Ordenação Externa

Estrutura de Dados II (BCC203)

Prof. Guilherme Tavares de Assis

Universidade Federal de Ouro Preto – UFOP

Instituto de Ciências Exatas e Biológicas – ICEB

Departamento de Computação – DECOM

Ordenação Externa

- A ordenação externa consiste em ordenar arquivos de tamanho maior que a memória interna disponível.
- Os métodos de ordenação externa são diferentes dos de ordenação interna.
  - Os algoritmos devem diminuir o número de acesso às unidades de memória externa.
- Nas memórias externas, os dados ficam em um arquivo sequencial.
  - Apenas um registro pode ser acessado em um dado momento: restrição forte comparada com as possibilidades de acesso em um vetor.

Ordenação Externa

- Fatores que determinam as diferenças das técnicas de ordenação externa em relação à ordenação interna:
  - Custo de acesso a memória secundária é muito maior do que o acesso a memória primária.
    - O custo principal na ordenação externa está relacionado à transferência de dados entre a memória interna e externa.
  - 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 estado atual da tecnologia.

Ordenação Externa

- O método mais importante de ordenação externa é o de ordenação por intercalação.
  - Intercalar significa combinar dois ou mais blocos ordenados em um único bloco ordenado.
  - A intercalação é utilizada como uma operação auxiliar na ordenação.

- O foco dos algoritmos para ordenação externa é reduzir o número de passadas sobre o arquivo.
  - Uma boa medida de complexidade de um algoritmo de ordenação por intercalação é o número de vezes que um item é lido ou escrito na memória interna.
  - Os bons métodos de ordenação externa geralmente envolvem menos do que dez passadas sobre o arquivo.

Ordenação Externa

- Estratégia geral dos métodos de ordenação externa:
  1. Quebre o arquivo em blocos do tamanho da memória interna disponível.
  2. Ordene cada bloco na memória interna.
  3. Intercale os blocos ordenados, fazendo várias passadas sobre o arquivo.
     - A cada passada são criados blocos ordenados cada vez maiores, até que todo o arquivo esteja ordenado.

Intercalação Balanceada de Vários Caminhos

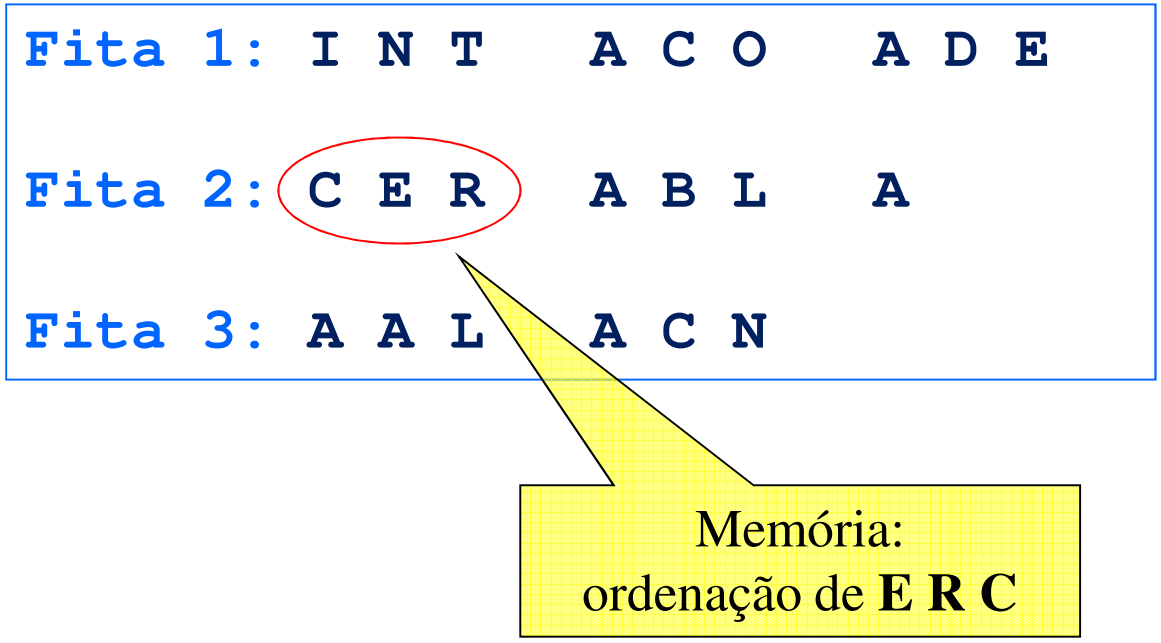
- Para exemplificação, ordene um arquivo, armazenado em uma fita de entrada, que possui os seguintes 22 registros:

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

- Considerações:
  - memória interna com capacidade para três itens;
  - seis unidades disponíveis de fita magnética.

Intercalação Balanceada de Vários Caminhos

- Fase de criação dos blocos ordenados, envolvendo:
  - quebra do arquivo em blocos do tamanho da memória interna disponível;
  - ordenação de cada bloco na memória interna.



Intercalação Balanceada de Vários Caminhos

- Fase de intercalação, envolvendo os passos:
  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.
     - 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.

Intercalação Balanceada de Vários Caminhos

- Resultado da 1ª passada sobre o arquivo na fase de intercalação:

```
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 de intercalação nos blocos formados após a 1ª passada sobre o arquivo, tem-se o arquivo ordenado (2ª passada sobre o mesmo):

```
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:
```

Intercalação Balanceada de Vários Caminhos

- Quantas passadas são necessárias para ordenar um arquivo de tamanho arbitrário?
  - Seja **n** o número de registros do arquivo.
  - Seja **m** o número de palavras possíveis na memória interna, sendo uma palavra igual a um registro do arquivo.

  Logo, a primeira fase produz **n/m** blocos ordenados.

  - Seja P(n) o número de passadas na fase de intercalação.
  - Seja f o número de fitas utilizadas em cada passada.

  Logo, $\mathbf{P(n) = \log_f (n/m)}$

  - No exemplo apresentado, tem-se:

$$P(n) = \log_3 (22/3) \approx 2$$

Intercalação Balanceada de Vários Caminhos

- No 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 saída.
  - Redistribua estes blocos entre as fitas de onde eles foram lidos.
- No exemplo, apenas quatro fitas seriam suficientes.
  - A intercalação dos blocos a partir das fitas 1, 2 e 3 seria dirigida para a fita 4.
  - Ao final, o segundo e o terceiro blocos ordenados de nove registros seriam transferidos de volta para as fitas 1 e 2.
- Custo envolvido: uma passada a mais em cada intercalação.

Implementação por meio de Substituição por Seleção

- A implementação do método de intercalação balanceada pode ser feita utilizando filas de prioridades.
  - As duas fases do método (quebra do arquivo e intercalação) podem ser implementadas de forma eficiente e elegante.
  - Substitui o menor item existente na memória interna pelo próximo item da fita de entrada.
- Estrutura ideal para implementação: *heap*.
- Operação de substituição na memória interna:
  - Retirar o menor item da fila de prioridades.
  - Colocar um novo item no seu lugar.
  - Reconstituir a propriedade do *heap*.

Implementação por meio de Substituição por Seleção

- Processo de funcionamento para geração dos blocos ordenados:
  - Inicialmente, **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.
  - Quando um item marcado vai para o topo da fila, o bloco corrente é encerrado e um novo bloco ordenado é iniciado.

Implementação por meio de Substituição por Seleção

Item marcado (vermelho)

Ent.	1	2	3
E	I	N	T
R	N	E	T
C	R	E	T
A	T	E	C
L	A	E	C
A	C	E	L
C	E	A	L
A	L	A	C
O	A	A	C
B	A	O	C
A	B	O	C

Ent.	1	2	3
L	C	O	A
A	L	O	A
N	O	A	A
C	A	N	A
E	A	N	C
A	C	N	E
D	E	N	A
A	N	D	A
	A	D	A
	A	D	
	D		

Blocos ordenados gerados

Bloco 1: I N R T
Bloco 2: A C E L
Bloco 3: A A B C L O
Bloco 4: A A C E N
Bloco 5: A A D

Implementação por meio de Substituição por Seleção

- Após gerados os blocos ordenados, faz-se a intercalação dos mesmos utilizando fila de prioridades.
  - Monte uma fila de prioridades de tamanho **f** a partir dos primeiros itens de cada um dos **f** blocos ordenados.
  - Repita o processo abaixo até que não haja mais itens nos blocos ordenados:
    - Substitua o item do topo da fila de prioridades, escrevendo-o em uma fita de saída, pelo próximo item do mesmo bloco do item que está sendo substituído.
    - Reconstitua a propriedade da fila de prioridades.

Implementação por meio de Substituição por Seleção

- Para valores pequenos de **f**, não é vantajoso utilizar seleção por substituição para intercalar blocos, já que o menor item pode ser obtido por **f-1** comparações.

- Quando f $\geq$ 8, o método é considerado adequado, realizando $\mathbf{\log_2 f}$ comparações para se obter o menor item.

Considerações Práticas

- As operações de entrada e saída de dados devem ser implementadas eficientemente.
  - Deve-se procurar realizar a leitura, a escrita e o processamento interno dos dados de forma simultânea.
- Os computadores de maior porte possuem uma ou mais unidades independentes para processamento de entrada e saída.
  - Assim, pode-se realizar processamento e operações de E/S simultaneamente.

Considerações Práticas

- Escolha da ordem de intercalação **f**:
  - Fitas magnéticas:
    - **f** deve ser igual ao número de unidades de fita 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 apresentado, porém o acesso direto é menos eficiente que o sequencial.
- *Sedegwick* (1988) sugere considerar **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

- Problemas com a intercalação balanceada de vários caminhos:
  - 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.
- Solução: intercalação polifásica.

Intercalação Polifásica

- 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 até que uma das fitas de entrada se esvazie.
  - A fita vazia torna-se a próxima fita de saída.

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 Polifásica

- 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 fitas é realizada.

Intercalação Polifásica

- A implementação da intercalação polifásica é simples.
- A parte mais delicada está na distribuição inicial dos blocos ordenados entre as fitas.
- No exemplo apresentado, a distribuição dos blocos nas diversas etapas foi:

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

Intercalação Polifásica

- 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 Externo

- *Quicksort* externo foi proposto por Monard em 1980.
- O algoritmo utiliza o paradigma de divisão e conquista.
- O algoritmo ordena *in situ* um arquivo $A = \{R_1, \ldots, R_n\}$ de **n** registros.
  - 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.

Quicksort Externo

- Para ordenar o arquivo $A = \{R_1, \ldots, R_n\}$, o algoritmo:
  - particiona A da seguinte forma:

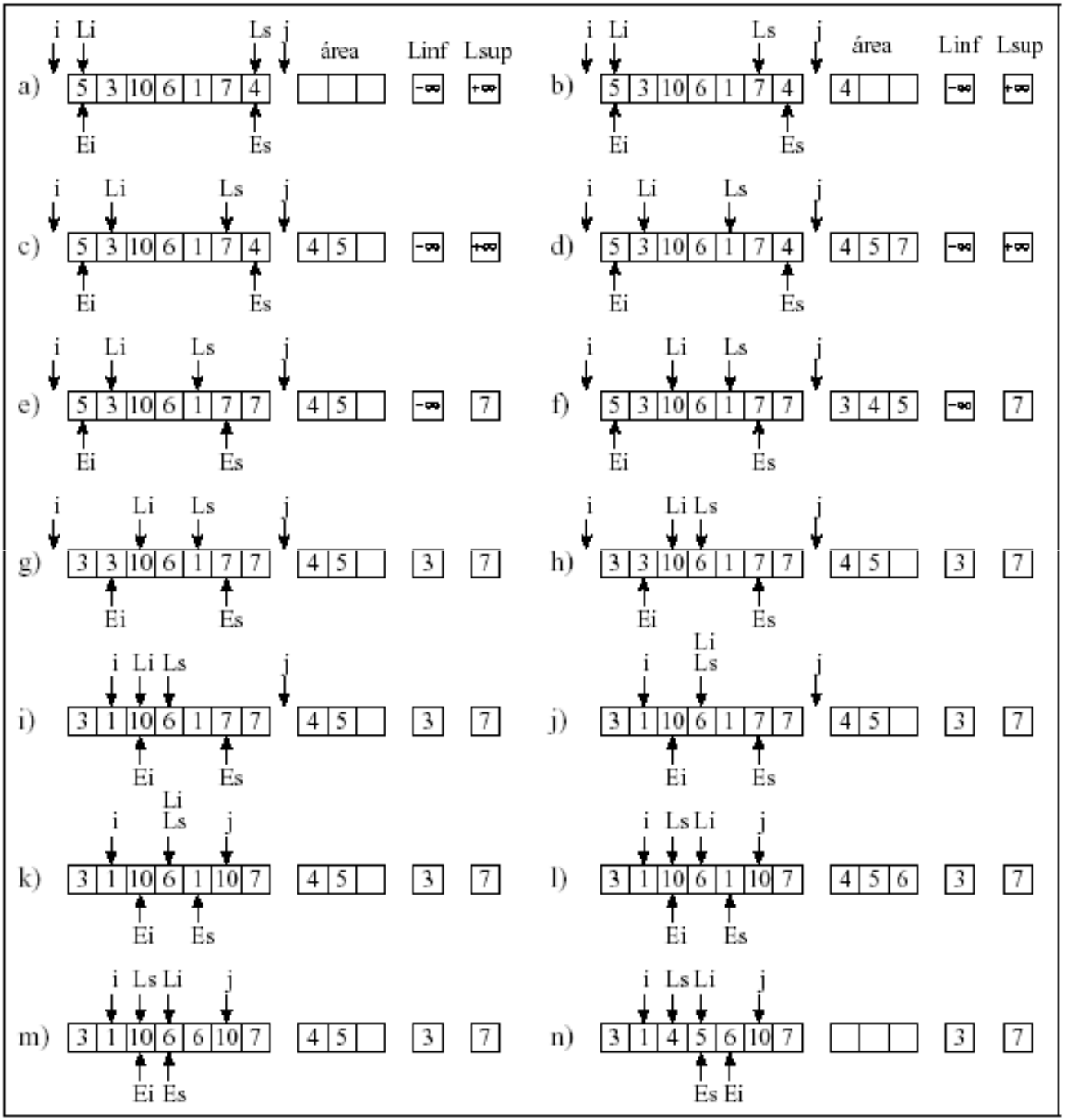
    $\{R_1, \ldots, R_i\} \leq \mathbf{R_{i+1} \leq R_{i+2} \leq \ldots \leq R_{j-2} \leq R_{j-1}} \leq \{R_j, \ldots, R_n\}$;
  - chama recursivamente o algoritmo em cada um dos subarquivos gerados:

    $A_1 = \{R_1, \ldots, R_i\}$ e $A_2 = \{R_j, \ldots, R_n\}$.

- Os registros ordenados $\{R_{i+1}, \ldots, R_{j-1}\}$ 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.

Quicksort Externo

- Para a partição do arquivo, é utilizada uma área de memória interna para o armazenamento do pivô.
  - Tamanho da área = j – i – 1, sendo necessariamente ≥ 3.
- Nas chamadas recursivas, deve-se considerar que:
  - deve ser ordenado, inicialmente, o subarquivo de menor tamanho;
  - 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.


a)
b)
c)
d)
e)
f)
g)
h)
i)
j)
k)
l)
m)
n)
i
Li
Ls
j
Ei
Es
área
Linf
Lsup
-∞
+∞

Quicksort Externo: Partição

- Considerações sobre o funcionamento de uma partição:
  - Os valores das chaves de $\mathbf{R_i}$ e $\mathbf{R_j}$ são denominados limite inferior (**Linf**) e limite superior (**Lsup**) da partição.
    - Tais limites são inicializados com os valores $-\infty$ e $+\infty$.
  - A leitura de **A** é controlada por apontadores de leitura inferior (**Li**) e superior (**Ls**).
    - A cada leitura no extremo inferior, **Li** é incrementado; a cada leitura no extremo superior, **Ls** é decrementado.
  - A escrita em **A** é controlada por apontadores de escrita inferior (**Ei**) e superior (**Es**).
    - A cada escrita no extremo inferior, **Ei** é incrementado; a cada escrita no extremo superior, **Es** é decrementado.

Quicksort Externo: Partição

- Os primeiros "tamanho da área" (**TamArea**) - 1 registros são lidos, alternativamente, dos extremos de **A** e armazenados na área de memória interna.
- Ao ler o TamArea-ésimo registro, cuja chave é **C**:
  - **C** é comparada com **Lsup** e, sendo maior, **j** recebe **Es** e o registro é escrito em $\mathbf{A_2}$;
  - caso contrário, **C** é comparada com **Linf** e, sendo menor, **i** recebe **Ei** e o registro é escrito em $\mathbf{A_1}$;
  - Caso contrário (**Linf** $\leq$ **C** $\leq$ **Lsup**), o registro é inserido na área de memória interna.
- Para garantir que os apontadores de escrita estejam atrás dos apontadores de leitura, a ordem alternada de leitura é interrompida se (**Li = Ei**) ou (**Ls = Es**).
  - Nenhum registro pode ser destruído durante a ordenação *in situ*.

Quicksort Externo: Partição

- Quando a área de memória enche, deve-se remover um registro da mesma, considerando os tamanhos atuais de $\mathbf{A_1}$ e $\mathbf{A_2}$.
  - Sendo **Esq** e **Dir** a 1ª e a última posição de **A**, os tamanhos de $\mathbf{A_1}$ e $\mathbf{A_2}$ são, respectivamente, ($\mathbf{T_1 = Ei – Esq}$) e ($\mathbf{T_2 = Dir – Es}$).
  - Se (**T1 < T2**), o registro de menor chave é removido da memória, sendo escrito em **Ei** ($\mathbf{A_1}$), e **Linf** é atualizado com tal chave.
  - Se (**T2 ≤ T1**), o registro de maior chave é removido da memória, sendo escrito em **Es** ($\mathbf{A_2}$), e **Lsup** é atualizado com tal chave.
- O objetivo é escrever o registro removido da memória no subarquivo de menor tamanho, no intuito de dividir **A** de forma uniforme e, assim, balancear a árvore gerada pelas recursões.
  - Isso minimiza a quantidade de operações de leitura e escrita efetuadas pelo algoritmo.

Quicksort Externo: Partição

- 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, o menor deles é removido e escrito na posição indicada por **Ei** em **A**.

Quicksort Externo: Programa

```
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);
  }
}
```

Quicksort Externo: Programa

```
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);
}
```

Quicksort Externo: Programa

```
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); }
```

Quicksort Externo: Programa

```
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);
```

Quicksort Externo: Programa

```
            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);
          }
      }
```

Quicksort Externo: Programa

```
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);
```

Quicksort Externo: Programa

```
  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;
}
```

Quicksort Externo: 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: $O\ (n^2 / TamArea\ )$
  - 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: $O\ (n / b \times log\ (n / TamArea)\ )$
  - Maior probabilidade de ocorrer.