```
heapsort (int *x, int n)
{
  int i, e, s, f, aux;
  /*fase de pré-processamento - cria heap inicial*/
  for (i=1; i<n; i++)
  {
    e = x[i];
    s = i;
    f = (s-1)/2;
    while (s>0 && x[f]<e)
    {
      x[s] = x[f];
      s = f;
      f = (s-1)/2;
    }
    x[s] = e;
  }
```

```
/*fase de seleção = remove x[0] várias vezes, inserindo-o
em sua posição correta e acertando o heap*/
for (i=n-1; i>0; i--) {
   aux = x[i];
   x[i] = x[0];
   f = 0;
   if (i==1)
      s = -1;
   else
      s = 1;
   if (i>2 && x[2]>x[1])
      s = 2;
   while (s>=0 && aux<x[s]) {
      x[f] = x[s];
      f = s;
      s = 2*f+1;
      if (s+1 <= i-1 && x[s]<x[s+1])
         s = s+1;
      if (s > i-1)
         s = -1;
}
x[f] = aux;   }/*chave de fechamento do for*/   } /*chave de fechamento da função*/
```

Heap Sort

Para analisar o heap sort, observe que uma árvore binária completa com *n* nós tem log (*n*+1) níveis. Por conseguinte, se cada elemento no vetor fosse uma folha, exigindo que fosse filtrado pela árvore inteira durante a criação e o ajuste do heap, a classificação ainda seria **O**(*n* log *n*).

No caso médio, o heap sort não é tão eficiente quanto o quick sort. Experimentos indicam que o heap sort exige, aproximadamente, o dobro do tempo do quick sort para a entrada classificada aleatoriamente. Entretanto, o heap sort é bem superior ao quick sort no pior caso. Na realidade, o heap sort permanece **O** (*n* log *n*) no pior caso.

Heap Sort

Essa classificação não é muito eficiente para *n* pequeno devido à sobrecarga da criação do heap inicial e do cálculo da posição de pais e filhos.

A exigência de espaço para o heap sort (índices do vetor à parte) requer somente um registro adicional para armazenamento temporário durante a troca, desde que usada a implementação em vetor de uma árvore binária quase completa.

O heapsort não é um algoritmo de ordenação estável. Porém, é possível adaptar a estrutura a ser ordenada de forma a tornar a ordenação estável.

UNIVASF

Comparação entre os Métodos

A ordenação interna é utilizada quando todos os registros do arquivo cabem na memória principal.

Veremos quadros comparativos do tempo total real para ordenar arranjos com 500, 5.000, 10.000 e 30.000 registros na ordem aleatória, na ordem ascendente e na ordem descendente, respectivamente.

Em cada tabela, o método que levou menos tempo real para executar recebeu o valor 1 e os outros receberam valores relativos a ele.

Ordem Aleatória dos Registros

	500	5000	10000	30000
Inserção	11.3	87	161	-
Seleção	16.2	124	228	-
Shellsort	1.2	1.6	1.7	2
Quicksort	1	1	1	1
Heapsort	1.5	1.6	1.6	1.6

Comparação entre os Métodos

Ordem Ascendente dos registros

	500	5000	10000	30000
Inserção	1	1	1	1
Seleção	128	1524	3066	-
Shellsort	3.9	6.8	7.3	8.1
Quicksort	4.1	6.3	6.8	7.1
Heapsort	12.2	20.8	22.4	24.6

Ordem Descendente dos Registros

Inserção	40.3	305	575	-
Seleção	29.3	221	417	-
Shellsort	1.5	1.5	1.6	1.6
Quicksort	1	1	1	1
Heapsort	2.5	2.7	2.7	2.9

UNIVASF

Classificação

É importante perceber que, quando o tamanho de uma lista *n* é pequeno, uma classificação **O**(n²) é em geral mais eficiente do que uma classificação **O**(n log n).

Isto acontece porque usualmente as classificações **O**(n²) são muito simples de programar e exigem bem poucas ações além de comparações e trocas em cada passagem. Por causa dessa baixa sobrecarga, a constante de proporcionalidade é bem pequena. Em geral, uma classificação **O**(n log n) é muito complexa e emprega um grande

número de operações adicionais em cada passagem para diminuir o número das passagens subsequentes. Sendo assim, sua constante de proporcionalidade é maior.

Quando *n* é grande, n^2 supera n log n, de modo que as constantes de proporcionalidade não desempenham um papel importante na determinação da classificação mais veloz. Entretanto, quando *n* é pequeno, n^2 não é muito maior que n log n de modo que uma grande diferença nessas constantes frequentemente faz com que a classificação **O**(n^2) seja mais rápida.

Ordenação Externa

A ordenação externa envolve arquivos compostos por um número de registros maior do que a memória interna do computador pode armazenar.

Os métodos de ordenação externa possuem particularidades que os diferenciam dos métodos de ordenação interna.

Em ambos os casos o problema é o mesmo: rearranjar os registros de um arquivo em ordem ascendente ou descendente.

UNIVASF

Ordenação Externa

Entretanto, na ordenação externa as estruturas de dados têm que levar em conta o fato de que os dados estão armazenados em unidades de memória externa, relativamente muito mais lentas do que a memória principal.

Nas memórias externas, tais como **fitas** e **discos magnéticos**, os dados são armazenados como um arquivo sequencial, onde apenas um registro pode ser acessado em um dado momento.

Esta é uma restrição forte se comparada com as possibilidades de acesso da estrutura de dados do tipo vetor.

UNIVASF

Ordenação Externa

Existem três importantes fatores que tornam os algoritmos para ordenação externa diferentes dos algoritmos para ordenação interna, sendo:

1. O custo para acessar um item é algumas ordens de grandeza maior do que os custos de processamento na memória interna.

   Custo principal ➔ o custo de transferir dados entre a memória interna e a memória externa.

2. Existem restrições severas de acesso aos dados.

   (a) Itens armazenados em fita magnética só podem ser acessados de forma sequencial.

UNIVASF

Ordenação Externa

(b) Itens armazenados em disco magnético podem ser acessados diretamente, mas a um custo maior do que o custo para acessar sequencialmente, o que contra-indica o uso do acesso direto.

3. O desenvolvimento de métodos de ordenação externa é muito dependente do estado atual da tecnologia. A grande variedade de tipos de unidades de memória externa pode tornar os métodos de ordenação externa dependentes de vários parâmetros que afetam seus desempenhos.

UNIVASF

Ordenação Externa

Fatores associados à eficiência dos Métodos de Ordenação Externa:

O aspecto sistema de computação deve ser considerado no mesmo nível do aspecto algorítmico.

A grande ênfase deve ser na minimização do número de vezes que cada item é transferido entre a memória interna e a memória externa.

Cada transferência deve ser realizada de forma tão eficiente quanto as características dos equipamentos disponíveis permitam.

UNIVASF

Ordenação Externa

Método de ordenação externa mais importante

⬇

Ordenação por Intercalação

Como vimos anteriormente, intercalar significa combinar dois ou mais blocos ordenados em um único bloco ordenado através de seleções repetidas entre os itens disponíveis em cada momento.

A intercalação é utilizada como uma operação auxiliar no processo de ordenação.

Ordenação Externa

Como veremos, a estratégia geral é a mesma. Contudo, algumas particularidades existem na implementação da intercalação na memória externa:

1. É realizada uma primeira passada sobre o arquivo, quebrando-o em blocos do tamanho da memória interna disponível. Cada bloco é então ordenado na memória interna.

2. Os blocos ordenados são intercalados, fazendo-se varias passadas sobre o arquivo. A cada passada são criados blocos ordenados cada vez maiores, até que todo o arquivo esteja ordenado.

UNIVASF

Ordenação Externa

Os algoritmos para ordenação externa devem procurar reduzir o número de passadas sobre o arquivo.

Os bons métodos de ordenação geralmente envolvem no total menos do que 10 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 auxiliar.

UNIVASF

Ordenação Externa

Ordenação por Intercalação – Mergesort

Princípio:

1. Partir o arranjo em dois.
2. Intercalar dois arranjos independentes.

Vantagens:

1. Pior caso: **O** (n log n)
2. Só precisa de acesso sequencial aos dados.
3. Boa opção quando os dados estão em uma lista encadeada.

Desvantagem: Espaço de armazenamento adicional é necessário.

UNIVASF

Ordenação Externa

Ordenação de Sequências - Fusão (intercalação ou merge) Direta

Corresponde a combinar duas ou mais sequências ordenadas para formar uma única sequência ordenada através da aplicação de repetidas seleções entre os componentes acessíveis em cada ocasião.

A operação de fusão é muito mais simples que a de ordenação, sendo empregada, como operação auxiliar, no processo mais complexo de ordenação sequencial.

UNIVASF

Ordenação Externa

O método Merge Sort (intercalação direta) funciona da seguinte maneira: divida o arquivo em **n** sub-arquivos de tamanho **1** e intercale pares de arquivos adjacentes.

Temos, então, aproximadamente **n/2** arquivos de tamanho **2**.

Repita esse processo até restar apenas um arquivo de tamanho **n**. Exemplo:

Ordenação Externa

As fases de particionamento não oferecem nenhuma contribuição ao processo de ordenação propriamente dito, uma vez que elas, de modo algum, efetuam a permutação de elementos.

Estas operações podem ser eliminadas, através da combinação da fase de particionamento com a de fusão.

Ao invés de se efetuar uma fusão para produzir uma sequência única, o resultado do processo de fusão é imediatamente redistribuído em duas ou mais sequências (fitas), as quais constituirão as fontes de dados que alimentarão o passo subsequente. Este método é chamado: **Fusão de uma Única Fase** ou **Fusão Balanceada**.

UNIVASF

Ordenação Externa

Estudaremos agora o processo de **Intercalação Balanceada de Vários Caminhos**, para tal consideraremos o processo de ordenação externa quando o arquivo a ser ordenado encontra-se armazenado em fita magnética.

Exemplo:

Considere um arquivo composto pelos registros com as seguintes chaves:

A S O R T I N G A N D M E R G I N G E X A M P L E

Os 25 registros devem ser ordenados de acordo com as chaves e colocados em uma fita de saída. Neste caso os registros são lidos um após o outro.

Ordenação Externa

Assuma que a memória interna do computador a ser utilizado só tem espaço para 3 registros, e o número de unidades de fita magnética é 6.

Chaves: A S O R T I N G A N D M E R G I N G E X A M P L E

Na 1ª etapa o arquivo é lido de 3 em 3 registros. Cada bloco de 3 registros é ordenado e escrito em uma das fitas de saída.

Para o exemplo são lidos os registros:

1. **A S O** e escrito o bloco **A O S** na fita 1.
2. A seguir são lidos os registros **R T I** e escrito o bloco **I R T** na fita 2, e assim por diante, conforme mostra o exemplo a seguir.

Ordenação Externa

Chaves: A S O R T I N G A N D M E R G I N G E X A M P L E

fita 1:
fita 2:
fita 3:

Observação: Quando 3 fitas são utilizadas denomina-se intercalação-de-3-caminhos.

Na 2ª etapa os blocos ordenados devem ser intercalados.

1. O 1º registro de cada uma das 3 fitas é lido para a memória interna, ocupando toda a memória interna.

Ordenação Externa

2. O registro contendo a menor chave dentre as 3 é retirado e colocado em uma fita de saída; e o próximo registro da fita que continha tal chave é lido para a memória interna.

3. Repete-se o processo até que o 3º registro de um dos blocos é lido, o que faz com que a fita em questão fique inativa até que o 3º registro das outras fitas também sejam lidos e escritos na fita de saída, formando um bloco de 9 registros ordenados.

A seguir o 2º bloco de 3 registros de cada fita é lido para formar outro bloco ordenado de 9 registros, o qual é escrito em uma outra fita.

Ordenação Externa

Ao final 3 novos blocos ordenados são obtidos, conforme mostra o exemplo:

Fitas obtidas na primeira etapa:

fita 1:	A O S	D M N	A E X
fita 2:	I R T	E G R	L M P
fita 3:	A G N	G I N	E

Fitas gerada na segunda etapa:

fita 4:			
fita 5:	D E G	G I M	N N R
fita 6:	A E E	L M P	X

A seguir mais uma intercalação-de-3-caminhos das fitas 4, 5 e 6 para as fitas 1, 2 e 3 completa a ordenação.

Ordenação Externa

Considerações sobre a Intercalação

Se o arquivo exemplo tivesse um número maior de registros, então vários blocos ordenados de 9 registros seriam formados nas fitas 4, 5 e 6.

Neste caso, a segunda passada produziria blocos ordenados de 27 registros nas fitas 1, 2 e 3.

A terceira passada produziria blocos ordenados de 81 registros nas fitas 4, 5 e 6, e assim sucessivamente, até obter-se um único bloco ordenado.

UNIVASF

Ordenação Externa

Neste ponto cabe a seguinte pergunta: quantas passadas são necessárias para ordenar um arquivo de tamanho arbitrário?

Considere um arquivo contendo **n** registros (neste caso cada registro contém apenas uma palavra) e uma memória interna de **m** palavras.

A passada inicial sobre o arquivo produz **n/m** blocos ordenados (se cada registro contiver **k** palavras, **k > 1**, então teríamos **n/m/k** blocos ordenados.)

Ordenação Externa

Seja **P** uma função de complexidade tal que **P**(**n**) é o número de passadas para a fase de intercalação dos blocos ordenados. Seja **f** o número de fitas utilizadas em cada passada.

Então, para uma intercalação-de-f-caminhos o número de passadas é: $\mathbf{P}(\mathbf{n}) = \log_{\mathbf{f}} (\mathbf{n}/\mathbf{m})$

No exemplo anterior, temos **n**=25, **m**=3 e **f**=3.

Logo,

$$\mathbf{P}(\mathbf{n}) = \log_{\mathbf{3}} (25/3) = 2$$