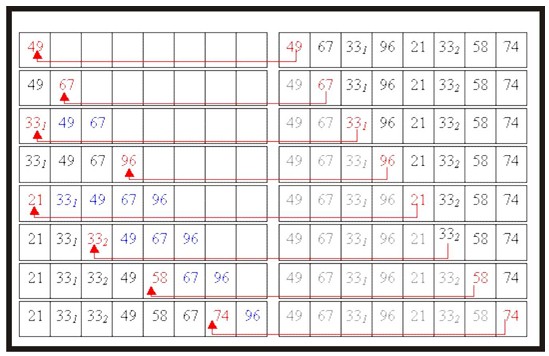
Algoritmos de Ordenação

* Insertion Sort
* Selection Sort
* Quicksort
* Shellsort
* Merge Sort

Insertion Sort

Insertion Sort é um algoritmo elementar de ordenação. É eficiente quando aplicado à um vetor com poucos elementos. Em cada passo, a partir de i = 2, o i-próximo item da sequência fonte é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado.



O algoritmo procede da seguinte maneira:

1. Zera os valores das variáveis de medição através do método ClearAll();

2. Inicia a contagem de tempo com a função start();

3. O primeiro laço de repetição tem a função de controlar o tamanho da sequência analisada;

4. O segundo laço é responsável de colocar o novo elemento da sequência, em relação à anterior, no lugar apropriado;

5. A cada comparação incrementa a variável mComparations e a cada movimentação incrementa a variável mMoviments;

6. Pausa a contagem de tempo e calcula o tempo gasto armazenando o valor na variável mTime;

Vantagens

- Fácil Implementação

- Algoritmo Estável

- O vetor já ordenado favorece a ordenação

Desvantagens

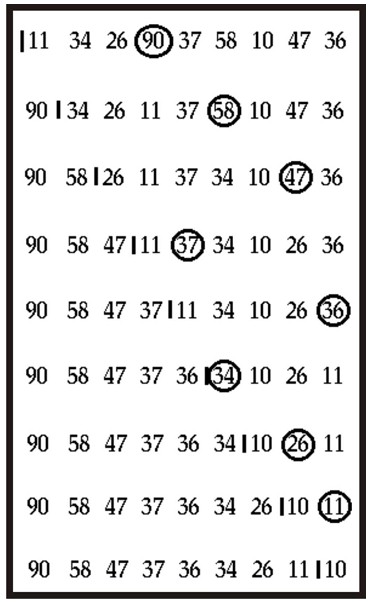
- Número grande de movimentações

- Ordem de complexidade quadrática

- Ineficiente quando o vetor está ordenado inversamente;

Selection Sort

Tem como princípio de funcionamento selecionar o menor item do vetor e a seguir trocá-lo pela primeira posição do vetor. Isto ocorre para os n-1 elementos restantes, depois com os n-2 itens, até que reste apenas um elemento. A principal diferença destes métodos em relação aos dois já apresentados é que ele realiza apenas uma troca por iteração.



O algoritmo procede da seguinte forma:

1. Zera os valores das variáveis de medição através do método ClearAll();

2. Inicia a contagem de tempo com a função start();

3. O primeiro laço determina a dimensão de busca do menor elemento;

4. O segundo laço é responsável por realizar a busca pelo menor elemento;

5. Feito a busca é realizada a troca do menor elemento pelo primeiro elemento;

6. Após a troca o processo é realizado novamente para os n-i itens restantes;

7. A cada comparação incrementa a variável mComparations e a cada movimen-tação incrementa a variável mMoviments;

8. Pausa a contagem de tempo e calcula o tempo gasto armazenando o valor na variável mTime;

Vantagens

- Fácil Implementação

- Pequeno número de movimentações

- Interessante para arquivos pequenos

Desvantagens

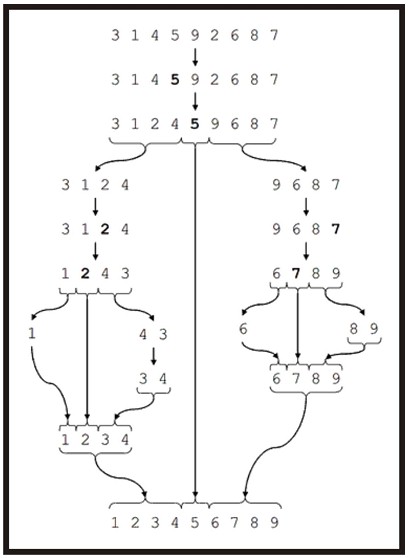
- O fato de o arquivo já estar ordenado não influencia em nada

- Ordem de complexidade quadrática

- Algoritmo não estável

Quicksort

É o algoritmo mais rápido que se conhece entre os de ordenação interna para uma ampla variedade de situações. Porém em raras instâncias especiais ele é lento. Adotando a estratégia Dividir para conquistar o funcionamento resume-se a dividir o problema de ordenar um vetor de n posições em dois outros menores. O QuickSort é provavelmente o mais utilizado, porém a sua implementação demanda um pouco mais de paciência e cautela.



O algoritmo procede da seguinte maneira:

1. Zera os valores das variáveis de medição através do método ClearAll();

2. Inicia a contagem de tempo com a função start ();

3. Escolha um elemento aux1 do vetor Array para ser o pivô;

4. Usando aux1 divida o vetor Array em duas partes, da seguinte maneira:

Começando com i = posição inicial, percorra o vetor em forma crescente até que um elemento Array[i] = aux1 seja encontrado.

Começando com j = posição nal, percorra o vetor em forma descendente até que um elemento A[j] = X seja encontrado;

Se i <= j, troque Array[i] com Array[j];

Continue percorrendo o vetor procurando elementos Array[i] e Array[j] até que os contadores i e j se cruzem (j < i).

5. Repita, recursivamente, os passos 3 e 4 para as duas partes

6. A cada comparação incrementa a variável mComparations e a cada movimentação incrementa a variável mMoviments;

7. Pausa a contagem de tempo e calcula o tempo gasto armazenando o valor na variável mTime;

Vantagens

- Extremamente Eficiente

- Necessita apenas de um pequena pilha como memória extra

- Complexidade n log n

Desvantagens

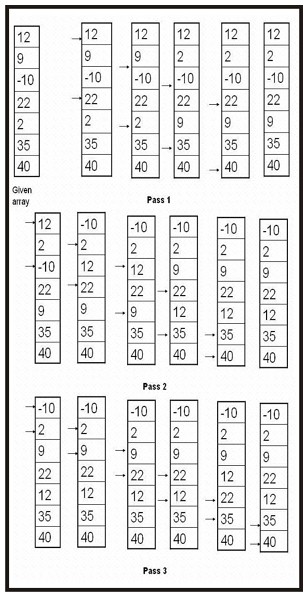
- Tem um pior caso de O(n2)

- Implementação difícil

- Não é estável

Shellsort

Este algoritmo é uma extensão do método Insertion Short proposto por Donald Shell em 1959. O algoritmo de inserção troca itens adjacentes quando está procurando o ponto de inserção na sequência destino. Se o menor item estiver na posição mais à direita no vetor, então o número de comparações e movimentações é igual a n-1 para encontrar o seu ponto de inserção. O ShellSort contorna este problema, permitindo trocas de registros distantes um do outro. De maneira geral ele passa várias vezes no vetor dividindo-o em vetores menores, e nestes são aplicados o algoritmo de ordenação por inserção tradicional. Dentre os programas de ordenação interna que tem ordem de complexidade quadrática, o ShellSort é o mais eficiente.



O algoritmo procede da seguinte forma:

1. Zera os valores das variáveis de medição através do método ClearAll();

2. Inicia a contagem de tempo com a função start();

3. O algoritmo usa uma variável auxiliar denominada distância de comparação(h);

4. O valor de h é inicializado com um valor próximo de n=2;

5. A troca é feita entre elementos que estão distanciados h posições (se estiverem fora de ordem);

6. Após todas as trocas de elementos cuja distância é h, o valor de h deve ser reduzido: conforme Equação 2.7;

7. O algoritmo é repetido até que a distância de comparação h seja igual a um (h = 1);

8. Para h = 1 (ultima passada) é executado o algoritmo de inserção;

9. A cada comparação incrementa a variável mComparations e a cada movimentação incrementa a variável mMoviments;

10. Pausa a contagem de tempo e calcula o tempo gasto armazenando o valor na variável mTime;

Vantagens

- Código Simples

- Interessante para arquivos de tamanho moderado

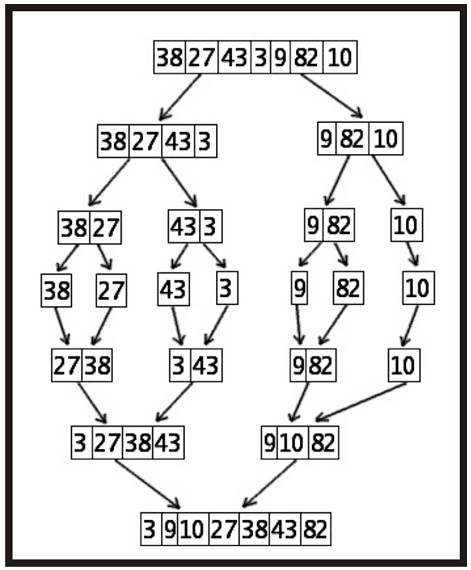
Desvantagens

- Algoritmo não estável

- Tempo de execução sensível à ordem inicial do arquivo.

Merge Sort

É outro algoritmo de ordenação do tipo dividir para conquistar. Sua ideia básica é criar uma sequência ordenada a partir de duas outras também ordenadas. Para isso, ele divide a sequência original em pares de dados, ordena-as; depois as agrupa em sequências de quatro elementos, e assim por diante, até ter toda a sequência dividida em apenas duas partes.



Os três passos úteis dos algoritmos dividir para conquistar, ou divide and con-quer, que se aplicam ao Merge Sort são:

1. Dividir: Dividir os dados em subsequências pequenas;

2. Conquistar: Classificar as duas metades recursivamente aplicando o merge Sort;

3. Combinar: Juntar as duas metades em um único conjunto já classificado.

O algoritmo procede da seguinte maneira:

1. É criado um vetor auxiliar do tamanho do intervalo;

2. Copia os valores do intervalo para o vetor auxiliar, de forma ordenada;

3. Copia o vetor auxiliar para o intervalo correspondente ao vetor a ser ordenado.

Vantagens

- Passível de ser transformado em estável

- Fácil Implementação

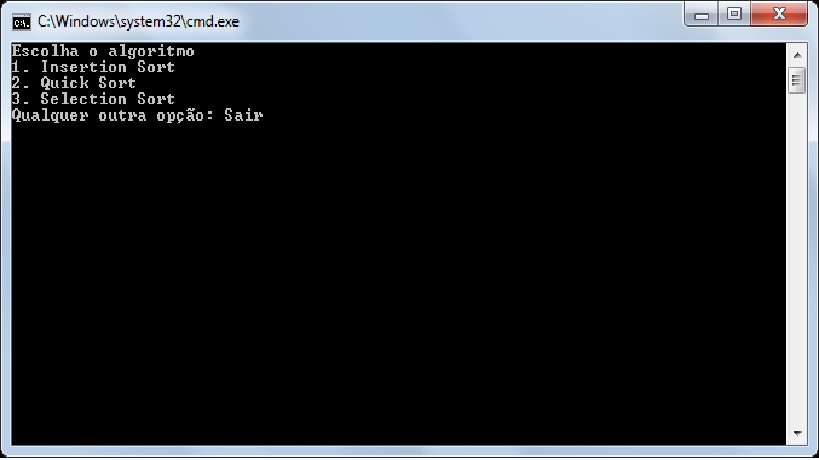
- Complexidade O (n log n)

Desvantagens

- Utiliza memória auxiliar

- Mais lento que o Heapsort

Tutorial sobre Aplicação



Tutorial do Programa de Ordenação

Este programa tem como função ordenar 100 números aleatoriamente, os números são criados através de uma função chamada Random, ou seja, você não precisa digitar 100 número para depois ele ordenar. Ele cria automaticamente.

Em seguida você deve escolher um dos 3 métodos de Ordenação:

* Caso queira escolher o método de Ordenação Insertion Sort digite 1 e aperte enter;
* Caso queira escolher o método de Ordenação Quick Sort digite 2 e aperte enter;
* Caso queira escolher o método de Ordenação Selection Sort digite 3 e aperte enter;

No momento que você escolher um desse métodos ele aparecera abaixo o lista ordenada dos 100 números conforme o método que você escolher. Para sair do programa basta digitar qualquer letra que não seja 1, 2 ou 3 e apertar enter.