Ordenação Externa

🎓 **Aluno**: Lucas de Araújo
Matrícula: 18.2.4049
Professor Doutor: Guilherme Tavares de Assis

Apresentação do Tema

A ordenação externa consiste em solucionar o problema de quando temos um arquivo de tamanho maior do que é possível armazenar na nossa memória interna (RAM) disponível. Nos métodos de ordenação externa, o número de comparações ainda é uma métrica consideravelmente boa para se analisar a eficiência de

determinado algoritmo, porém, o número de transferências (igual na pesquisa externa) é a melhor métrica para se avaliar a eficiência do algoritmo. Além disso, nas memórias externas, os dados ficam em um arquivo sequencial

Observações:

- Os algoritmos devem **diminuir o número de acesso** às unidades de memória externa sempre que possível
- Apenas **um registro** pode ser acessado em um dado momento (restrição forte), diferenciando-se da possibilidade mais branda de acesso feita em um vetor

Fatores que determinam as diferenças entre técnicas de ordenação externa e interna

- O custo de acesso a memória secundária **é muito maior** que o custo de acesso a memória primária
- Restrições de acesso a dados:
  - Fitas são acessadas somente sequencialmente
  - Em discos, acesso direto é muito caro
- Os métodos de ordenação externa são dependentes do hardware atual (assim como os métodos de pesquisa interna)

Métodos de Ordenação Externa

Intercalação

O método mais importante de ordenação externa é a ordenação por **intercalação.** Intercalar consiste em combinar dois ou mais blocos ordenados em único bloco ordenado (similar ás páginas) e transferí-los para a memória principal (saindo da memória secundária). Com isso, o número de passadas sobre o arquivo é reduzida junto ao custo do algoritmo

Dessa forma, podemos dizer que o foco dos algoritmos de ordenação externa consiste em reduzir a intercalação, ou seja, reduzir o número de passadas sobre o arquivo.

- Uma medida boa de complexidade de um algoritmo por intercalação é o número de vezes que determinado item é lido ou escrito na memória interna.
  - Quanto menor for este número de vezes, melhor.

- Bons métodos de ordenação externa geralmente envolvem menos do que dez passadas sobre o arquivo

🗣️ "Qual a estratégia geral dos métodos de ordenação?"

1. Quebrar o arquivo em blocos do tamanho da memória interna disponível
2. Ordenar cada bloco na memória interna
3. Intercalar os blocos ordenados, fazendo várias passadas sobre o arquivo.

- Cada passada irá gerar blocos ordenados cada vez maiores, até que todo arquivo esteja ordenado

Exemplo de Intercalação (Balanceada de Vários Caminhos)

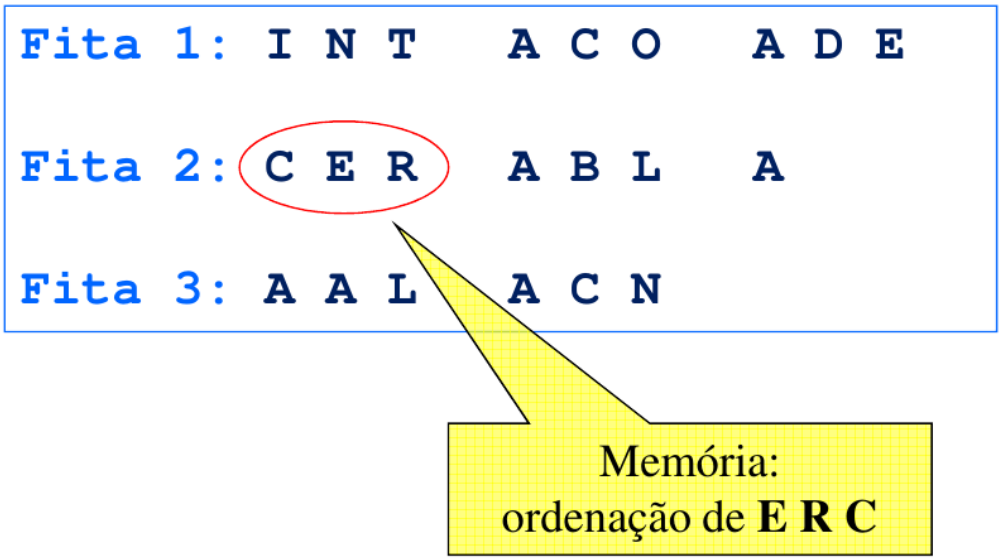
🗣️ "Ordene um arquivo armazenado em uma fita de entrada que possui os 22 registros seguintes"

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

📌 Considerações Importantes:
A memória interna tem capacidade para três itens
A fita magnética tem capacidade para seis unidade

Primeira Fase: Criação dos Blocos Ordenados

Nesta fase será feita a quebra do arquivo em blocos do tamanho disponível na memória interna, e logo após trazê-los para memória interna, será feita a devida ordenação deste bloco.



Após a ordenação, estes blocos serão levados novamente paras as fitas de entrada como disposto acima.

> 📌 **Observação**: O número de fitas deve ser a maior quantidade de possível para que tenhamos um processo mais eficiente

Segunda Fase: Intercalação

- Primeiro fazemos a leitura do primeiro registro de cada fita e jogamos eles para memória principal

$$\begin{matrix} I & C & A \\ F1 & F2 & F3 \end{matrix}$$

- Segundamente, fazemos uma busca da retirada do registro contendo a menor chave, armazenando-o em uma fita de saída

$$\begin{matrix} A \\ F3 \end{matrix}$$

> 📌 Valor da complexidade: $O(F)$
> Percorremos todo o vetor em busca do registro contendo a menor chave

- Em seguida, fazemos a leitura de um novo registro da fita de onde o registro retirado é proveniente (Nesse caso, o registro da fita $F3$)
    - Ao ler o terceiro registro de um dos blocos (ou seja, o último nesse cenário), 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
- Por fim, repetimos estes mesmos passos para os blocos restantes

Após a **1° Passada** sobre o arquivo na fase de intercalação, teremos a seguinte disposição das fitas de saída

```
Fita 4: A A C E I L N R T
Fita 5: A A A B C C L N O
Fita 6: A A D E
```

Repetindo o processo novamente para estas fitas, teremos no final o seguinte arquivo ordenado (com **o total de 2 passadas**)

```
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:
```

Nós encerramos esse processo quando todas as fitas fitas de entrada que temos, somente a primeira se encontra preenchida e todas as outras encontram-se vazias

Informações sobre a Intercalação Balanceada de Vários Caminhos

Para ordenar um arquivo de tamanho arbitrário, podemos calcular a quantidade de passadas necessárias para ordená-lo da seguinte forma

- Seja N o número de registros do arquivo
- Seja o M o número de palavras possíveis na memória interna (sendo a palavra equivalente a um registro do arquivo)

Com isso, temos que a primeira fase produz $\frac{N}{M}$ blocos ordenados

- Agora, seja $P(N)$ o número de passadas na fase de intercalação
- Seja F o número de fitas utilizadas em cada passada

No fim, teremos a seguinte fórmula:

$$P(N) = \log_F(\frac{N}{M})$$

No exemplo anterior, utilizando a fórmula, temos:

$$P(N) = \log_3(\frac{22}{3}) \approx 2$$

No nosso exemplo, foram utilizadas $2F$ fitas para uma intercalação-de-F-caminhos. É possível usar apenas $F + 1$ fitas, a saber:

- Encaminhe todos os blocos para uma única fita de sáida
- Redistribua estes blocos entre as fitas de onde eles foram lidos

No nosso exemplo, seriam necessário apenas quatro fitas $(3 + 1)$ desta forma

- A intercalação dos blocos a partir das fitas 1, 2 e 3 iria ser dirigida para a fita 4
- Ao final, o segundo e o terceiro bloco ordenados de nove registros seriam transferidos de volta para as fitas 1 e 2

> 📌 O processo de redistribuição da fita de saída para colocar os blocos novamente nas fitas de entrada implicam em mais uma passada sobre o arquivo, o que automaticamente gera um maior custo.
> Porém, mesmo com esta etapa adicional, em certos cenários, trabalhar com $F + 1$ fitas pode ser mais eficiente que $2F$ fitas

Implementação por meio de *Substituição por Seleção*

A implementação do método de intercalação balanceada (descrito aqui) pode ser feita utilizando as famigeradas **filas de prioridades**

- As duas fases do método podem ser implementadas de forma eficiente e elegante

- Substitui-se o menor item existente na memória interna pelo próximo item na fita de entrada

A estrutura ideal para implementação é o **Heap**

> "Em ciência da computação, um heap (monte) é uma estrutura de dados especializada, baseada em árvore, que é essencialmente uma árvore quase completa que satisfaz a propriedade heap: se P é um nó pai de C, então a chave (o valor) de P é maior que ou igual a (em uma heap máxima) ou menor que ou igual a (em uma heap mínima) chave de C. O nó no "topo" da heap (sem pais) é chamado de nó raiz."
> - Fonte

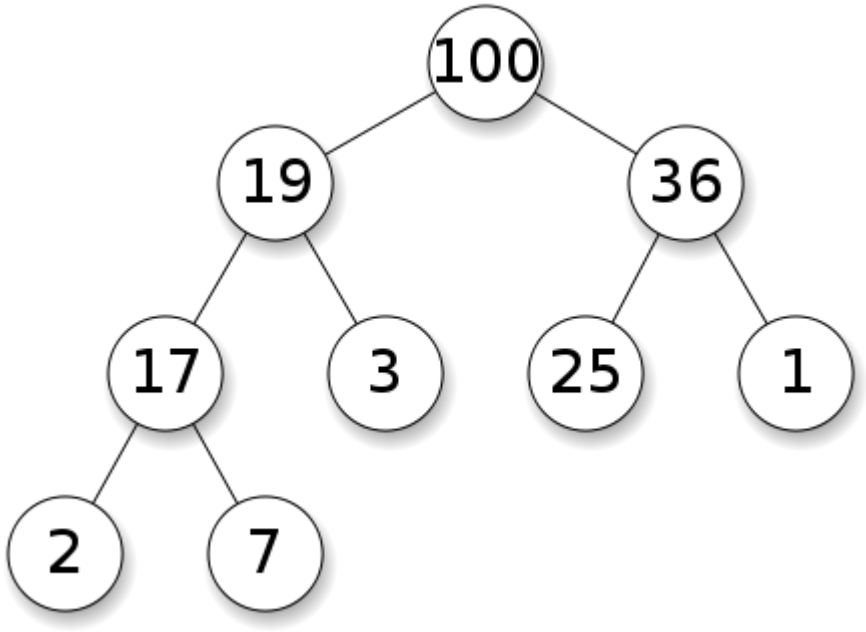


Ilustração de um Max Heap

Com a utilização da estrutura heap, nós podemos reduzir o tempo de comparação dentro dos vetores na memória principal e com isso também reduzir o tempo de intercalação

- O primeiro elemento no heap será o menor elemento do vetor e irá ser colocado dentro das fitas.

- Dessa forma, a cada iteração, é feita uma reconstituição do heap para que o menor elemento sempre esteja posicionado no primeiro lugar do vetor

O heap possui complexidade logarítmica, ou seja, é mais eficiente que a complexidade linear apresentada anteriormente

Porém por conta do uso do heap, perdemos a referência da fita que foi responsável por trazer determinado registro. Por conta disso, nosso vetor deve guardar duas posições

1. A chave do registro
2. Número da fita de onde veio esta chave

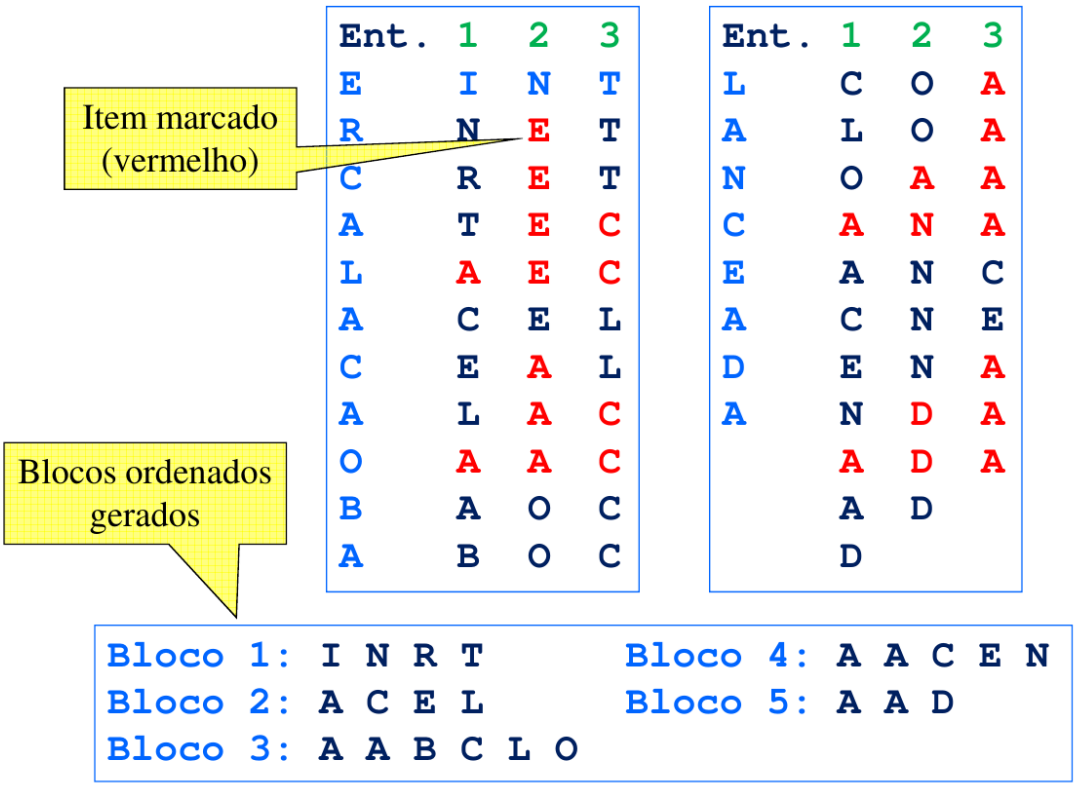
O heap também é muito eficiente no processo de geração dos blocos ordenados. Podemos utilizá-lo da seguinte forma

- Inicialmente, os M itens são inseridos na fila de prioridades inicialmente vazia
- O menor item da fila de prioridades é substituído pelo próximo item de entrada
    - Se o próximo item é menor do que o que está saindo, então ele deve ser marcado como membro do próximo bloco, e assim, tratado como maior do que todos os itens do bloco corrente

E é menor I, logo o elemento é marcado

- Quando um item marcado vai para o topo da fila:
    - O bloco corrente é encerrado
    - E por fim, um novo bloco ordenado é iniciado

> 📌 Se todos os elementos do vetor estiverem marcados, teremos que desmarcar todos e iniciar a criação de um novo bloco e levá-los a uma nova fita



Caso chegarmos no cenário onde todos os elementos estão todos ordenados, o elemento que irá entrar no heap a partir daqui sempre será maior que o elemento que estiver saindo. Portanto, nenhum elemento será marcado e com isto, nenhum novo bloco será gerado e não haverá intercalação. No final, teremos um bloco na fita 1 que representa o arquivo ordenado

No nosso pior caso (arquivo decrescente sendo ordenado ascendentemente), teremos a complexidade equivalente ao método de ordenação interna. Nesse cenário, teremos criações de blocos homogêneos do mesmo tamanho do vetor utilizado para alocação durante a primeira fase

É aconselhável utilizar-se o heap quando a quantidade de fitas é $F \geq 8$, pois o método é considerado adequado graças a possibilidade de possuirmos um vetor em memória principal de maior tamanho e com isso, realizando cerca de $\log_2 F$ comparações para se obter o menor item

Considerações Práticas

As operações de entrada e saída de dados devem ser implementadas de maneira eficiente pois serão usadas constantemente. Deve-se procurar realizar a leitura, escrita e o processamento interno dos dados de forma simultânea

Os computadores de maior porte possuem uma ou mais unidades independentes para o processamento de entrada e saída. Assim, pode-se realizar o processamento de operações de entrada e saída simultaneamente

Devemos tomar cuidados na escolha da ordem de intercalação F

- **Fitas Magnéticas**:
  - O F deve ser igual ao número de unidades de fitas disponíveis menos um, já que a fase de intercalação pode usar F fitas de entrada e uma fita de saída
  - O número de fitas de entrada deve ser no mínimo dois
- **Discos Magnéticos**:
  - Mesmo raciocínio, porém o acesso direto é menos eficiente que o sequencial

Sedegwick, após o experimento realizado em 1988, sugere considerar o F grande o suficiente para completar a ordenação em poucos passos

- Porém, a melhor escolha para F depende de parâmetros relacionados com o sistema de computação disponível

Intercalação Polifásica

A intercalação balanceada 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 $F + 1$

Observação: Existe um custo de uma cópia adicional do arquivo

Por resolvermos estes problemas, podemos utilizar a intercalação polifásica

Processo de funcionamento da Intercalação Polifásica

- Os blocos previamente ordenados serão distribuídos de forma desigual entre as fitas que se encontram disponíveis (**Exceto** uma fita)
- Após isto, a intercalação de blocos ordenados é executada até que uma das fitas de entrada se esvazie.
- Ao encontrar-se vazia, a fita vazia torna-se a próxima fita de entrada

📌 Se os blocos forem distribuídos de maneira homogênea, temos maior chance de não termos blocos em fitas separadas para que se realize a intercalação

Exemplo do funcionamento da Intercalação Polifásica

- 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:
```

📌 **Observações**:
- A intercalação é realizada em muitas fases
- As fases não envolvem todos os blocos
- Nenhuma cópia direta entre as fitas é realizada

O processo está **encerrado** quando temos apenas um único bloco (completamente ordenado) em uma única fita

Implementação da Intercalação Polifásica

A implementação da intercalação polifásica é simples, sua parte mais delicada está na distribuição inicial que envolve os blocos ordenados entre as fitas

No exemplo apresentado, a distribuição dos blocos nas diversas etapas pode ser representada da seguinte forma:

fita1	fita2	fita3	total
3	2	0	5
1	0	2	3
0	1	1	2
1	0	0	1

Análise da Intercalação Polifásica

- 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 Externo

Proposto em 1980 por Monard, o **Quicksort Externo é um método de ordenação externa** baseado no método Quicksort Interno (em memória principal). Utiliza do paradigma de divisão e conquista

O algoritmo realiza a ordenação *in situ* em um arquivo (ou seja, é realizada a ordenação diretamente dentro do arquivo). Os registros encontram-se armazenados consecutivamente em memória secundária de acesso randômico

O algoritmo utiliza somente $O(\log n)$ unidades de memória interna, não sendo necessária alguma memória externa adicional .

Passo a Passo Geral

Para ordenar o arquivo $A = \{R_1, \dots, R_n\}$, o algoritmo irá realizar os seguintes passos:

- Particionar A de acordo com o esquema:
  - $\{R_1, \dots, R_i\} \leq R_{i+1} \leq Ri + 2 \leq \dots \leq R_{j-2} \leq R_{j-1} \leq \{R_j, \dots, R_n\}$
- Chama recursivamente o algoritmo em cada um dos sub arquivos gerados:
  - $A = \{R_1, \dots, R_i\}$ e $A_2 = \{R_j, \dots, R_n\}$

Os registros ordenados $\{R_{i+1}, \dots, R_{j-1}\}$ correspondem ao pivô do algoritmo, encontrando-se na memória interna durante a execução do mesmo. Os subarquivos A_1 e A_2 contêm os registros menores que R_{i+1} e maiores que R_{j-1}, respectivamente

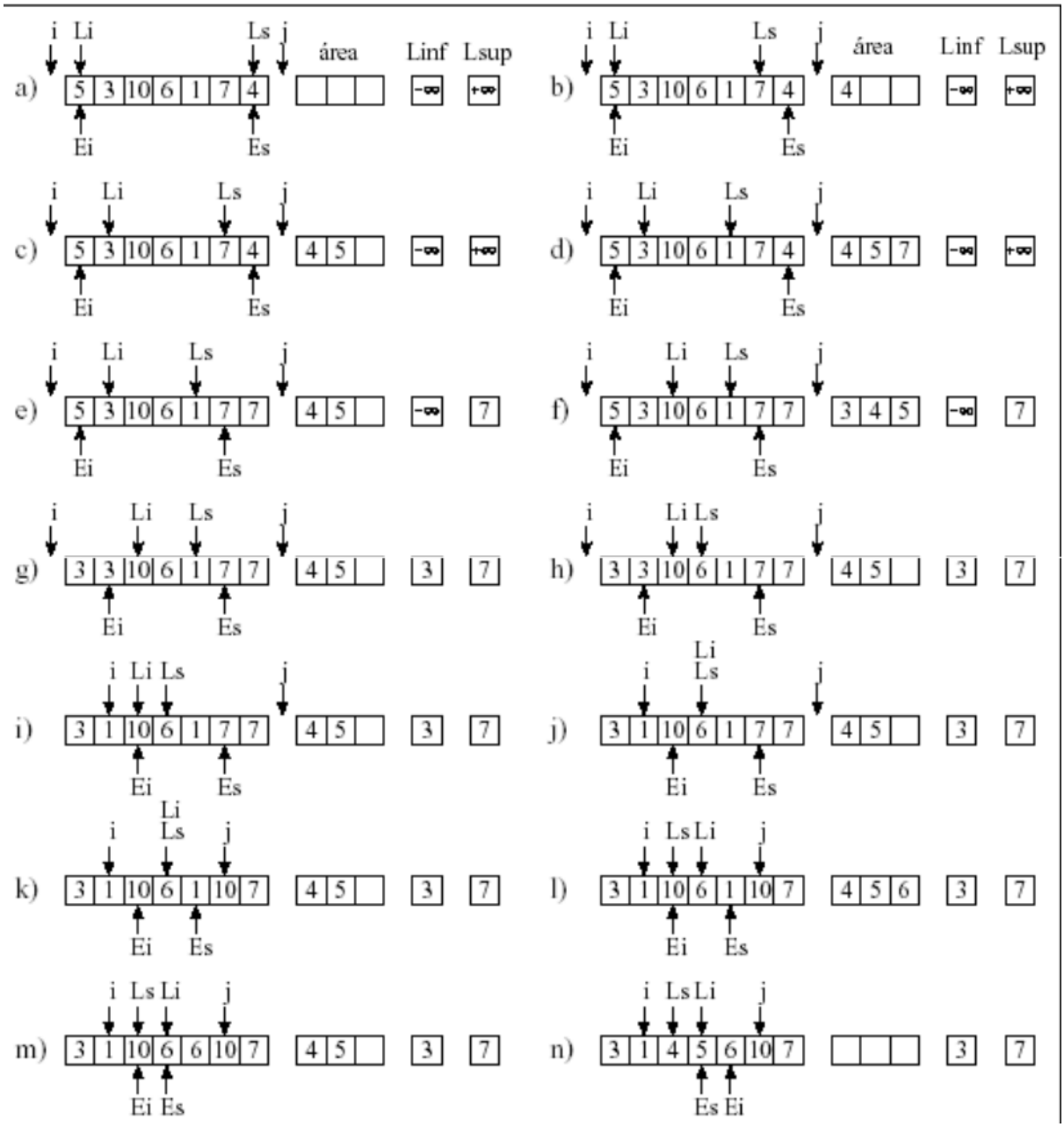
Para a partição do arquivo, será utilizada uma área T de memória interna para o armazenamento do pivô

- $T = j - i - 1$ com $T \geq 3$

Nas chamadas recursivas, deve-se considerar que

- Deve ser ordenado, inicialmente, o sub arquivo de menor tamanho
- Os subarquivos vazios ou com um único registro são ignorados
- Caso o arquivo de entrada A possua no máximo $(j - i - 1)$ registros, ele é ordenado em um único passo

Ilustração do Passo a Passo



- Li = Leitura Inferior (somente lê)
- Ei = Escrita Inferior (somente escreve)
- Ls = Leitura Superior (somente lê)
- Es = Escrita Superior (somente escreve)

Processo de Partição

Os primeiros $TamArea - 1$ registros são lidos, de maneira alternada, dos extremos de A e armazenados na memória interna

- Ao ler $(TamArea - ésimo)$ registro, cuja chave é C
  - C é comparada com L_{sup} e, sendo maior, J recebe E_s e o registro é escrito em A_2
  - Caso contrário, C é comparada com L_{inf} e, sendo menor, I recebe E_I e o registro é escrito em A_1
  - Caso contrário, onde$L_{inf} \leq C \leq L_{sup}$, o registro é inserido na memória interna
- 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 $L_s = E_s$
- Nenhum registro pode ser destruído durante a ordenação *in situ*

Quando a área da memória enche, deve-se remover um registro da mesma, considerando os tamanhos atuais dos dois sub arquivos (A_1 e A_2)

- Sendo Esq e Dir a primeira e a última posição respectivamente de A, os tamanhos de A_1 e A_2 são, respectivamente, ($T_1 = E_i - Esq$) e ($T_2 = Dir - E_s$)
- Se $T_1 < T_2$, o registro de menor chave será removida da memória principal e escrito em $E_i(A_1)$ eL_{inf} é atualizado com este chave
- Se $T_2 \leq T_1$, o registro de maior chave é removido da memória sendo escrito em $E_s(A_2)$ é atualizado com tal chave

Realizando este processo, dividimos o arquivo de forma uniforme e facilitamos o balanceamento da árvore, dessa forma reduzindo a quantidade de operações de leitura e escrita realizadas pelo algoritmo

- O processo de partição continua até que $L_s < L_i$
- Neste momento, os registros armazenados na memória interna devem ser copiados (já se encontrando ordenados) em A

Enquanto existir registros na área de memória, o menor deles é removido e escrito na posição indicada por E_i em A

Código do Quicksort Externo

```
void QuicksortExterno(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs,
                      int Esq, int Dir)
{ int i, j;
  TipoArea Area;    /* Area de armazenamento interna*/
  if (Dir - Esq < 1) return;
  FAVazia(&Area);
  Particao(ArqLi, ArqEi, ArqLEs, Area, Esq, Dir, &i, &j);
  if (i - Esq < Dir - j)
  { /* ordene primeiro o subarquivo menor */
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
  }
  else
  { QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
  }
}

void LeSup(FILE **ArqLEs,TipoRegistro *UltLido,int *Ls,short *OndeLer)
{ fseek(*ArqLEs, (*Ls - 1) * sizeof(TipoRegistro),SEEK_SET );
  fread(UltLido, sizeof(TipoRegistro), 1, *ArqLEs);
  (*Ls)--; *OndeLer = FALSE;
}

void LeInf(FILE **ArqLi,TipoRegistro *UltLido,int *Li,short *OndeLer)
{ fread(UltLido, sizeof(TipoRegistro), 1, *ArqLi);
  (*Li)++; *OndeLer = TRUE;
}

void InserirArea(TipoArea *Area, TipoRegistro *UltLido, int *NRArea)
{ /*Insere UltLido de forma ordenada na Area*/
  InsereItem(*UltLido, Area);  *NRArea = ObterNumCelOcupadas(Area);
}
```

```
void EscreveMax(FILE **ArqLEs, TipoRegistro R, int *Es)
{ fseek(*ArqLEs, (*Es - 1) * sizeof(TipoRegistro),SEEK_SET );
  fwrite(&R, sizeof(TipoRegistro), 1, *ArqLEs);  (*Es)--;
}

void EscreveMin(FILE **ArqEi, TipoRegistro R, int *Ei)
{ fwrite(&R, sizeof(TipoRegistro), 1, *ArqEi);  (*Ei)++; }

void RetiraMax(TipoArea *Area, TipoRegistro *R, int *NRArea)
{ RetiraUltimo(Area, R);  *NRArea = ObterNumCelOcupadas(Area); }

void RetiraMin(TipoArea *Area, TipoRegistro *R, int *NRArea)
{ RetiraPrimeiro(Area, R);  *NRArea = ObterNumCelOcupadas(Area); }
```

```
void Particao(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs,
              TipoArea Area, int Esq, int Dir, int *i, int *j)
{ int Ls = Dir, Es = Dir, Li = Esq, Ei = Esq,
      NRArea = 0, Linf = INT_MIN, Lsup = INT_MAX;
  short OndeLer = TRUE;  TipoRegistro UltLido, R;
  fseek (*ArqLi, (Li - 1)* sizeof(TipoRegistro), SEEK_SET );
  fseek (*ArqEi, (Ei - 1)* sizeof(TipoRegistro), SEEK_SET );
  *i = Esq - 1;  *j = Dir + 1;
  while (Ls >= Li)
    { if (NRArea < TAMAREA - 1)
      { if (OndeLer)
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
        else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        InserirArea(&Area, &UltLido, &NRArea);
        continue;
      }
      if (Ls == Es)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else if (Li == Ei) LeInf(ArqLi, &UltLido, &Li, &OndeLer);
           else if (OndeLer) LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
                else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

```
            if (UltLido.Chave > Lsup)
            { *j = Es; EscreveMax(ArqLEs, UltLido, &Es);
              continue;
            }
            if (UltLido.Chave < Linf)
            { *i = Ei; EscreveMin(ArqEi, UltLido, &Ei);
              continue;
            }
            InserirArea(&Area, &UltLido, &NRArea);
            if (Ei - Esq < Dir - Es)
            { RetiraMin(&Area, &R, &NRArea);
              EscreveMin(ArqEi, R, &Ei); Linf = R.Chave;
            }
            else { RetiraMax(&Area, &R, &NRArea);
                   EscreveMax(ArqLEs, R, &Es); Lsup = R.Chave;
                 }
          }
        while (Ei <= Es)
          { RetiraMin(&Area, &R, &NRArea);
            EscreveMin(ArqEi, R, &Ei);
          }
      }

typedef int TipoApontador;
/*--Entra aqui o Programa C.23--*/
typedef TipoItem TipoRegistro;
/*Declaracao dos tipos utilizados pelo quicksort externo*/
FILE *ArqLEs;    /* Gerencia o Ls e o Es */
FILE *ArqLi;     /* Gerencia o Li */
FILE *ArqEi;     /* Gerencia o Ei */
TipoItem R;
/*--Entram aqui os Programas J.4, D.26, D.27 e D.28--*/
int main(int argc, char *argv[])
{ ArqLi = fopen ("teste.dat", "wb");
  if(ArqLi == NULL){printf("Arquivo nao pode ser aberto\n"); exit(1);}
  R.Chave = 5;  fwrite(&R, sizeof(TipoRegistro), 1, ArqLi);
  R.Chave = 3;  fwrite(&R, sizeof(TipoRegistro), 1, ArqLi);
  R.Chave = 10; fwrite(&R, sizeof(TipoRegistro), 1, ArqLi);
  R.Chave = 6;  fwrite(&R, sizeof(TipoRegistro), 1, ArqLi);
  R.Chave = 1;  fwrite(&R, sizeof(TipoRegistro), 1, ArqLi);
  R.Chave = 7;  fwrite(&R, sizeof(TipoRegistro), 1, ArqLi);
  R.Chave = 4;  fwrite(&R, sizeof(TipoRegistro), 1, ArqLi);
  fclose(ArqLi);
```

```
  ArqLi = fopen ("teste.dat", "r+b");
  if (ArqLi == NULL){printf("Arquivo nao pode ser aberto\n"); exit(1);}
  ArqEi = fopen ("teste.dat", "r+b");
  if (ArqEi == NULL){printf("Arquivo nao pode ser aberto\n"); exit(1);}
  ArqLEs = fopen ("teste.dat", "r+b");
  if (ArqLEs == NULL) {printf("Arquivo nao pode ser aberto\n"); exit(1);}
  QuicksortExterno(&ArqLi, &ArqEi, &ArqLEs, 1, 7);
  fflush(ArqLi); fclose(ArqEi); fclose(ArqLEs); fseek(ArqLi,0, SEEK_SET);
  while(fread(&R, sizeof(TipoRegistro), 1, ArqLi)) { printf("Registro=%d\n", R.Chave);}
  fclose(ArqLi); return 0;
}
```

Análise do Quicksort Externo

Consideremos N o número de registros a serem ordenados e B o tamanho do bloco de leitura ou gravação do sistema operacional. Dessa forma, temos:

- **Melhor Caso:** $O(\frac{N}{B})$
  - Ocorre quando o arquivo de entrada já está ordenado
- **Pior Caso**: $O(\frac{N^2}{TamArea})$
  - Ocorre quando as partições geradas possuem tamanhos inadequados, ou seja, o maior tamanho possível e vazio
  - A medida que N cresce, a probabilidade de ocorrência deste pior caso tende a zero
- **Caso Médio**: $O(\frac{N}{B} * \log(\frac{N}{TamArea}))$
  - Este é o caso com maior probabilidade de ocorrência