Processamento Cosequencial e Ordenação de Arquivos Grandes

Estrutura de Dados II

Ordenação de Arquivos em Disco (cont.)

QuickSort Interno

- Ideia
  - Dividir o problema de ordenar um conjunto com *n* itens em dois problemas menores
    - Os problemas menores são ordenados independentemente e depois os resultados são combinados para produzir a solução do problema maior

QuickSort Interno

QuickSort(Esq, Dir) **// QuickSort(1,n) //primeira chamada**

Particao(Esq, Dir, i, j)

If (Esq < j) QuickSort(Esq, j)

If (i < Dir) QuickSort(i, Dir)

Particao

```
*i = Esq;  *j = Dir;
x = A[(*i + *j) / 2]; /* obtem o pivo x */
do
{ while (x.Chave > A[*i].Chave) (*i)++;
  while (x.Chave < A[*j].Chave) (*j)--;
  if (*i <= *j)
  { w = A[*i]; A[*i] = A[*j]; A[*j] = w;
    (*i)++; (*j)--;
  }
} while (*i <= *j);
```

enquanto está no lugar correto anda

se ainda não cruzou

enquanto não cruzar

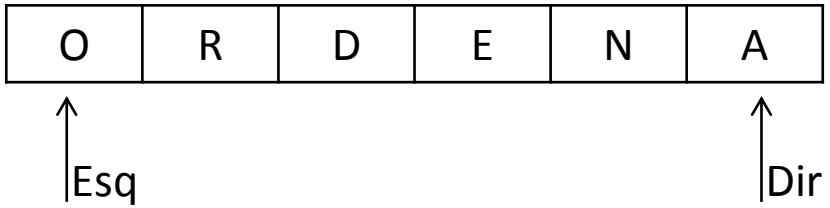
QuickSort Interno

QuickSort(Esq, Dir) [1, 6]

Particao(Esq, Dir, i, j) [1, 6, ?, ?]

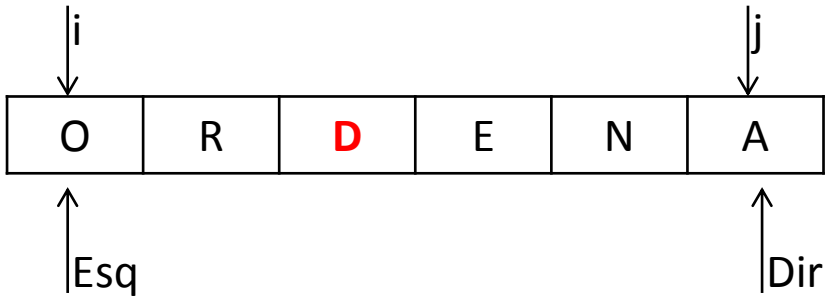
If (Esq < j) QuickSort(Esq, j)

If (i < Dir) QuickSort(i, Dir)



QuickSort Interno

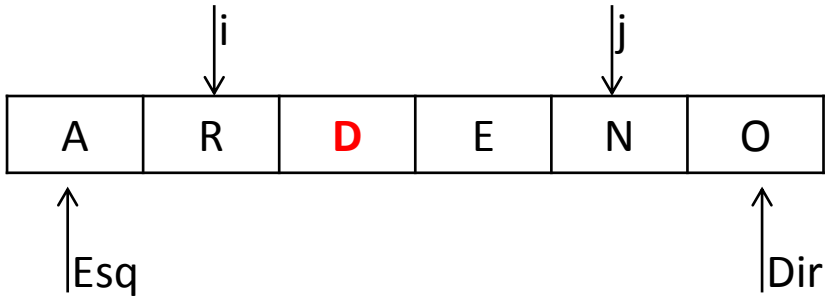
(a)



pivô = x = D
(1+6)/2 = 3

i=1
j=6

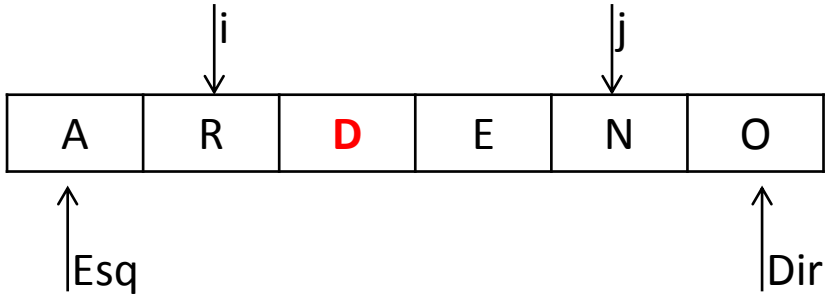
(b)




```
*i = Esq;  *j = Dir;
x = A[(*i + *j) / 2]; /* obtem o pivo x */
do
   { while (x.Chave > A[*i].Chave) (*i)++;
     while (x.Chave < A[*j].Chave) (*j)--;
     if (*i <= *j)
     { w = A[*i]; A[*i] = A[*j]; A[*j] = w;
       (*i)++; (*j)--;
     }
   } while (*i <= *j);
```

QuickSort Interno

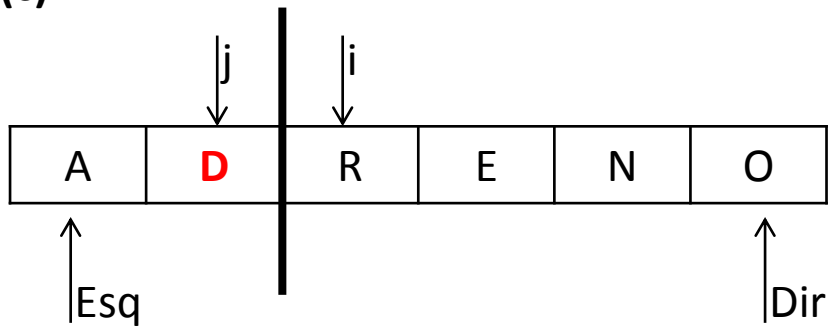
(b)



pivô = x = D
(1+6)/2 = 3

i=1
j=6

(c)




```
*i = Esq;  *j = Dir;
x = A[(*i + *j) / 2]; /* obtem o pivo x */
do
   { while (x.Chave > A[*i].Chave) (*i)++;
     while (x.Chave < A[*j].Chave) (*j)--;
     if (*i <= *j)
     { w = A[*i]; A[*i] = A[*j]; A[*j] = w;
       (*i)++; (*j)--;
     }
   } while (*i <= *j);
```

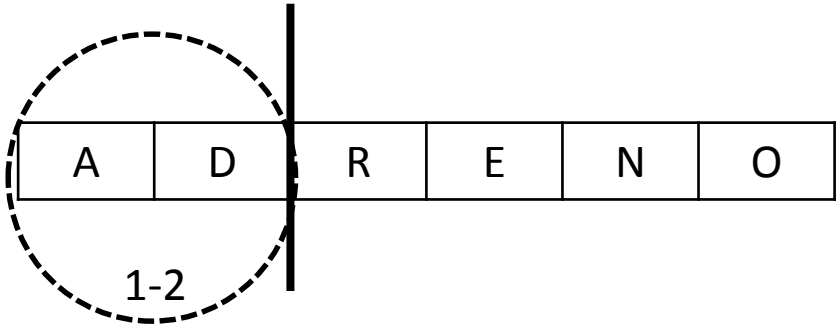
QuickSort Interno

QuickSort(Esq, Dir) [1, 6]

Particao(Esq, Dir, i, j) [1, 6, 3, 2]

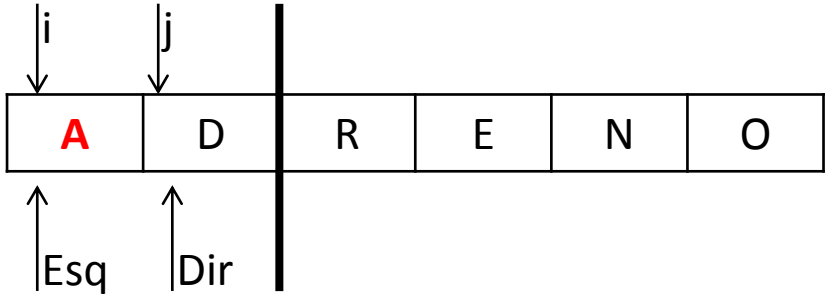
If (Esq < j) QuickSort(Esq, j) [1, 2]

If (i < Dir) QuickSort(i, Dir) [3, 6]



QuickSort Interno

(a)



pivô = x = A
(1+2)/2 = 1

i=1
j=2

(b)


```
*i = Esq;  *j = Dir;
x = A[(*i + *j) / 2]; /* obtem o pivo x */
do
   { while (x.Chave > A[*i].Chave) (*i)++;
     while (x.Chave < A[*j].Chave) (*j)--;
     if (*i <= *j)
     { w = A[*i]; A[*i] = A[*j]; A[*j] = w;
       (*i)++; (*j)--;
     }
   } while (*i <= *j);
```

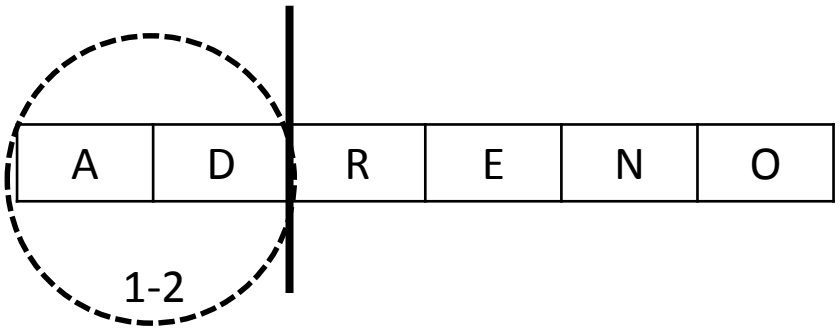
QuickSort Interno

QuickSort(Esq, Dir) [1, 2]

Particao(Esq, Dir, i, j) [1, 2, 2, 0]

If (Esq < j) QuickSort(Esq, j) [1, 0] Falha!!!

If (i < Dir) QuickSort(i, Dir) [2, 2] Falha!!!

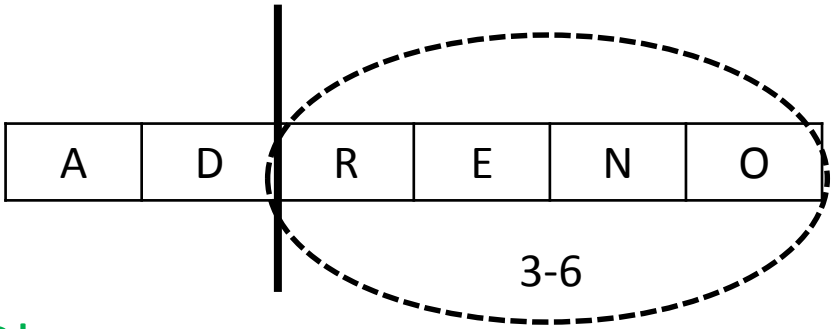


QuickSort(Esq, Dir) [1, 6]

Particao(Esq, Dir, i, j) [1, 6, 3, 2]

If (Esq < j) QuickSort(Esq, j) [1, 2] Ok

If (i < Dir) QuickSort(i, Dir) [3, 6]



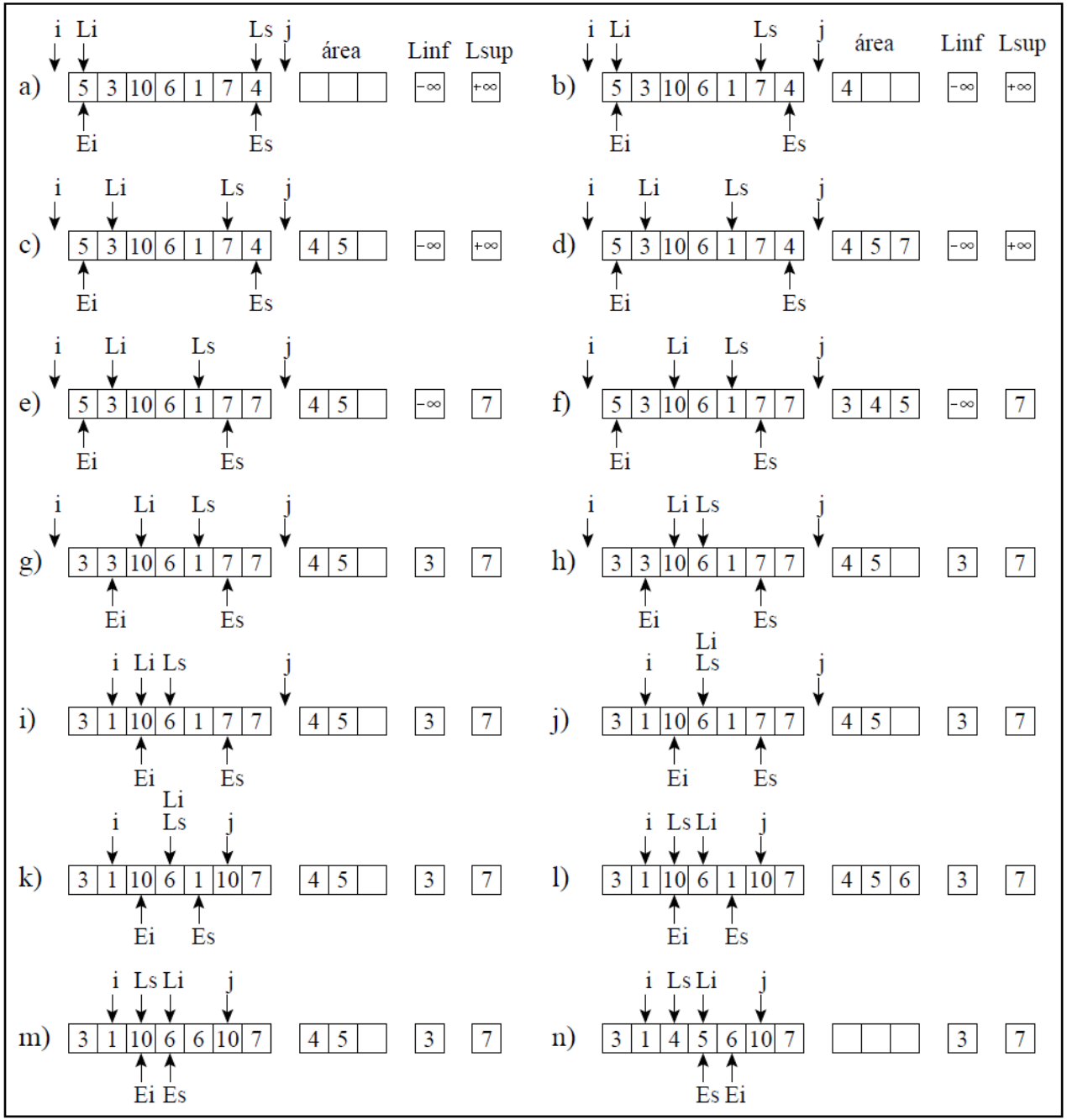
terminar em casa...

QuickSort Externo

- Ordena *in situ* um arquivo A = $\{R_1, ..., R_n\}$ de *n* registros
- O objetivo é particionar A da seguinte maneira: $\{R_1, ..., R_i\} \leq R_{i+1} \leq R_{i+2} \leq ... \leq R_{j-2} \leq R_{j-1} \leq \{R_j, ..., R_n\}$
  - Para tanto, utiliza uma área de armazenamento na memória interna de tamanho TamArea = j – i – 1, com TamArea ≥ 3
  - O algoritmo é chamado recursivamente em cada um dos subarquivos $A_1 = \{R_1, ..., R_i\}$ e $A_2 = \{R_j, ..., R_n\}$

QuickSort Externo: Particao

- Enquanto os ponteiros dos extremos esquerdo e direito não se cruzarem
  - Leia alternadamente os registros dos extremos esquerdo e direito até terem (TamArea-1) registros na área temporária
  - Altere a ordem de leitura dos extremos, caso necessário, e leia o último registro a ser inserido na área temporária
    - Se o último registro lido for maior que o último registro escrito no extremo direito, escreve o registro no extremo direito
    - Se o último registro lido for menor que o último registro escrito no extremo esquerdo, escreve o registro no extremo esquerdo
    - Se o último registro lido estiver entre o último registro escrito no extremo esquerdo e direito, insere-o na área temporária
      - Se a área temporária estiver cheia, retire o menor/maior registro, dependendo do tamanho dos subarquivos dos extremos esquerdo/direito
- Se ainda restarem registros na área temporária, escreva os registros restantes ordenadamente no extremo esquerdo

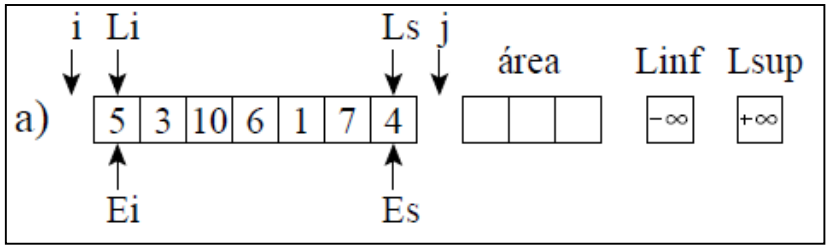

a)
i Li Ls j
5 3 10 6 1 7 4
Ei Es
área
Linf -∞
Lsup +∞
b)
i Li Ls j
5 3 10 6 1 7 4
Ei Es
área 4
Linf -∞
Lsup +∞
c)
i Li Ls j
5 3 10 6 1 7 4
Ei Es
4 5
Linf -∞
Lsup +∞
d)
i Li Ls j
5 3 10 6 1 7 4
Ei Es
4 5 7
Linf -∞
Lsup +∞
e)
i Li Ls j
5 3 10 6 1 7 7
Ei Es
4 5
Linf -∞
Lsup 7
f)
i Li Ls j
5 3 10 6 1 7 7
Ei Es
3 4 5
Linf -∞
Lsup 7
g)
i Li Ls j
3 3 10 6 1 7 7
Ei Es
4 5
Linf 3
Lsup 7
h)
i Li Ls j
3 3 10 6 1 7 7
Ei Es
4 5
Linf 3
Lsup 7
i)
i Li Ls j
3 1 10 6 1 7 7
Ei Es
4 5
Linf 3
Lsup 7
j)
i Li Ls j
3 1 10 6 1 7 7
Ei Es
4 5
Linf 3
Lsup 7
k)
i Li Ls j
3 1 10 6 1 10 7
Ei Es
4 5
Linf 3
Lsup 7
l)
i Ls Li j
3 1 10 6 1 10 7
Ei Es
4 5 6
Linf 3
Lsup 7
m)
i Ls Li j
3 1 10 6 6 10 7
Ei Es
4 5
Linf 3
Lsup 7
n)
i Ls Li j
3 1 4 5 6 10 7
Es Ei
Linf 3
Lsup 7

```
void QuicksortExterno(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs, int Esq, int Dir)
{ int i, j;
  TipoArea Area;    /* Area de armazenamento interna*/

  printf("Entrou QS\n");

  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 */
    printf("Primeiro: %d - %d; %d - %d\n",Esq,i,j,Dir);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
  }
  else
    { printf("Segundo: %d - %d; %d - %d\
      QuicksortExterno(ArqLi, ArqEi, Arq
      QuicksortExterno(ArqLi, ArqEi, Arq
    }
}
```

MAIN()

```
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);
```

```c
//Li = leitura inferior
//Ls = leitura superior
//Ei = escrita inferior
//Es = escrita superior
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;//Começa pelo lado direito
  TipoRegistro UltLido, R;

  printf("Entrou Particao\n");

  //posiciona os ponteiros
  fseek (*ArqLi, (Li - 1)* sizeof(TipoRegistro), SEEK_SET );
  fseek (*ArqEi, (Ei - 1)* sizeof(TipoRegistro), SEEK_SET );

  *i = Esq - 1;
  *j = Dir + 1;
```

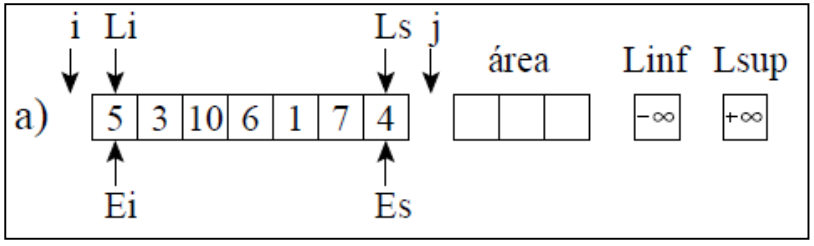


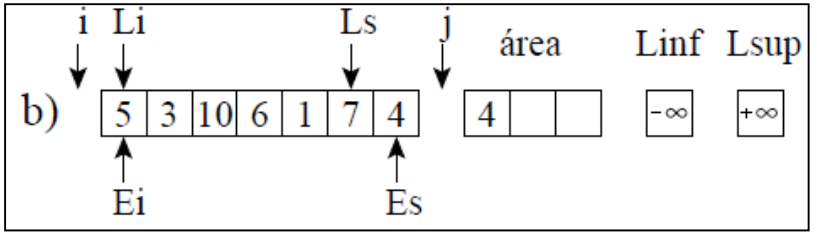
Ls = 7
Es = 7
Li = 1
Ei = 1
NRArea = 0
Linf = -inf*
Lsup = +inf*

i = 0
j = 8

```
while (Ls >= Li)//Enquanto não cruzar
 { //Leitura
   if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
   { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
       LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
     else
       LeInf(ArqLi, &UltLido, &Li, &OndeLer);

     InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
     continue;//volta while, para ler o próximo do arquivo
   }
   //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
   //dos apontadores de leitura
   //faz com que nenhuma informação seja destruída durante a ordenação
   if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
     LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
   else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
               LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
             else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

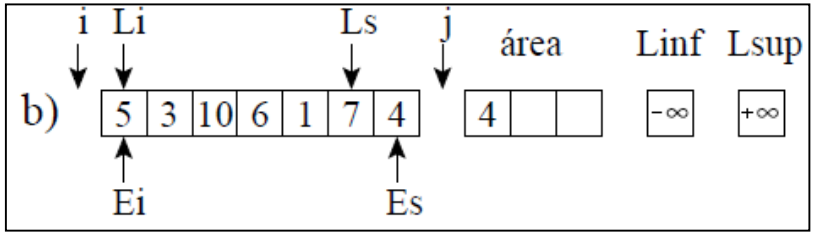
lê direita

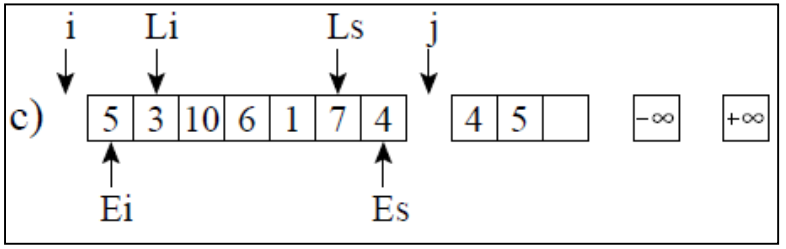




```
while (Ls >= Li)//Enquanto não cruzar
 { //Leitura
   if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
   { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
       LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
     else
       LeInf(ArqLi, &UltLido, &Li, &OndeLer);

     InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
     continue;//volta while, para ler o próximo do arquivo
   }
   //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
   //dos apontadores de leitura
   //faz com que nenhuma informação seja destruída durante a ordenação
   if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
     LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
   else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
               LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
             else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

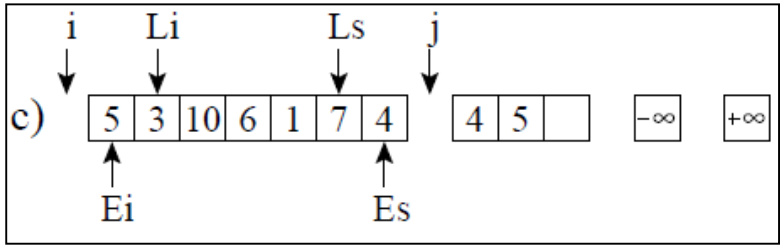
lê esquerda





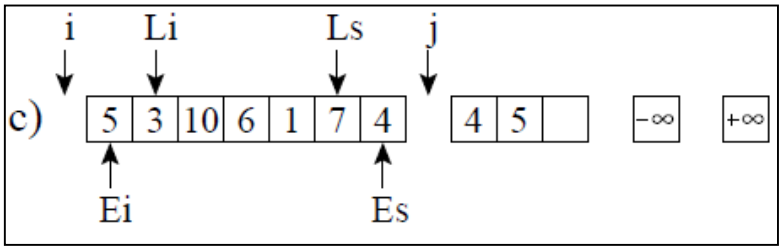
```
while (Ls >= Li)//Enquanto não cruzar
  { //Leitura
    if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
    { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else
        LeInf(ArqLi, &UltLido, &Li, &OndeLer);

      InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
      continue;//volta while, para ler o próximo do arquivo
    }

    //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
    //dos apontadores de leitura
    //faz com que nenhuma informação seja destruída durante a ordenação
    if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
    else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

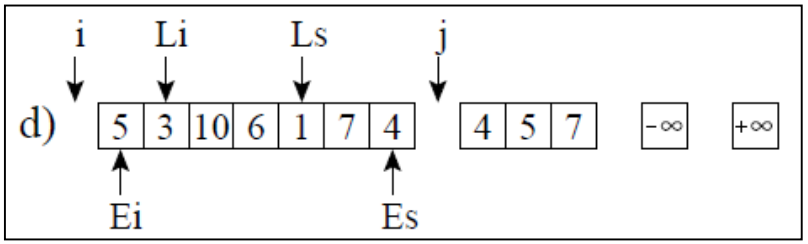
lê direita



```
//Compara a Ultima chave lida com o Lsup
//Se for maior, escreve na posição Es e atualiza "j" (marcador do subarquivo A2)
if (UltLido.Chave > Lsup)
{ *j = Es;
  EscreveMax(ArqLEs, UltLido, &Es);
  continue;//volta while, para ler o próximo do arquivo
}

//Compara a Ultima chave lida com o Linf
//Se for menor, escreve na posição Ei e atualiza "i" (marcador do subarquivo A1)
if (UltLido.Chave < Linf)
{ *i = Ei;
  EscreveMin(ArqEi, UltLido, &Ei);
  continue;//volta while, para ler o próximo do arquivo
}

//Se não for maior nem menor, i.e., está entre os dois, insere na memória temporária
InserirArea(&Area, &UltLido, &NRArea);
```

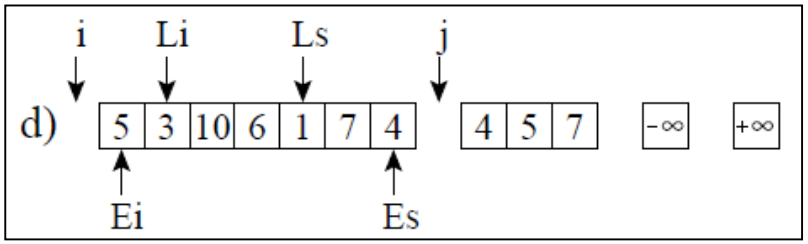
lê direita

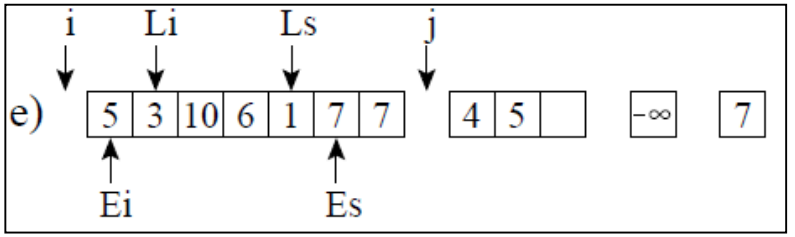




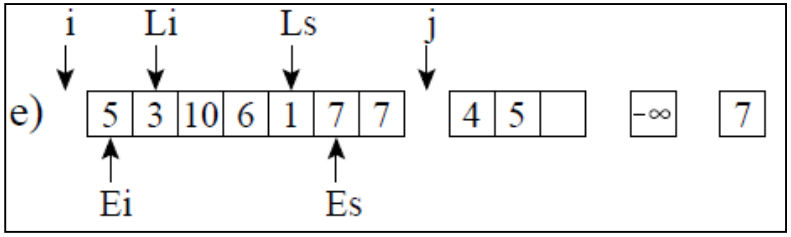
```
  //Quando a área temporária estiver cheia, deve-se remover um elemento da mesma
  //Como decidir qual remover? Considerando o Tam. de A1 e A2 (objetivo: dividir uniformemente
  //  os arquivos, de modo que a árvore das chamadas recursivas seja mais balanceada,
  //  diminuindo as operações de leitura e escrita)
  //T1 = Ei - Esq (nro. elementos em A1); T2 = Es - Dir (nro. elementos em A2)
  //Se A1 tem menos registros, retira da área a menor chave e a escreve em Ei, atualizando
  //  Linf; caso contrário, retira a maior e a escreve em E2, atualizando Lsup
  if (Ei - Esq < Dir - Es) //se A1 for menor
  { RetiraMin(&Area, &R, &NRArea);
    EscreveMin(ArqEi, R, &Ei);
    Linf = R.Chave;
  }
  else //se forem do mesmo tamanho ou A2 menor
  { RetiraMax(&Area, &R, &NRArea);
    EscreveMax(ArqLEs, R, &Es);
    Lsup = R.Chave;
  }
}//fim do while
```

remove maior



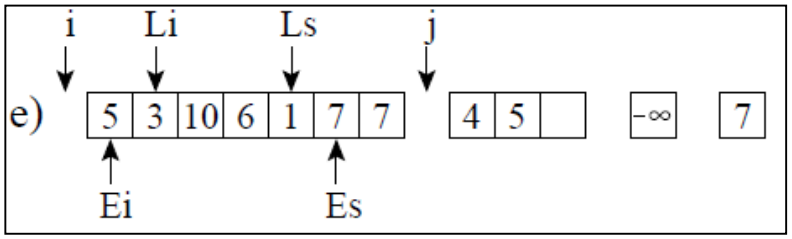


```
while (Ls >= Li)//Enquanto não cruzar
 { //Leitura
   if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
   { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
       LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
     else
       LeInf(ArqLi, &UltLido, &Li, &OndeLer);

     InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
     continue;//volta while, para ler o próximo do arquivo
   }
   //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
   //dos apontadores de leitura
   //faz com que nenhuma informação seja destruída durante a ordenação
   if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
     LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
   else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
               LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
             else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```



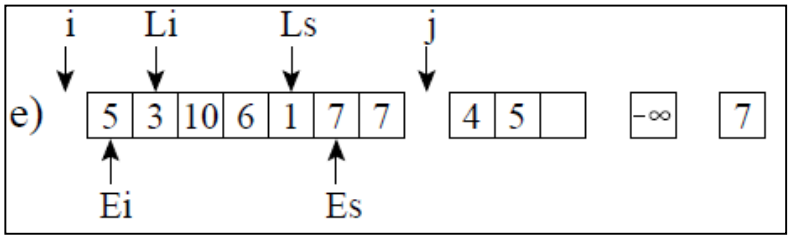
```
while (Ls >= Li)//Enquanto não cruzar
  { //Leitura
    if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
    { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else
        LeInf(ArqLi, &UltLido, &Li, &OndeLer);

      InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
      continue;//volta while, para ler o próximo do arquivo
    }

    //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
    //dos apontadores de leitura
    //faz com que nenhuma informação seja destruída durante a ordenação
    if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
    else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

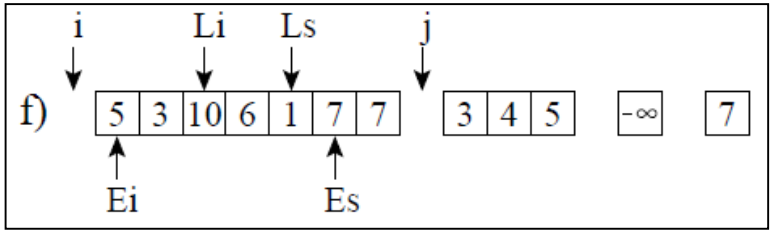
lê esquerda



```
//Compara a Ultima chave lida com o Lsup
//Se for maior, escreve na posição Es e atualiza "j" (marcador do subarquivo A2)
if (UltLido.Chave > Lsup)
{ *j = Es;
  EscreveMax(ArqLEs, UltLido, &Es);
  continue;//volta while, para ler o próximo do arquivo
}

//Compara a Ultima chave lida com o Linf
//Se for menor, escreve na posição Ei e atualiza "i" (marcador do subarquivo A1)
if (UltLido.Chave < Linf)
{ *i = Ei;
  EscreveMin(ArqEi, UltLido, &Ei);
  continue;//volta while, para ler o próximo do arquivo
}

//Se não for maior nem menor, i.e., está entre os dois, insere na memória temporária
InserirArea(&Area, &UltLido, &NRArea);
```

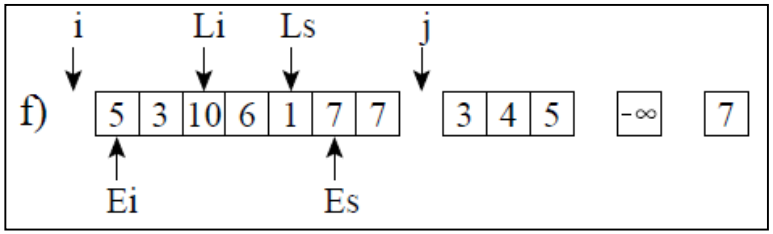
lê esquerda

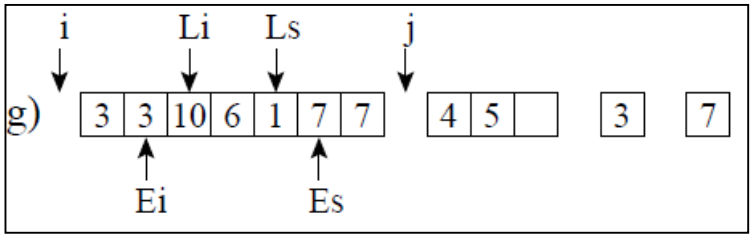




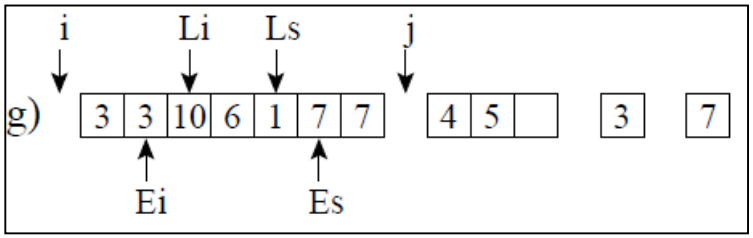
```
   //Quando a área temporária estiver cheia, deve-se remover um elemento da mesma
   //Como decidir qual remover? Considerando o Tam. de A1 e A2 (objetivo: dividir uniformemente
   //  os arquivos, de modo que a árvore das chamadas recursivas seja mais balanceada,
   //  diminuindo as operações de leitura e escrita)
   //T1 = Ei - Esq (nro. elementos em A1); T2 = Es - Dir (nro. elementos em A2)
   //Se A1 tem menos registros, retira da área a menor chave e a escreve em Ei, atualizando
   //  Linf; caso contrário, retira a maior e a escreve em E2, atualizando Lsup
   if (Ei - Esq < Dir - Es) //se A1 for menor
   { RetiraMin(&Area, &R, &NRArea);
     EscreveMin(ArqEi, R, &Ei);
     Linf = R.Chave;
   }
   else //se forem do mesmo tamanho ou A2 menor
   { RetiraMax(&Area, &R, &NRArea);
     EscreveMax(ArqLEs, R, &Es);
     Lsup = R.Chave;
   }
}//fim do while
```

remove menor



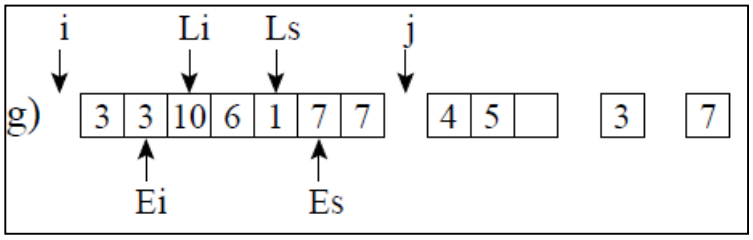


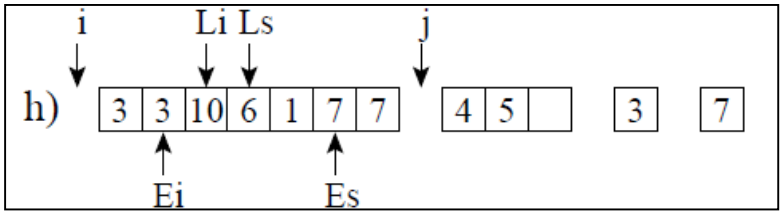
```
while (Ls >= Li)//Enquanto não cruzar
 { //Leitura
   if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
   { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
       LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
     else
       LeInf(ArqLi, &UltLido, &Li, &OndeLer);

     InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
     continue;//volta while, para ler o próximo do arquivo
   }
   //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
   //dos apontadores de leitura
   //faz com que nenhuma informação seja destruída durante a ordenação
   if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
     LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
   else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
               LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
             else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```



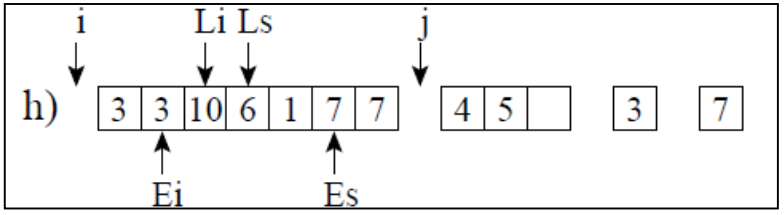
```
while (Ls >= Li)//Enquanto não cruzar
  { //Leitura
    if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
    { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else
        LeInf(ArqLi, &UltLido, &Li, &OndeLer);

      InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
      continue;//volta while, para ler o próximo do arquivo
    }

    //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
    //dos apontadores de leitura
    //faz com que nenhuma informação seja destruída durante a ordenação
    if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
    else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

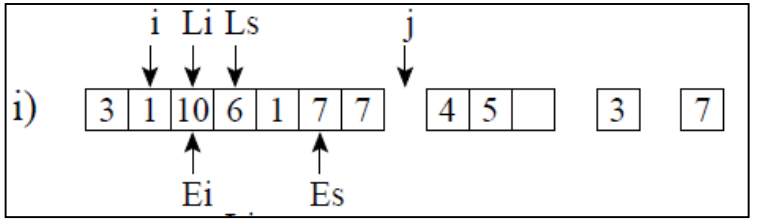
lê direita



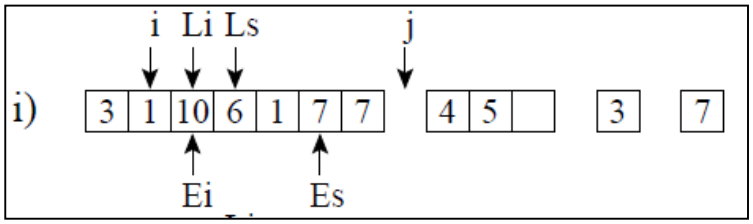


```
//Compara a Ultima chave lida com o Lsup
//Se for maior, escreve na posição Es e atualiza "j" (marcador do subarquivo A2)
if (UltLido.Chave > Lsup)
{ *j = Es;
  EscreveMax(ArqLEs, UltLido, &Es);
  continue;//volta while, para ler o próximo do arquivo
}

//Compara a Ultima chave lida com o Linf
//Se for menor, escreve na posição Ei e atualiza "i" (marcador do subarquivo A1)
if (UltLido.Chave < Linf)
{ *i = Ei;
  EscreveMin(ArqEi, UltLido, &Ei);
  continue;//volta while, para ler o próximo do arquivo
}

//Se não for maior nem menor, i.e., está entre os dois, insere na memória temporária
InserirArea(&Area, &UltLido, &NRArea);
```



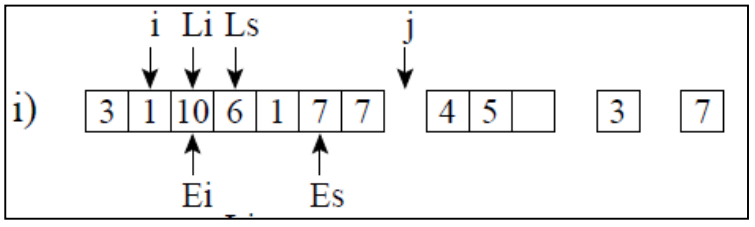


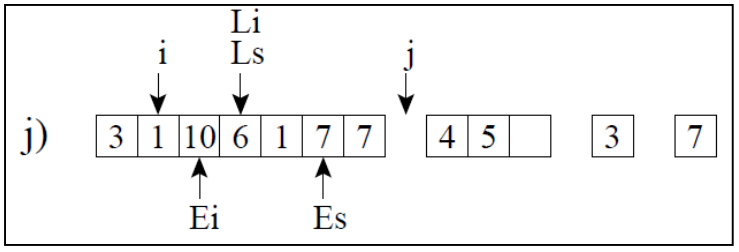
```
while (Ls >= Li)//Enquanto não cruzar
 { //Leitura
   if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
   { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
       LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
     else
       LeInf(ArqLi, &UltLido, &Li, &OndeLer);

     InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
     continue;//volta while, para ler o próximo do arquivo
   }
   //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
   //dos apontadores de leitura
   //faz com que nenhuma informação seja destruída durante a ordenação
   if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
     LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
   else if (Li == Ei)
         LeInf(ArqLi, &UltLido, &Li, &OndeLer);
       else if (OndeLer)//se não houver problema, lê do lado correto
             LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
           else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```



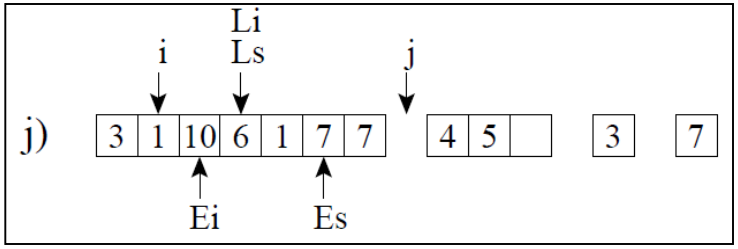
```
while (Ls >= Li)//Enquanto não cruzar
  { //Leitura
    if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
    { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else
        LeInf(ArqLi, &UltLido, &Li, &OndeLer);

      InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
      continue;//volta while, para ler o próximo do arquivo
    }

    //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
    //dos apontadores de leitura
    //faz com que nenhuma informação seja destruída durante a ordenação
    if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
    else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

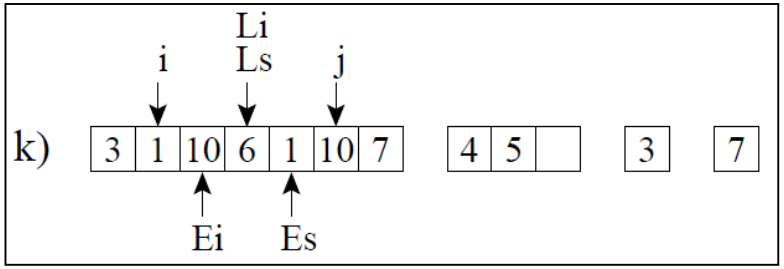
lê esquerda



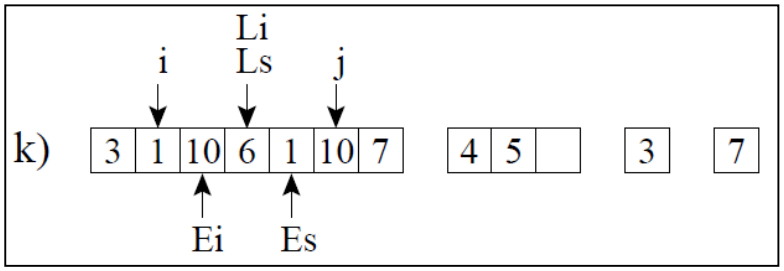


```
//Compara a Ultima chave lida com o Lsup
//Se for maior, escreve na posição Es e atualiza "j" (marcador do subarquivo A2)
if (UltLido.Chave > Lsup)
{ *j = Es;
  EscreveMax(ArqLEs, UltLido, &Es);
  continue;//volta while, para ler o próximo do arquivo
}

//Compara a Ultima chave lida com o Linf
//Se for menor, escreve na posição Ei e atualiza "i" (marcador do subarquivo A1)
if (UltLido.Chave < Linf)
{ *i = Ei;
  EscreveMin(ArqEi, UltLido, &Ei);
  continue;//volta while, para ler o próximo do arquivo
}

//Se não for maior nem menor, i.e., está entre os dois, insere na memória temporária
InserirArea(&Area, &UltLido, &NRArea);
```



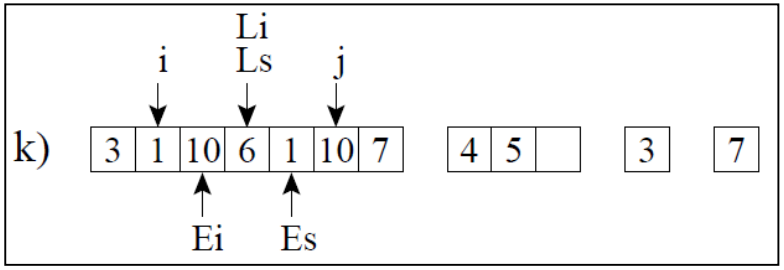


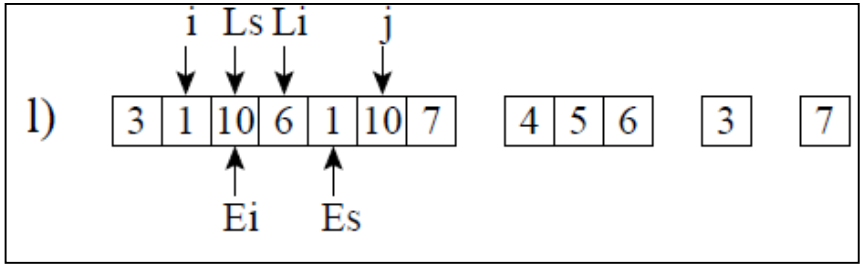
```
while (Ls >= Li)//Enquanto não cruzar
 { //Leitura
   if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
   { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
       LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
     else
       LeInf(ArqLi, &UltLido, &Li, &OndeLer);

     InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
     continue;//volta while, para ler o próximo do arquivo
   }
   //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
   //dos apontadores de leitura
   //faz com que nenhuma informação seja destruída durante a ordenação
   if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
     LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
   else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
               LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
             else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```



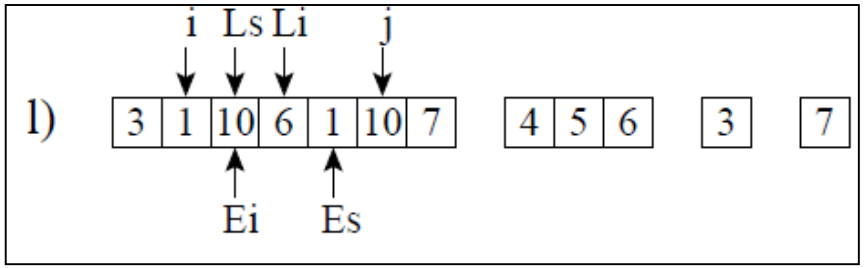
```
while (Ls >= Li)//Enquanto não cruzar
  { //Leitura
    if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
    { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else
        LeInf(ArqLi, &UltLido, &Li, &OndeLer);

      InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
      continue;//volta while, para ler o próximo do arquivo
    }

    //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
    //dos apontadores de leitura
    //faz com que nenhuma informação seja destruída durante a ordenação
    if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
    else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

lê direita



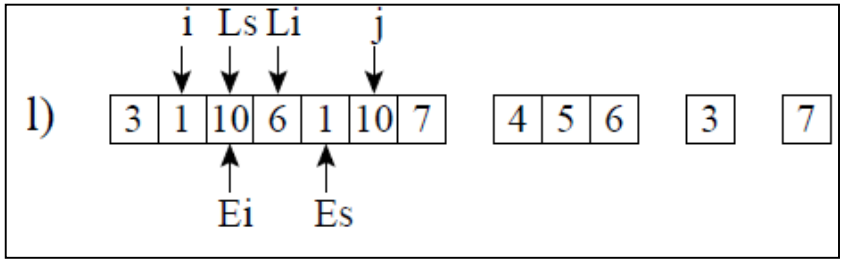


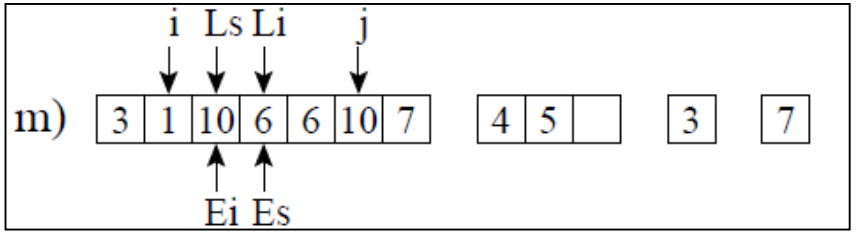
```
//Compara a Ultima chave lida com o Lsup
//Se for maior, escreve na posição Es e atualiza "j" (marcador do subarquivo A2)
if (UltLido.Chave > Lsup)
{ *j = Es;
  EscreveMax(ArqLEs, UltLido, &Es);
  continue;//volta while, para ler o próximo do arquivo
}

//Compara a Ultima chave lida com o Linf
//Se for menor, escreve na posição Ei e atualiza "i" (marcador do subarquivo A1)
if (UltLido.Chave < Linf)
{ *i = Ei;
  EscreveMin(ArqEi, UltLido, &Ei);
  continue;//volta while, para ler o próximo do arquivo
}

//Se não for maior nem menor, i.e., está entre os dois, insere na memória temporária
InserirArea(&Area, &UltLido, &NRArea);
```



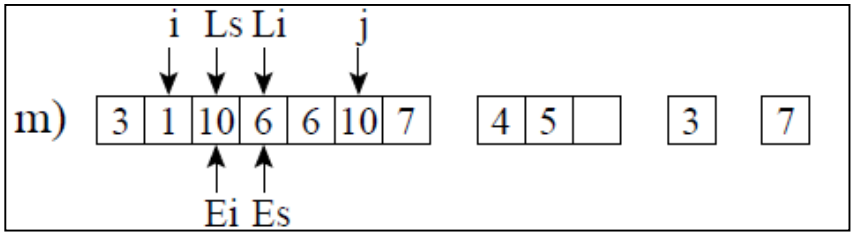
```
   //Quando a área temporária estiver cheia, deve-se remover um elemento da mesma
   //Como decidir qual remover? Considerando o Tam. de A1 e A2 (objetivo: dividir uniformemente
   //  os arquivos, de modo que a árvore das chamadas recursivas seja mais balanceada,
   //  diminuindo as operações de leitura e escrita)
   //T1 = Ei - Esq (nro. elementos em A1); T2 = Es - Dir (nro. elementos em A2)
   //Se A1 tem menos registros, retira da área a menor chave e a escreve em Ei, atualizando
   //  Linf; caso contrário, retira a maior e a escreve em E2, atualizando Lsup
   if (Ei - Esq < Dir - Es) //se A1 for menor
   { RetiraMin(&Area, &R, &NRArea);
     EscreveMin(ArqEi, R, &Ei);
     Linf = R.Chave;
   }
   else //se forem do mesmo tamanho ou A2 menor
   { RetiraMax(&Area, &R, &NRArea);
     EscreveMax(ArqLEs, R, &Es);
     Lsup = R.Chave;
   }
 }//fim do while
```

remove maior

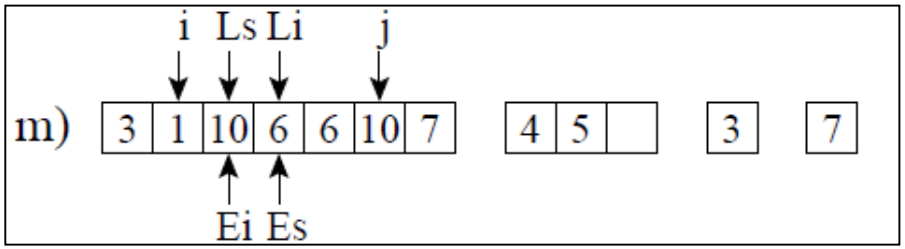


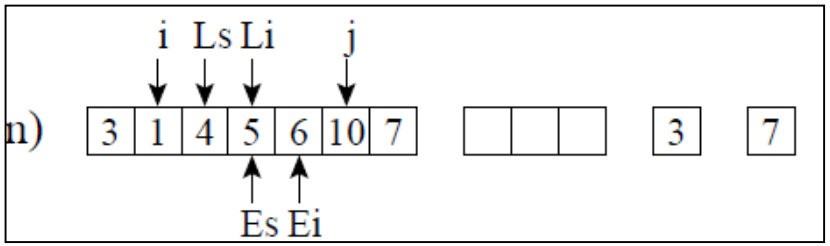


```
while (Ls >= Li)//Enquanto não cruzar
  { //Leitura
    if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
    { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else
        LeInf(ArqLi, &UltLido, &Li, &OndeLer);

      InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
      continue;//volta while, para ler o próximo do arquivo
    }

    //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
    //dos apontadores de leitura
    //faz com que nenhuma informação seja destruída durante a ordenação
    if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
    else if (Li == Ei)
          LeInf(ArqLi, &UltLido, &Li, &OndeLer);
        else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```



```
//Se ainda existirem registros na área temporária, copiar os mesmos de maneira ordenada para
//  o arquivo. Desse modo, vai extraindo a menor chave e escrevendo em Ei
while (Ei <= Es)
   { RetiraMin(&Area, &R, &NRArea);
     EscreveMin(ArqEi, R, &Ei);
   }
```

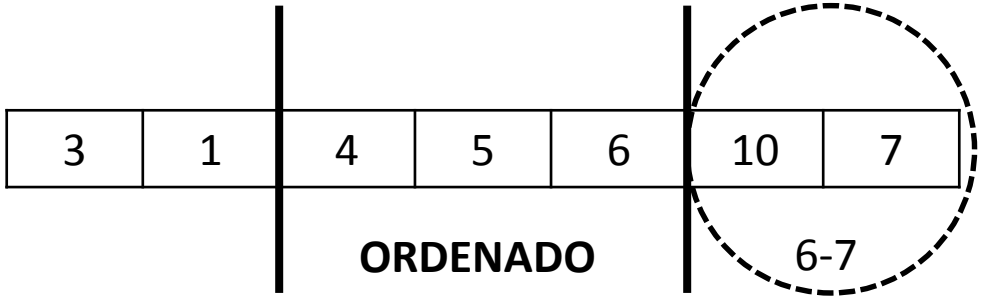




```
void QuicksortExterno(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs, int Esq, int Dir)
{ int i, j;
  TipoArea Area;    /* Area de armazenamento interna*/

  printf("Entrou QS\n");

  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 */
    printf("Primeiro: %d - %d; %d - %d\n",Esq,i,j,Dir);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
  }
  else
    { printf("Segundo: %d - %d; %d - %d\n",j,Dir,Esq,i);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    }
}
```

((2-1) < (7-6))

Chamada pendente 1



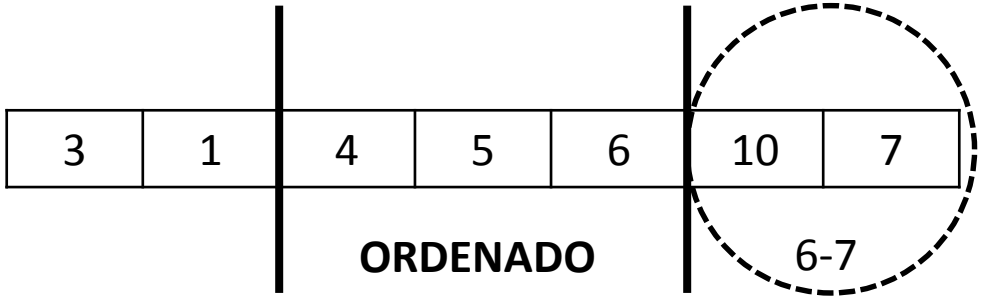
Esq = 1
Dir = 7

i = 2
j = 6

```
void QuicksortExterno(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs, int Esq, int Dir)
{ int i, j;
  TipoArea Area;    /* Area de armazenamento interna*/

  printf("Entrou QS\n");

  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 */
    printf("Primeiro: %d - %d; %d - %d\n",Esq,i,j,Dir);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
  }
  else
    { printf("Segundo: %d - %d; %d - %d\n",j,Dir,Esq,i);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    }
}
```

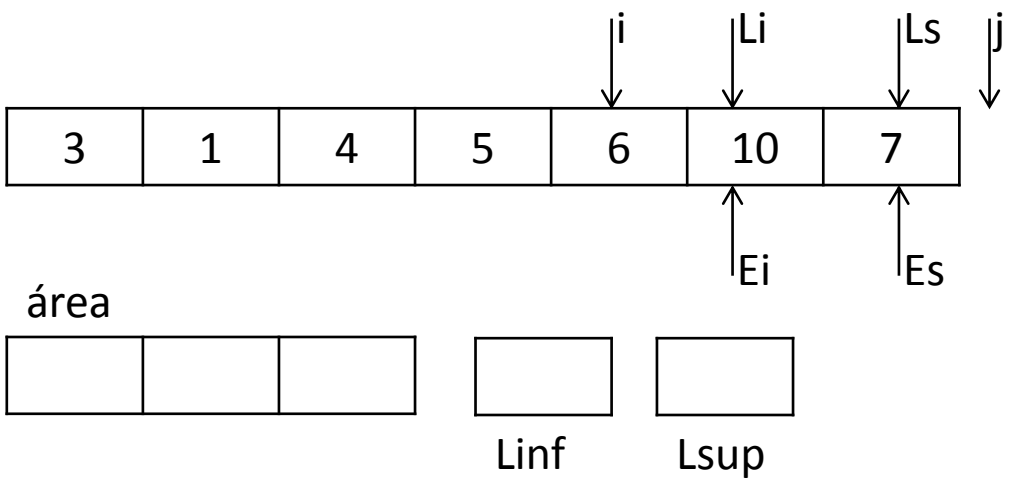
((7-6) < 1)

chama novamente



Esq = 6
Dir = 7

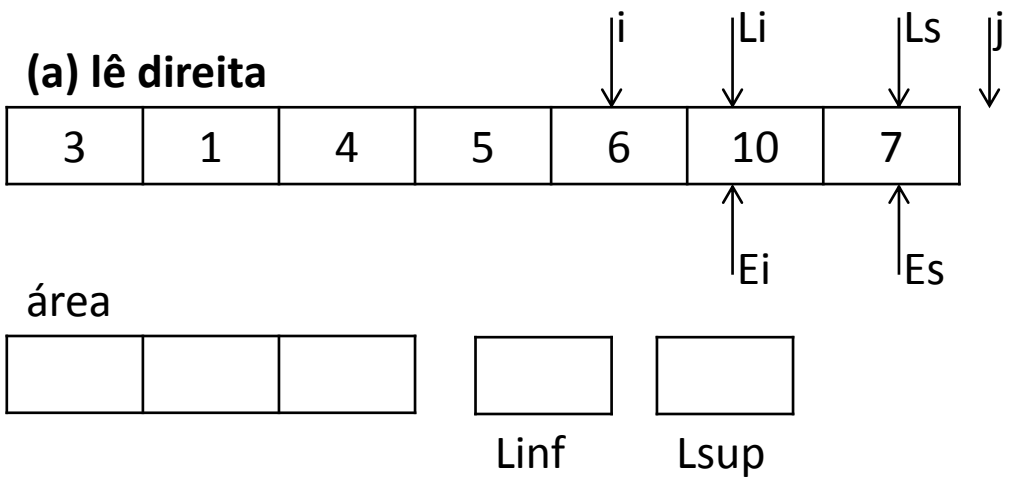
```c
//Li = leitura inferior
//Ls = leitura superior
//Ei = escrita inferior
//Es = escrita superior
void Particao(FILE **ArqLi, FILE **ArqEi,  FILE **ArqLEs, TipoArea Area,
          int Esq, int Dir, int *i, int *j)
{ int Ls = Dir, Es = Dir, Li = Esq, Ei = Esq, NRArea = 0, Linf = INT_MIN,
      Lsup = INT_MAX;
  short OndeLer = TRUE;//Começa pelo lado direito
  TipoRegistro UltLido, R;

  printf("Entrou Particao\n");

  //posiciona os ponteiros
  fseek (*ArqLi, (Li - 1)* sizeof(TipoRegistro), SEEK_SET );
  fseek (*ArqEi, (Ei - 1)* sizeof(TipoRegistro), SEEK_SET );

  *i = Esq - 1;
  *j = Dir + 1;
```



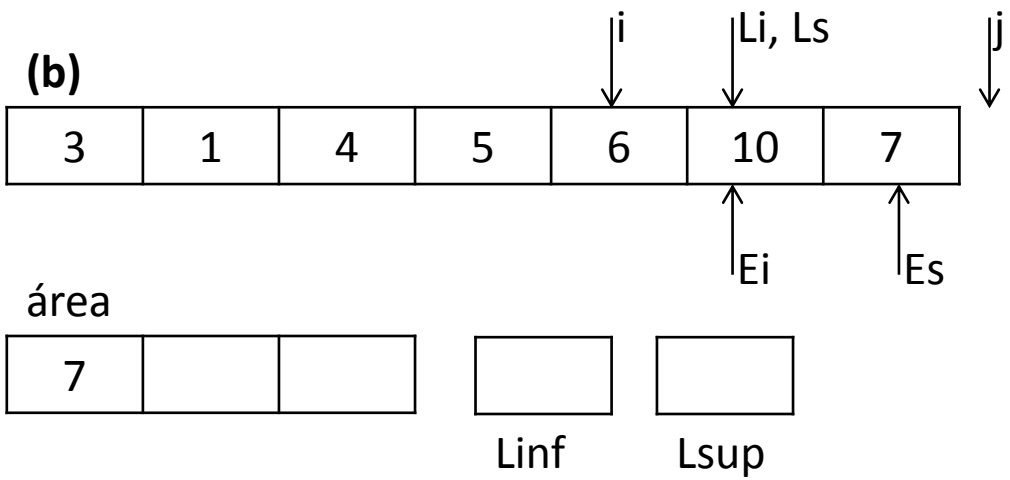
Ls = 7
Es = 7
Li = 6
Ei = 6
NRArea = 0
Linf = -inf*
Lsup = +inf*

i = 5
j = 8

```
while (Ls >= Li)//Enquanto não cruzar
{ //Leitura
  if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
  { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
    else
      LeInf(ArqLi, &UltLido, &Li, &OndeLer);

    InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
    continue;//volta while, para ler o próximo do arquivo
  }

  //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
  //dos apontadores de leitura
  //faz com que nenhuma informação seja destruída durante a ordenação
  if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
    LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
  else if (Li == Ei)
         LeInf(ArqLi, &UltLido, &Li, &OndeLer);
       else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

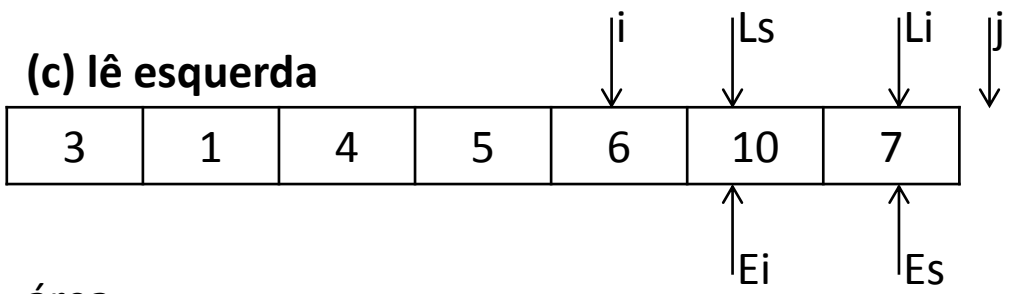


```
while (Ls >= Li)//Enquanto não cruzar
{ //Leitura
  if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
  { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
    else
      LeInf(ArqLi, &UltLido, &Li, &OndeLer);

    InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
    continue;//volta while, para ler o próximo do arquivo
  }

  //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
  //dos apontadores de leitura
  //faz com que nenhuma informação seja destruída durante a ordenação
  if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
    LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
  else if (Li == Ei)
         LeInf(ArqLi, &UltLido, &Li, &OndeLer);
       else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

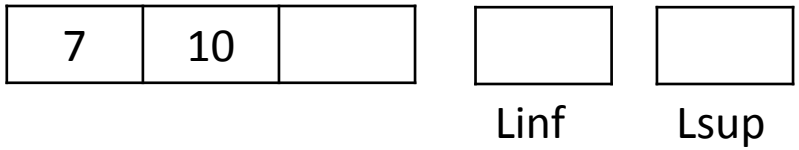


```
while (Ls >= Li)//Enquanto não cruzar
{ //Leitura
  if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
  { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
    else
      LeInf(ArqLi, &UltLido, &Li, &OndeLer);

    InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
    continue;//volta while, para ler o próximo do arquivo
  }

  //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
  //dos apontadores de leitura
  //faz com que nenhuma informação seja destruída durante a ordenação
  if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
    LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
  else if (Li == Ei)
         LeInf(ArqLi, &UltLido, &Li, &OndeLer);
       else if (OndeLer)//se não houver problema, lê do lado correto
              LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
            else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

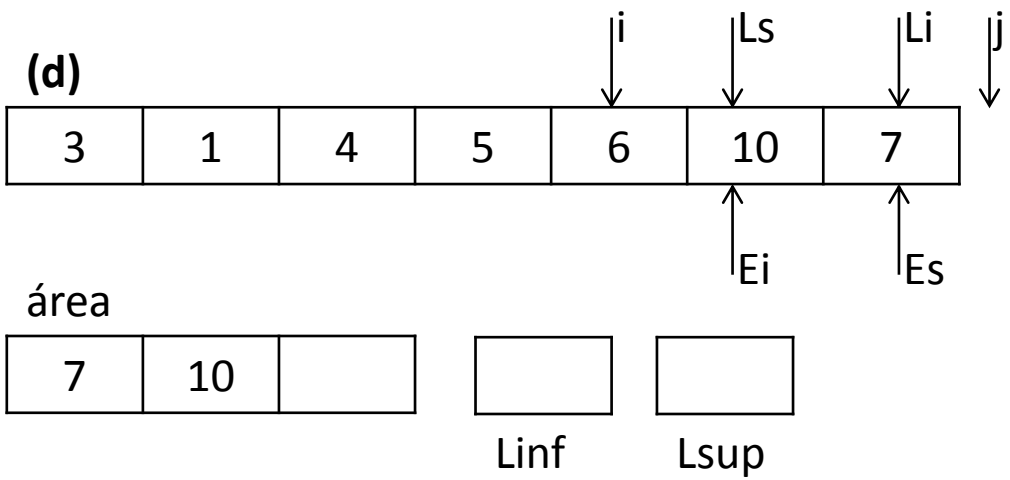
(c) lê esquerda



área

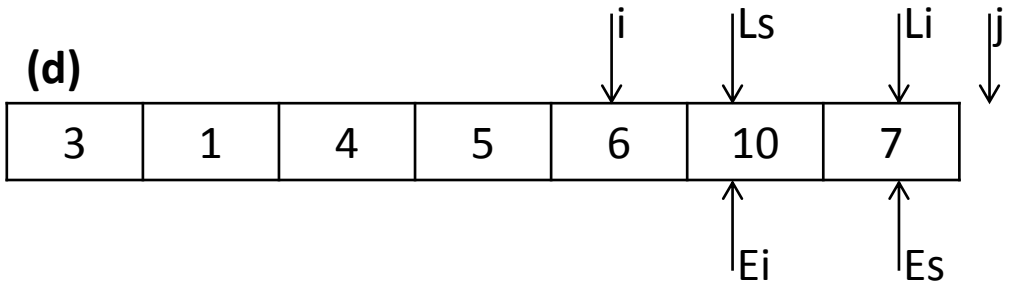


```
while (Ls >= Li)//Enquanto não cruzar
  { //Leitura
    if (NRArea < TamArea - 1) //Se memória temporária não estiver cheia
    { if (OndeLer)//Controla o lado em que a leitura será realizada: se T->D; se F->E
        LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
      else
        LeInf(ArqLi, &UltLido, &Li, &OndeLer);

      InserirArea(&Area, &UltLido, &NRArea);//Insere na memória
      continue;//volta while, para ler o próximo do arquivo
    }

    //Para garantir que os apontadores de escrita estejam pelo menos uma passo atrás
    //dos apontadores de leitura
    //faz com que nenhuma informação seja destruída durante a ordenação
    if (Ls == Es)//se os dois apontam para o mesmo lugar, não importa de onde deve-se ler (OndeLer)
      LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);//vai ler do lado que está com problema
    else if (Li == Ei)
           LeInf(ArqLi, &UltLido, &Li, &OndeLer);
         else if (OndeLer)//se não houver problema, lê do lado correto
                LeSup(ArqLEs, &UltLido, &Ls, &OndeLer);
              else LeInf(ArqLi, &UltLido, &Li, &OndeLer);
```

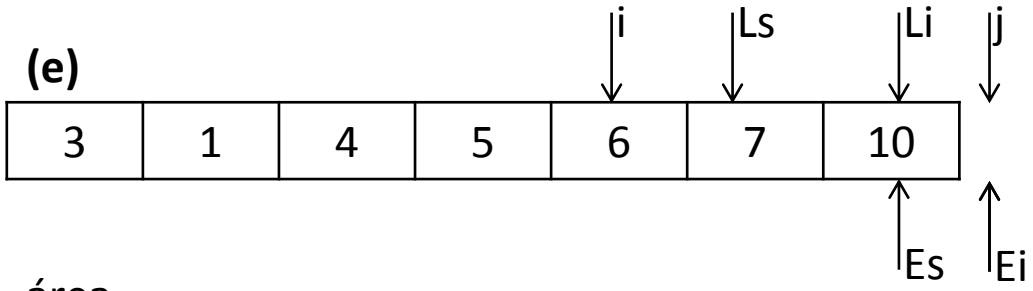


```
//Se ainda existirem registros na área temporária, copiar os mesmos de maneira ordenada para
//  o arquivo. Desse modo, vai extraindo a menor chave e escrevendo em Ei
while (Ei <= Es)
   { RetiraMin(&Area, &R, &NRArea);
     EscreveMin(ArqEi, R, &Ei);
   }
```

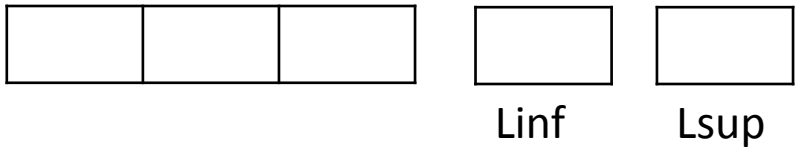
(d)



(e)



área



Esq = 6
Dir = 7

i = 5
j = 8

```
void QuicksortExterno(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs, int Esq, int Dir)
{ int i, j;
  TipoArea Area;    /* Area de armazenamento interna*/

  printf("Entrou QS\n");

  if (Dir - Esq < 1) return;

  FAVazia(&Area);

  Particao(ArqLi, ArqEi, ArqLEs, Area, Esq, Dir, &i, &j);

  if (i - Esq < Dir - j) ((5-6) < (7-8))
  {  /* ordene primeiro o subarquivo menor */
    printf("Primeiro: %d - %d; %d - %d\n",Esq,i,j,Dir);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
  }
  else
    { printf("Segundo: %d - %d; %d - %d\n",j,Dir,Esq,i);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    }
}
```

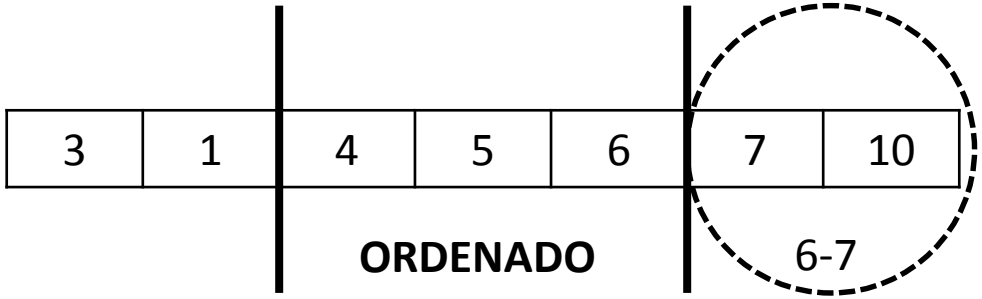
vai tentar de 8 a 7 e vai falhar!!! (-1)

vai tentar de 6 a 5 e vai falhar!!! (-1)

Esq = 6
Dir = 7

i = 5
j = 8

```
void QuicksortExterno(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs, int Esq, int Dir)
{ int i, j;
  TipoArea Area;    /* Area de armazenamento interna*/

  printf("Entrou QS\n");

  if (Dir - Esq < 1) return;

  FAVazia(&Area);

  Particao(ArqLi, ArqEi, ArqLEs, Area, Esq, Dir, &i, &j);

  if (i - Esq < Dir - j) ((2-1) < (7-6))
  {  /* ordene primeiro o subarquivo menor */
    printf("Primeiro: %d - %d; %d - %d\n",Esq,i,j,Dir);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
  }
  else
    { printf("Segundo: %d - %d; %d - %d\n",j,Dir,Esq,i);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir); Fim pendente 1
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i); Chama pendente 2
    }
}
```



```
void QuicksortExterno(FILE **ArqLi, FILE **ArqEi, FILE **ArqLEs, int Esq, int Dir)
{ int i, j;
  TipoArea Area;    /* Area de armazenamento interna*/

  printf("Entrou QS\n");

  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 */
    printf("Primeiro: %d - %d; %d - %d\n",Esq,i,j,Dir);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
  }
  else
    { printf("Segundo: %d - %d; %d - %d\n",j,Dir,Esq,i);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, j, Dir);
      QuicksortExterno(ArqLi, ArqEi, ArqLEs, Esq, i);
    }
}
```

((2-1) < (7-6))

Fim pendente 1

Chama pendente 2

terminar em casa...

Considerações Finais

- A complexidade de pior caso é $O(n^2/TamArea)$
  - n é o número de registros a serem ordenados
  - TamArea o número de registros que podem ser armazenados na área de armazenamento em memória interna

Referência

- N. Ziviani. Projeto de Algoritmos com Implementações em Pascal e C. Segunda Edição. Thomson. 2004.