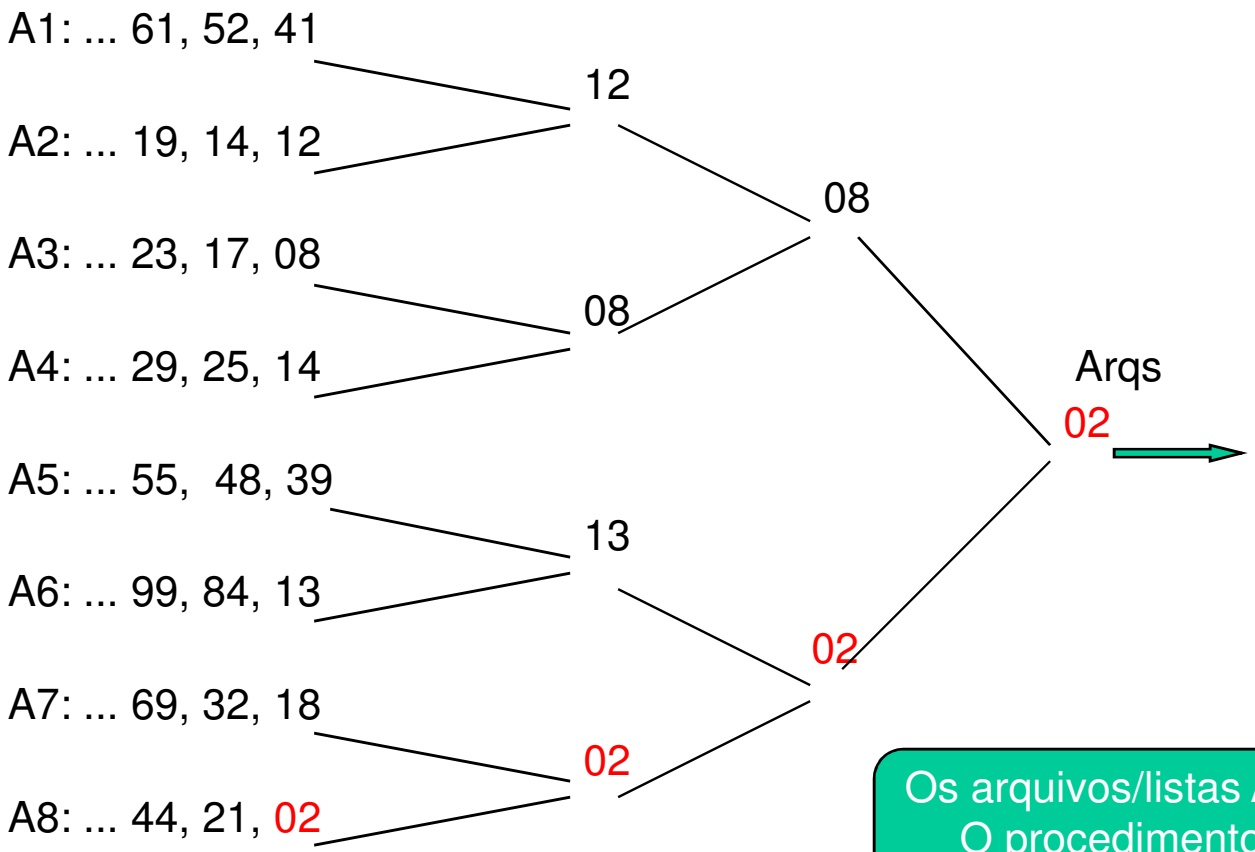
A6: ... 99, 84, 13

A7: ... 69, 32, 18

A8: ... 44, 21, 02

Intercalação Múltipla:

Árvore de Seleção

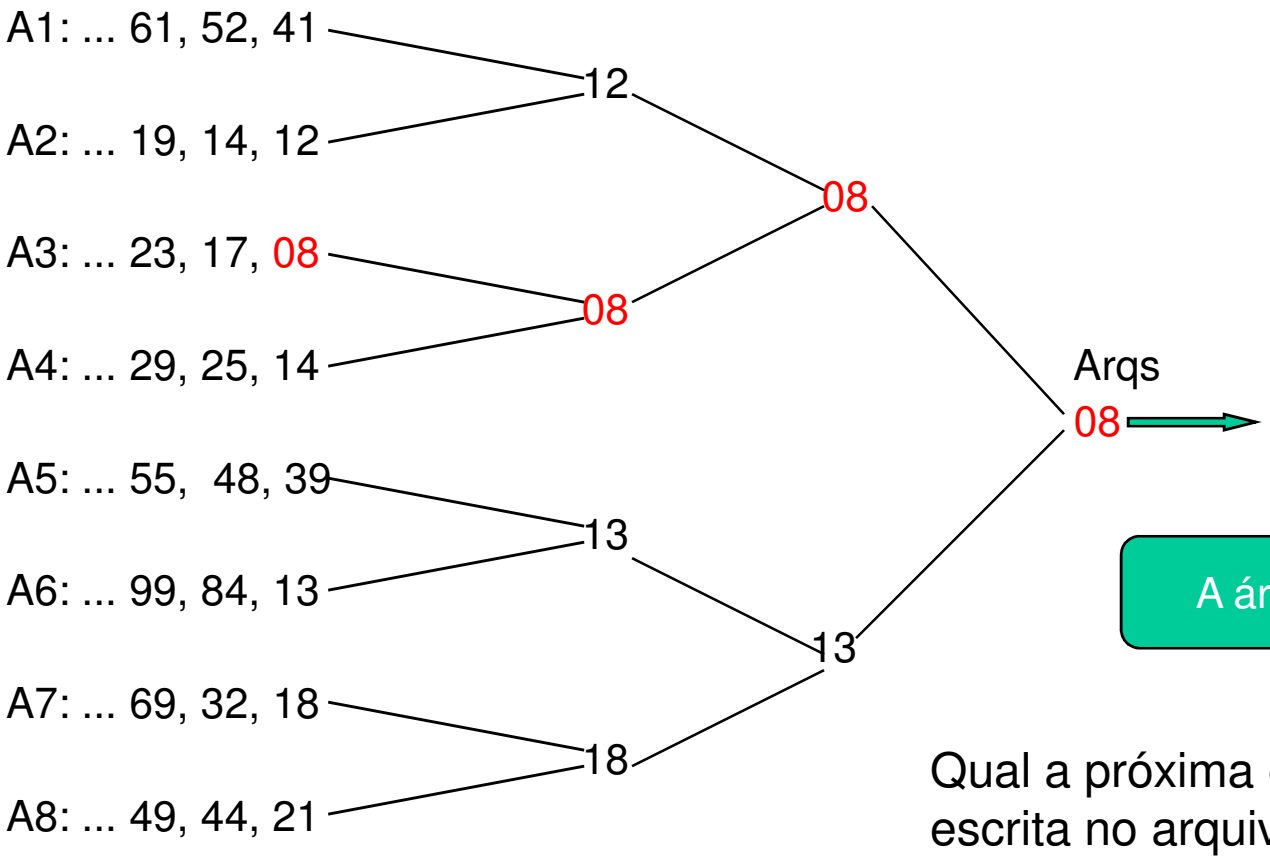


Os arquivos/listas A_i estão ordenados.
O procedimento retorna a chave
com o valor mínimo.

Intercalação Múltipla:

Árvore de Seleção

As novas comparações aproveitam boa parte da estrutura já montada (e armazenada) da árvore.



A árvore é binária.

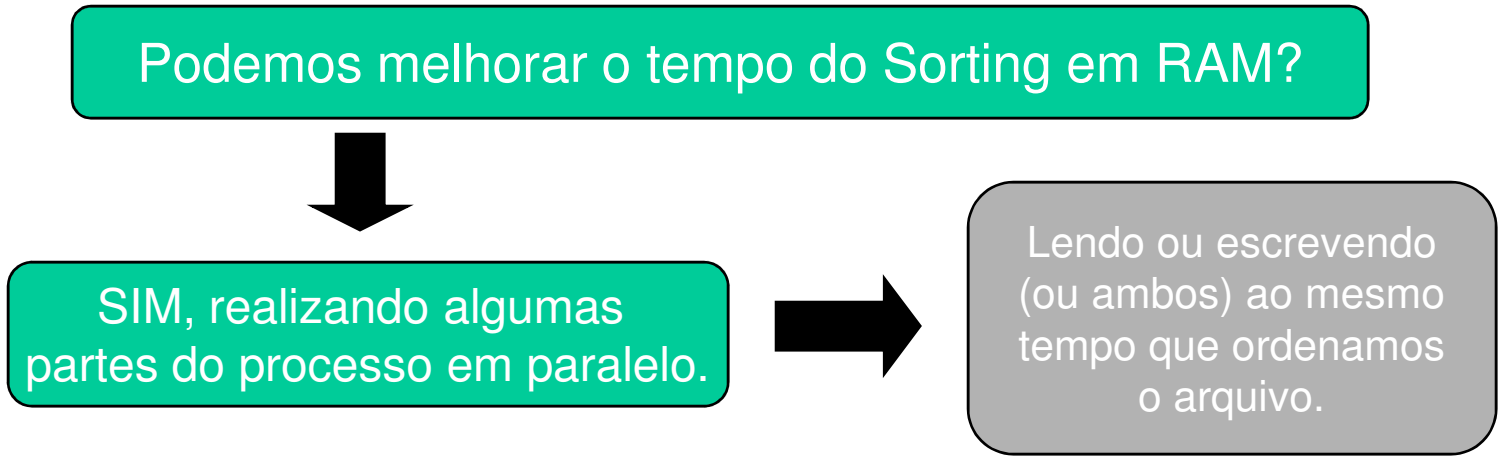
Qual a próxima chave a ser escrita no arquivo de saída?

Sorting em RAM

Internal Sorting:

1. Ler o arquivo (possivelmente inteiro) do disco para a memória
2. Ordenar os registros usando um algortimo de ordenação
3. Escrever o arquivo de volta no disco.

Tempo total = Soma dos tempos dos passos 1, 2 e 3.



Sorting em RAM (Heapsort)

HEAPSORT: sobrepondo processamento e I/O.

O heapsort utiliza uma estrutura de dados chamada heap para ordenar os elementos a medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser sucessivamente removidos da raiz da heap, na ordem desejada.

Não é necessário esperar até que todo o arquivo esteja na memória para iniciar a ordenação.

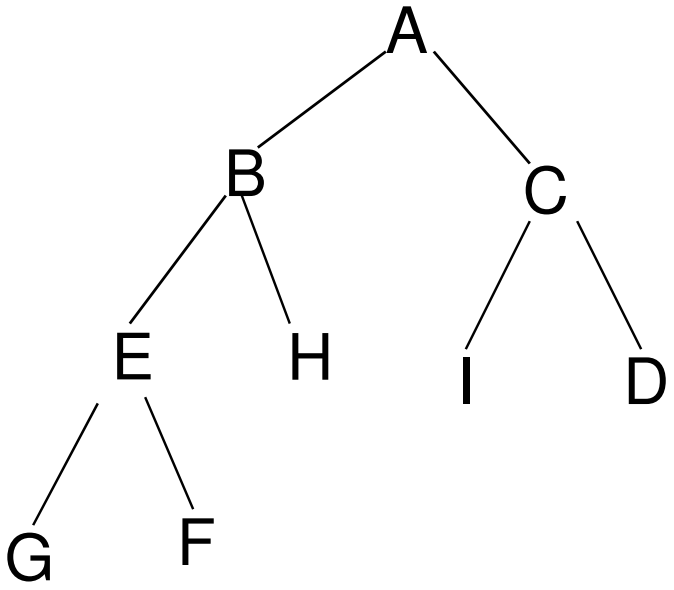
Heapsort é relativamente rápido. O heap pode ser construído em paralelo à leitura.

Sorting em RAM (Heapsort)

- HEAP
  - É uma árvore binária com as seguintes propriedades:
    - Cada nó tem uma única chave, e aquela chave é maior ou igual a chave de seu pai.
    - É uma árvore binária quase completa.
    - A raiz tem a menor chave.
    - Pode ser armazenada num vetor, onde cada nó i tem como filhos os nós $2i$ e $2i + 1$; e o pai de um nó j é o nó $\lfloor j/2 \rfloor$.

Sorting em RAM (Heapsort)

HEAP (exemplo)



1	2	3	4	5	6	7	8	9
A	B	C	E	H	I	D	G	F

A heap pode ser representada como uma árvore ou como um vetor.

Sorting em RAM (Heapsorting)

- Construção da Estrutura HEAP em tempo de leitura/Escrita:
    - Duas Partes:
        - Construção da HEAP
        - Escrita das Chaves Ordenadamente
- Exemplo:
    - Inserção das Chaves: F D C G H I B E A

Sorting em RAM (Heapsorting)

- Exemplo:
  - Inserção das

  Chaves: F D C G H I B E A

  Como fica a estrutura da HEAP na forma de vetor durante a inserção?

Sorting em RAM (Heapsort)

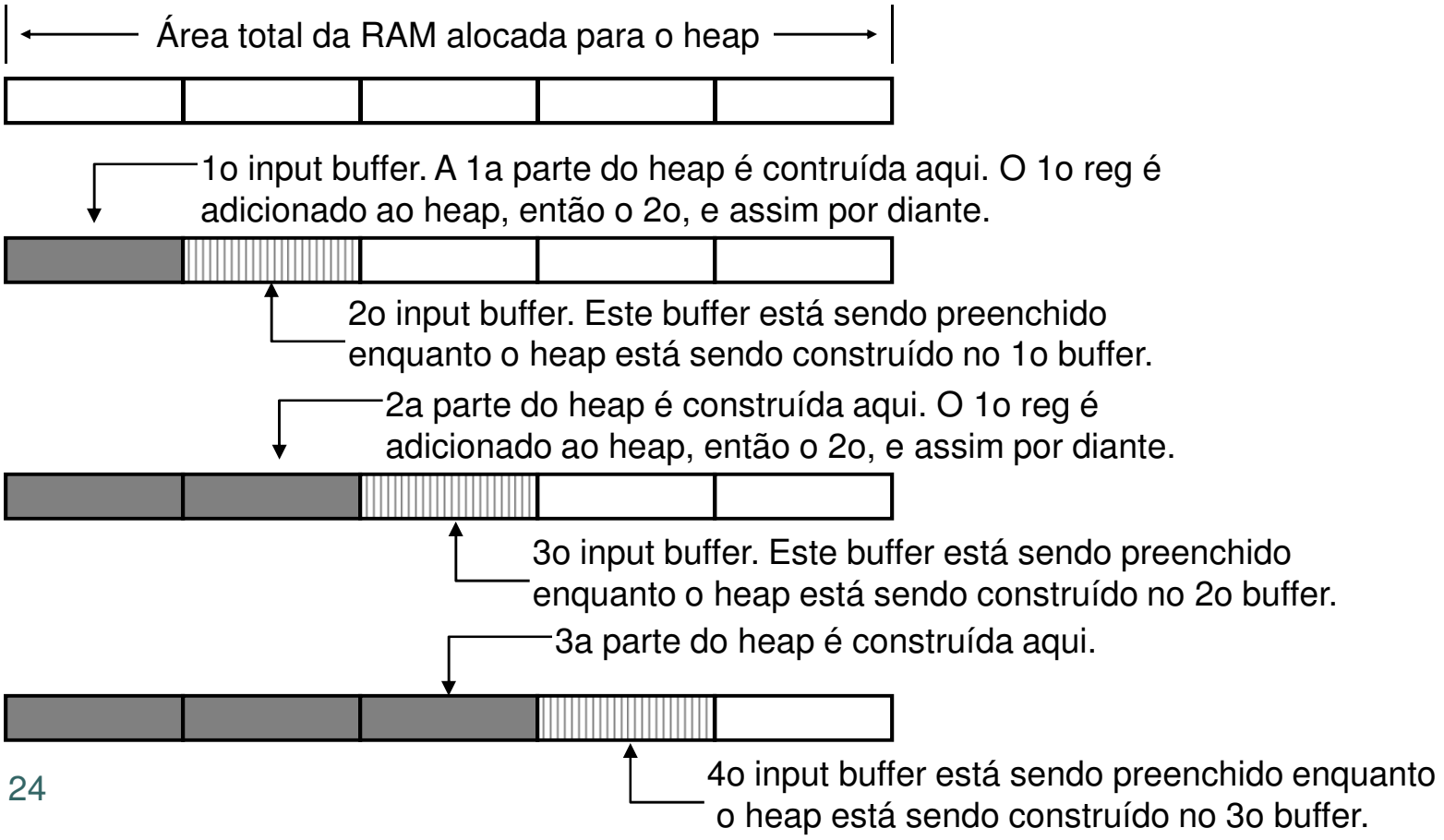
Construindo a HEAP enquanto vai lendo o arquivo:

```
For i:=1 to RECORD_COUNT do
    Leia o próximo registro e coloque no final do vetor; chame-o de k
    Enquanto k for menor do que a chave do seu pai:
        Troque k com o seu pai.
```

Sorting em RAM (Heapsort)

- Construindo o HEAP enquanto vai lendo o arquivo:
  - Como fazer a leitura sobrepôr o procedimento de construção do HEAP?
    - Não vamos fazer um seek cada vez que queremos um novo registro.
    - Vamos ler um bloco de registros por vez e colocar num buffer, e então operar nos registros que estão naquele bloco antes de proceder a um novo bloco.
    - Com MULTIPLE BUFFERING, podemos processar as chaves que estão num bloco, e simultaneamente podemos estar lendo outros blocos.

Sorting em RAM (Heapsort)



Sorting em RAM (Heapsort)

- Sorting enquanto vai escrevendo o arquivo:
  - Algoritmo para remover o conteúdo do heap ordenado:

```
For i:=1 to RECORD_COUNT do
    Remove a chave da 1a posição do array (é a menor chave)
    Põe o último elemento do array (chame-o de K) na raiz (início do vetor), e
        defina o heap como tendo um elemento a menos.
    Restaure as propriedades do heap, ou seja, enquanto K for maior do
        que um dos seus filhos:
        Troque K com a menor das chaves dos seus filhos.
```

Sorting em RAM (Heapsort)

- Sorting enquanto vai escrevendo o arquivo:
  - Sabemos imediatamente qual chave será escrita 1o no arquivo ordenado; depois sabemos qual será escrita em 2o; e assim por diante.
  - Portanto, assim que nós tivermos identificado um bloco de chaves, nós podemos escrever aquele bloco no disco, enquanto identificamos o próximo bloco.

Na próxima aula...

- Mostrar como merging (intercalação) fornece a base para ordenação de arquivos grandes.