Universidade Federal
de Ouro Preto

ENYA LUÍSA GOMES DOS SANTOS
19.2.4201

RESUMO II - ORDENAÇÃO EXTERNA

Resumo apresentado por exigência da disciplina BCC203 - Estrutura de Dados II, da Universidade Federal de Ouro Preto.
Professor: Guilherme Tavares Assis

Enya Luísa Gomes dos Santos
19.2.4201

Ordenação externa

O que é?

Técnicas de ordenação quando todos os dados não cabem em memória principal. A métrica mais importante em um método de ordenação externa é o número de transferência, entre a memória principal e a memória externa. É importante pensar nas restrições de acesso a dados em mecanismos de memórias secundárias utilizados.

O método mais importante de ordenação externa é o de ordenação por intercalação. **Intercalar**: combinar dois ou mais blocos ordenados em um único bloco ordenado. Vale ressaltar que é uma operação muito custosa, pois envolve transferências de dados entre a memória principal e a memória secundária.

Estratégia geral dos métodos de ordenação externa:

1. Quebra do arquivo em blocos de tamanho disponível na memória interna.
2. Ordenação de cada bloco na memória interna.
3. Intercalação dos blocos ordenados, fazendo várias passadas sobre o arquivo, em que, a cada passada são criados blocos ordenados cada vez maiores, até que todo o arquivo esteja ordenado.

Intercalação balanceada de vários caminhos

Imagine um arquivo com os 22 registros a seguir:

I N T E R C A L A C A O B A L A N C E A D A

Considerações:

- A memória interna é capaz de armazenar três itens;
- Há seis unidades disponíveis de fita magnética;

Nesse método, a ideia de ser vários caminhos, vem do fato de que utiliza-se de dispositivos de memória secundária de forma temporária.

Os métodos baseados em interação possuem duas fases, sendo elas:

Fase 01: Criação dos blocos ordenados.

Enya Luísa Gomes dos Santos
19.2.4201

Fase 02: Intercalação.

Representação:

I N T E R C A L A C A O B A L A N C E A D A

Criação dos blocos ordenados:

Na memória principal:

Pega os três itens da memória secundária (quantidade que cabe em memória principal) e passa para o vetor na memória principal, a fim de ser ordenado.	Vetor já ordenado em memória principal através de algum método de ordenação.
I N T	I N T
E R C	C E R

A cada passada de ordenação de três itens, o vetor já ordenado é inserido em memória secundária, as fitas de entrada. O número de fitas deve ser igual a quantidade de itens que cabem na memória principal.

Fitas de entrada:

Fita 01: **I N T A C O A D E**

Fita 02: **C E R A B L A**

Fita 03: **A A L A C N**

Fase de intercalação:

1. Leitura do primeiro registro de cada fita.
2. Retirada do registro contendo a menor chave, armazenando-o em uma fita de saída.
3. Leitura de um novo registro da fita de onde o registro retirado é proveniente.

Enya Luísa Gomes dos Santos
19.2.4201

- Ao ler o terceiro registro de um dos blocos, a fita correspondente fica inativa.
- A fita é reativada quando os terceiros registros das outras fitas forem lidos.
- Neste momento, um bloco de nove registros ordenados foi formado na fita de saída.

4. Repetição do processo para os blocos restantes.

As fitas de saída recebem os blocos intercalados.

Lê o primeiro item de cada fita e armazena no vetor da memória principal:

I	**C**	**A**

Remove o menor elemento e ele é substituído pelo próximo elemento de onde veio, e segue repetindo o processo:

I	**C**	**A**

I	**C**	**L**

I	**R**	**L**

Quando se tira o L, não se pega o próximo elemento da fita, pois o bloco o qual L pertence já acabou. O que acontece é que essa posição fica vazia:

N	**R**	**--**

.
.
.

Quando se chega no final de todos os blocos 1 de todas as três fitas finaliza-se a intercalação completa, e na Fita 04 é gerado o primeiro bloco ordenado. E a mesma ideia é replicada para os blocos seguintes nas fitas seguintes:

--	**--**	**--**

Enya Luísa Gomes dos Santos
19.2.4201

Fitas de saída:

Fita 04: **A A C E I L N R T**

Fita 05: **A A A B C C L N O**

Fita 06: **A A D E**

Após essa etapa, a estratégia é replicada, agora as fitas de saída se tornarão fitas de entrada.

Pegando o primeiro de cada uma das fitas de saída:

A	A	A

A	A	A

C	A	A

C	A	A

.
.
.

Após chegar ao final dos blocos das fitas de saída, se obtém um bloco com todos os dados ordenados na fita de entrada 01. Nesse momento, com a fita 02 e 03 vazias, e a fita 01 com todos os itens ordenados, sabemos que esse processo **chegou ao fim**.

Fitas de entrada:

Fita 01: **A A A A A A A B C C C D E E I L L N N O R T**

Fita 02:

Fita 03:

Ao total, nesse exemplo, houve três passadas sobre o arquivo, 1 na primeira fase e 2 na segunda fase.

Enya Luísa Gomes dos Santos
19.2.4201

Calculando a quantidade de passadas em um arquivo:

n -> número de registro no arquivo

m -> número de quantidade de posições existentes na memória principal para realizar o processo de ordenação.

Na primeira fase do processo de ordenação gera n/m blocos ordenados.

f -> número de fitas utilizadas em cada passada, (entrada)

P(n) -> número de passadas na fase de intercalação

$$P(n) = \log_f (n/m)$$

No exemplo, tem-se: $P(n) = \log_3 (22/3) = 2 + 1$ (da primeira fase)

No exemplo, foram utilizadas **2f** fitas para uma intercação-de-**f**-caminhos.

Também há a f + 1, onde tem-se *f* fitas de entrada e 1 fita de saída.

1º Fase: Essa fase não se diferencia em nada da anterior.

Fitas de entrada:

Fita 01: **I N T A C O A D E**

Fita 02: **C E R A B L A**

Fita 03: **A A L A C N**

2º Fase (intercalação):

Passo 01: Da mesma forma será ordenado os blocos da primeira fase, a diferença é que agora não há três fitas para armazenar os blocos, e sim um que receberá todos os blocos.

Fita de saída

Fita 04: **A A C E I L N R T A A A B C C L N O A A D E**

Passo 2: Agora cada bloco da fita de saída será colocado nas fitas de entrada. Esse processo envolve mais uma passada dentro do arquivo.

Fitas de entrada:

Fita 01: **A A C E I L N R T**

Fita 02: **A A A B C C L N O**

Fita 03: **A A D E**

Passo 3: Agora a intercalação ocorre novamente, gerando um bloco com os dados ordenados na fita de saída.

Fita de saída

Fita 04: **A A A A A A A B C C C D E E I L L N N O R T**

1ª fase: Geração dos blocos iniciais ordenados:

- Método de ordenação interna

2ª fase: Intercalação

- Balanceada de vários caminhos (2f fitas)
- Balanceada de vários caminhos (f+1 fitas)

Observações:

O fato de distribuir inicialmente em mais fitas, pode melhorar a quantidade de intercalação.

Uma coisa que pode ser feita é a leitura total do bloco - caso haja espaço na memória principal - pois assim diminui a quantidade de transferências realizadas. Por exemplo, ler todos os blocos **I N T C E R A A L** - um em cada fita - de uma vez e armazenar em outra estrutura da memória principal.

Implementação por meio de Substituição por seleção

Essa técnica utiliza o *heap.* Em uma estrutura de heap o pai tem que ser sempre maior (Max Heap) ou menor (Min Heap) que os dois filhos.

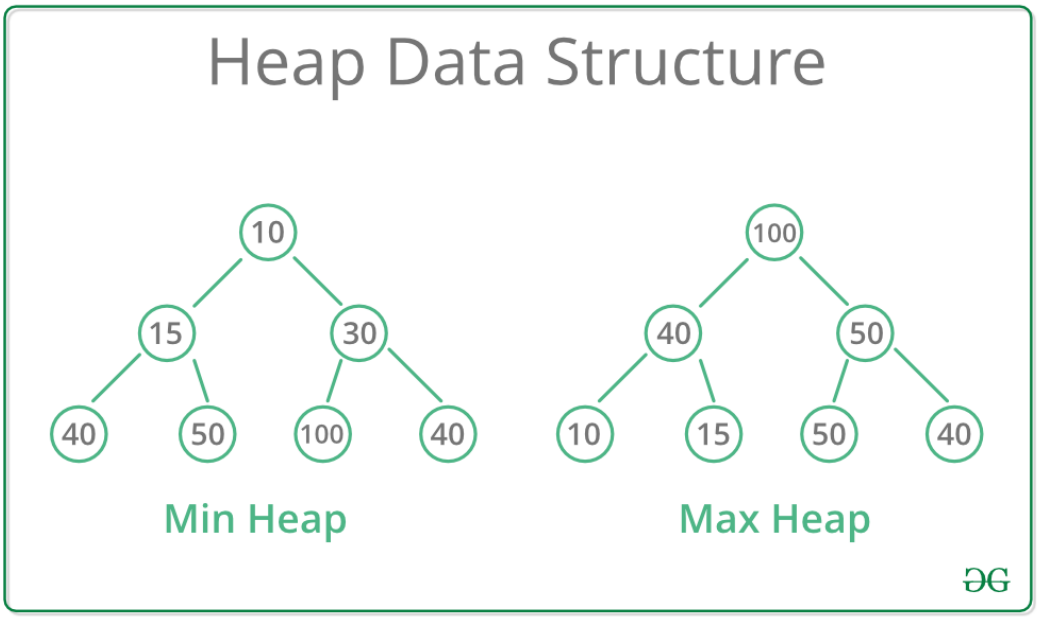

Figura 01 - Representação de uma estrutura Heap. Fonte: https://www.geeksforgeeks.org/heap-data-structure/

Sem esse método, na fase de intercalação do vetor na memória principal, a todo momento é feito uma busca linear O(n) para encontrar o menor elemento.

Porém, com a substituição por seleção isso melhora, antes de encontrar o melhor, faz-se o heap no vetor da memória principal, e o heap vai reconstituindo (verificar se os filhos do pai continuam menor, caso isso não ocorra, o heap é reconstituído), e essa reconstituição é O(log n).

Essa reconstituição pode gerar um problema, como os itens têm seu lugar trocado, a posição do meu vetor não indica mais a fita do qual o item se refere. Assim, não mais a possibilidade de referenciar a fita pela posição do vetor, uma solução é que, dentro do vetor, além do item, será guardado também a fita de onde ele veio.

O mais interessante do *heap* é que com ele, há uma grande ajuda na primeira fase do método, na fase de geração dos blocos ordenados.

1ª fase: Geração dos blocos iniciais ordenados:

- ***Substituição por seleção**

2ª fase: Intercalação

- Balanceada de vários caminhos (2f fitas)
- Balanceada de vários caminhos (f+1 fitas)

Representação, utilizando a substituição por seleção na primeira fase:

I N T E R C A L A C A O B A L A N C E A D A

1ª Fase:

I	**N**	**T**

Aplicando o Heap:

I	**N**	**T**

O **E** é marcado pois é **menor que o elemento que saiu**, o **I**, isso significa que nesse momento, esse elemento é considerado maior que os demais.

E*	**N**	**T**

Como o **R** é **maior que o elemento que saiu**, o **N**, ele não será marcado.

R	**E***	**T**

C*	E*	T

T	E*	C*

Nesta situação todos os elementos estão marcados, logo o que irá acontecer é que todos serão desmarcados, e o processo inicia normalmente novamente, porém agora, também começa a geração de um novo bloco, que entrará na nova fita de entrada.

A*	E*	C*

A	E	C

L	E	C

A*	E	L

.
.
.

Fitas de entrada:

Fita 01: **I N R T A A C E N**

Fita 02: **A C E L A A D**

Fita 03: **A A B C L O**

Os blocos gerados têm tamanhos diferentes, esse fato é muito bom para a intercalação, pois além de se ter menos blocos formados, temos blocos que já estão com os elementos ordenados.

Observações:

Caso os dados já estejam ordenados em ordem crescente, como ao sair, o próximo item a ser inserido no array sempre será maior, não haverá nenhum

elemento marcado e será gerado somente um bloco representando os itens já ordenados, e a etapa de intercalação nem irá existir, pois chegará ao fim.

Já quando o arquivo está ordenado decrescentemente, todos os novos elementos serão marcados, o que vai gerar muitos blocos do tamanho do vetor interno, se igualado ao método de ordenação interna. Esse sendo o pior caso.

A leitura dos itens do arquivo, com exceção dos primeiros, será de um a um, o que gera mais números de transferências, o que pode demorar. Uma solução, se houver memória, é, após pegar os primeiros itens, em uma estrutura auxiliar com um vetor temporário que lê uma maior quantidade de elementos, diminuindo assim a quantidade de transferências.

O processo de heap, como dito anteriormente, também pode ser usado no processo de intercalação. Os melhores casos para uso do heap na intercalação são quando o número de fitas é **f >= 8**, realizando $\log_2 f$ comparações para se encontrar o menor item.

Considerações práticas

Os métodos de pesquisa externa, para dados muito grandes é um processo demorado, uma boa solução é a utilização de uma ou mais unidades independentes para processamento de entrada e saída.

Para se escolher as quantidades de fitas a serem utilizadas, o ideal é considerar o suficiente para completar a ordenação em poucos passos, ou seja, **f** o maior possível, esse fator depende também de parâmetros relacionados com o sistema computacional disponível.

Intercalação Polifásica

Veio para solucionar problemas da intercalação balanceada, sendo eles:

- Necessita de um grande número de fitas, fazendo várias leituras e escritas entre as fitas envolvidas.
    - Para uma intercalação balanceada de f caminhos são necessárias, geralmente, 2f fitas.

- Alternativamente, pode-se copiar o arquivo de uma única fita de saída para f fitas de entrada, reduzindo o número de fitas para f + 1.

- Há um custo de uma cópia adicional do arquivo.

Processo de funcionamento da intercalação polifásica:

- Os blocos ordenados são distribuídos de forma desigual entre as fitas disponíveis.
- Uma fita é deixada livre.
- Em seguida, a intercalação de blocos ordenados é executada 20 até que uma das fitas de entrada se esvazie.
- A fita vazia torna-se a próxima fita de saída.

  Observação: nesta etapa a primeira fase já foi executada.

■ Blocos ordenados obtidos por meio de seleção por substituição:

```
Fita 1: I N R T     A C E L   A A B C L O
Fita 2: A A C E N   A A D
Fita 3:
```

■ Intercalação-de-2-caminhos das fitas 1 e 2 para a fita 3:

```
Fita 1: A A B C L O
Fita 2:
Fita 3: A A C E I N N R T   A A A C D E L
```

■ Intercalação-de-2-caminhos das fitas 1 e 3 para a fita 2:

```
Fita 1:
Fita 2: A A A A B C C E I L N N O R T
Fita 3: A A A C D E L
```

■ Intercalação-de-2-caminhos das fitas 2 e 3 para a fita 1:

```
Fita 1: A A A A A A A B C C C D E E I L L N N O R T
Fita 2:
Fita 3:
```

Figura 02 - Exemplificação da intercalação polifásica

Observações:

O estado final do processo de ordenação é quando há apenas um bloco em apenas uma fita com as restantes vazias.

Enya Luísa Gomes dos Santos
19.2.4201

A intercalação é realizada em muitas fases.

As fases não envolvem todos os blocos.

Nenhuma cópia direta entre fitas é realizada.

Caso ocorra de os bloco ordenados serem distribuídos de forma igualitária entre as fitas, isso irá gerar, com a intercalação, somente uma fita com esses blocos intercalados, nessa situação é necessário copiar um dos bloco para outra fita, e assim o processo poderá continuar normalmente.

Análise:

- A análise da intercalação polifásica é complicada.
- O que se sabe é que ela é ligeiramente melhor do que a intercalação balanceada para valores pequenos de f.
- Para valores de f > 8, a intercalação balanceada de vários caminhos pode ser mais rápida.

Quicksort

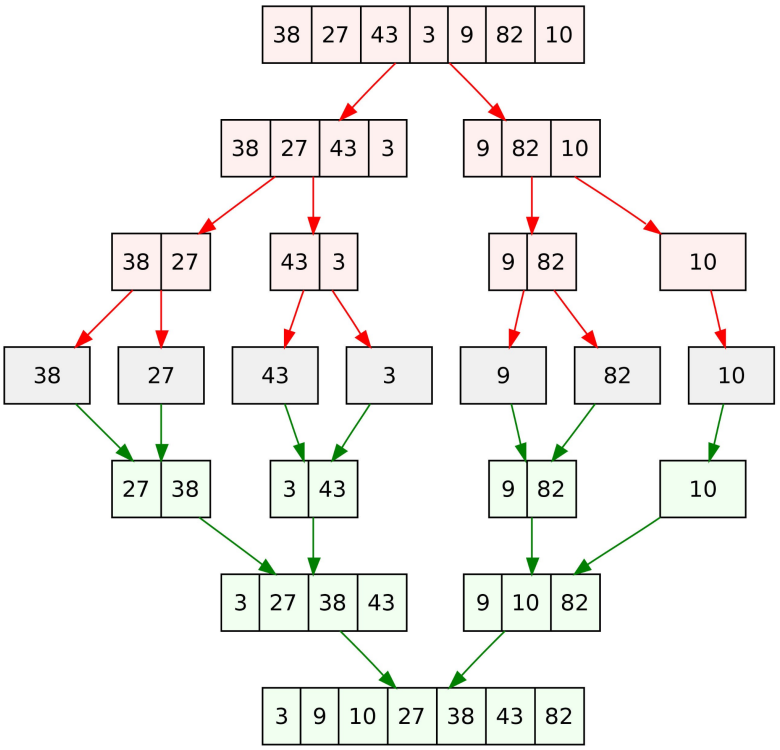

Figura 03 - Exemplificação do funcionamento do método de quicksort interno. Fonte: https://www.ma-no.org/en/programming/java/java-sorting-algorithm-quick-sort

Ideia do quicksort

- Dividir para conquistar
- Escolher o pivô
- Dividir os elementos entre os menores que o pivô e outro entre os maiores que o pivô
- Complexidade n logarítmica O(n log n)

Quicksort Externo

Os métodos anteriores trabalham com o método de intercalação. O quicksort externo utiliza o paradigma de divisão e conquista.

O algoritmo ordena *in situ* - a ordenação ocorre dentro do próprio arquivo - um arquivo A = {R_1, … , R_n} de *n* registros. A grande diferença entre o quicksort externo e o interno é que, o pivô não é de só um elemento, se trata de um conjunto de elementos, e esse conjunto do pivô será colocado dentro da memória interna, e o máximo de unidade de memória interna utilizada é log n, ou seja, O(*log n*).

Para ordenar o arquivo A = {R1, ... , Rn}, o algoritmo:

- Particiona *A* da seguinte forma:
  $\{R_1, ... , R_i\} \leq R_{i+1} \leq R_{i+2} \leq ... \leq R_{j-2} \leq R_{j-1} \leq \{R_j, ... , R_n\}$
- Onde os registros ordenados $\{R_{i+1}, ..., R_{j-1}\}$ correspondem ao pivô do algoritmo e encontrando-se na memória interna durante a execução do mesmo.
- Chama recursivamente o algoritmo em cada um dos subarquivos gerados:
  $A_1 = \{R_1 , . . . , R_i\}$ e $A_2 = \{R_j, . . . ,R_n\}$
- Os subarquivos A_1 e A_2 contêm os registros menores que R_{i+1} e maiores que R_{j-1}, respectivamente.

Tamanho da área = j − i − 1, sendo necessariamente ≥ 3.

Enya Luísa Gomes dos Santos
19.2.4201

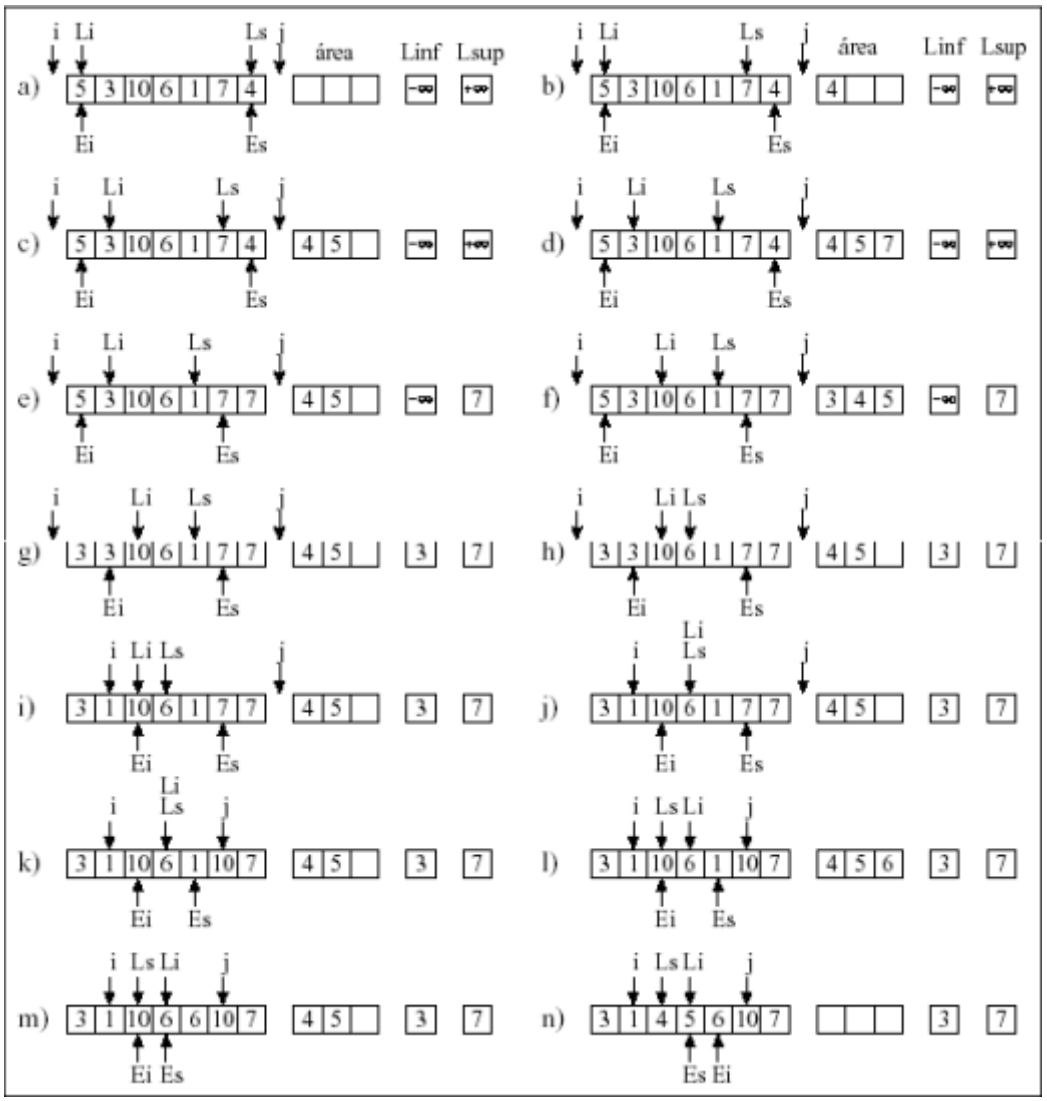

Figura 04 - Funcionamento do quicksort externo

Variáveis presentes na figura 04 e sua função:

- **Esq:** Limite em termos de posição inferior.
- **Dir:** Limite em termos de posição superior.
- **Li:** Apontador de leitura inferior.
- **Ls:** Apontador de leitura superior.
- **Ei:** Apontador de escrita inferior.
- **Es:** Apontador de escrita superior.
- **Linf:** Limite inferior do pivô, o valor que é imediatamente menor que o menor elemento do pivô, (-inf).
- **Lsup:** Limite superior do pivô, o valor que é imediatamente maior que o maior elemento do pivô, (+inf).

- ***i:** Valor retornado pela partição.
- ***j:** Valor retornado pela partição.

Funcionamento da partição:

Os primeiros "tamanho da área" (TamArea) - 1 registros são lidos, alternativamente (superior e inferior), dos extremos de *A* e armazenados na área de memória interna, os pivôs. E esse processo finaliza quando houver apenas uma posição livre para os pivôs na memória interna.

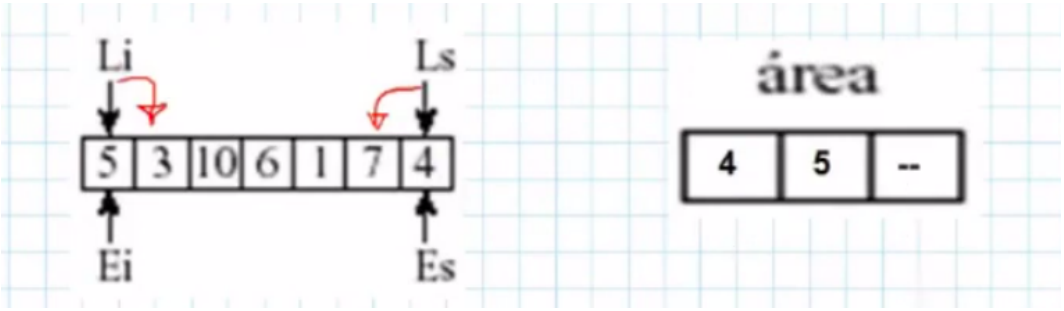

Figura 05 - Leitura dos itens para o pivô

Ao ler o o *TamArea-ésimo* registro*,* cuja chave é **C**, e a depender do valor de **C** teremos três situações:

1. **C** é comparada com **Lsup** e, sendo maior, **j** recebe **Es** e o registro é escrito em **A2**;
2. Caso contrário, **C** é comparada com **Linf** e, sendo menor, **i** recebe **Ei** e o registro é escrito em **A1**;
3. Caso contrário (**Linf** ≤ **C** ≤ **Lsup**), o registro é inserido na área de memória interna.

Lembrando que, a inserção na memória principal ocorre de forma ordenada. Quando a área de memória principal enche, deve-se remover um registro da mesma, considerando os tamanhos atuais de **A1** e **A2**.

Em suma, ou retira o menor, adiciona em **A1** e atualiza o **Linf** para esse valor, ou pega o maior, adiciona em **A2** e atualiza o Lsup para esse valor. Para decidir qual das duas opções será realizada temos que considerar o balanceamento dos dois subarquivos, ou seja, o objetivo é escrever o registro removido da memória no subarquivo de menor tamanho, para isso há as seguintes situações:

1. Sendo **Esq** e **Dir** a 1ª e a última posição de A, os tamanhos de **A1** e **A2** são, respectivamente, (**T1 = Ei – Esq**) e (**T2 = Dir – Es**).
2. Se **(T1 < T2)**, o registro de menor chave é removido da memória, sendo escrito em **Ei** (**A1**), e Linf é atualizado com tal chave.
3. Se **(T2 ≤ T1)**, o registro de maior chave é removido da memória, sendo escrito em **Es** (**A2**), e **Lsup** é atualizado com tal chave.

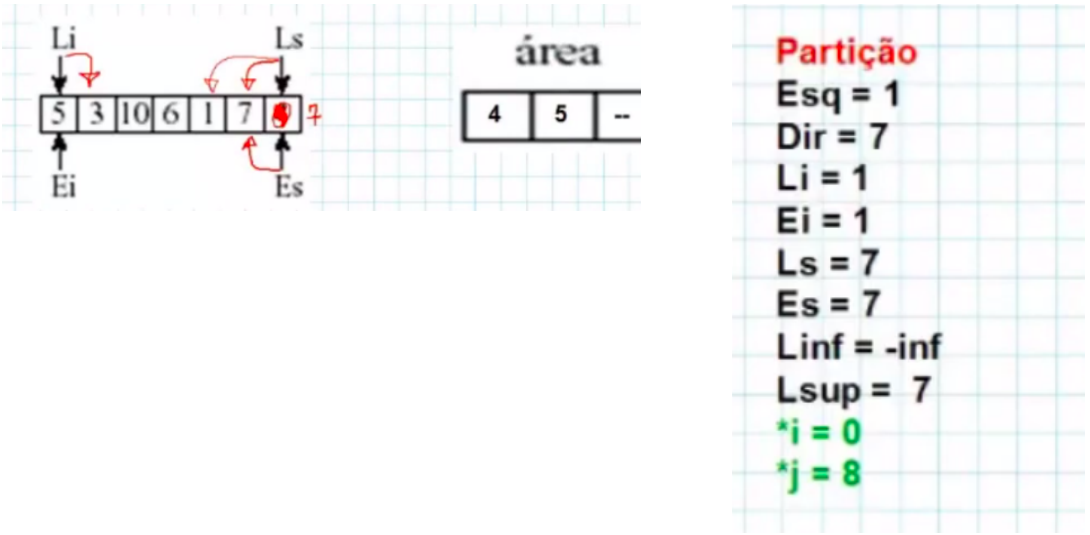

Figura 06 - Remoção de um registro da memória principal de acordo com as condições apresentadas acima.

Após essa remoção, o processo se repete pegando novamente a chave **C** e depois removendo novamente quando a memória principal está completa. O processo de partição continua até que **Li** e **Ls** se cruzem, ou seja, (**Ls** < **Li**).

Neste momento, os registros armazenados na área de memória interna devem ser copiados, já ordenados, em **A**. Enquanto existir registros na área de memória principal, o menor deles é removido e escrito na posição indicada por **Ei** em **A**.

No fim da primeira execução, o valor retornado **i*** indica os elementos menores que o pivô e **j*** os elementos maiores que o pivô, ou seja **A1** e **A2**. Com isso, ocorre uma chamada recursiva para **A1** e **A2**, e a parte do meio, que se trata do pivô, já está ordenada. **A1** varia de **Esq** até **i***, e **A2** de **j*** até **Dir**.

Observações:

No momento da leitura, **C**, se **$E_i = L_i$** e a leitura de **C** estiver ocorrendo em **Ls**, a alternância é quebrada para se ler em **L_i** por desse modo se garante que, antes de escrever um elemento em **E_i** o elemento dessa posição já tenha sido lido. O contrário também pode acontecer. Ou seja, para garantir que os apontadores de escrita estejam atrás dos apontadores de leitura, a ordem alternada de leitura é interrompida se **$(L_i = E_i)$** ou **$(L_s = E_s)$**.

Análise

Seja **n** o número de registros a serem ordenados e seja **b** o tamanho do bloco de leitura ou gravação do SO.

Melhor caso: O (n / b)

- Ocorre quando o arquivo de entrada já está ordenado.

Pior caso: O (n2 / TamArea)

- Ocorre quando as partições geradas possuem tamanhos inadequados: maior tamanho possível e vazio.
- A medida que n cresce, a probabilidade de ocorrência do pior caso tende a zero.

Caso Médio: O (n / b × log (n / TamArea))

- Maior probabilidade de ocorrer.