Classificação por Seleção - selection sort

Outro método também simples de ordenação é a ordenação por seleção.

Princípio de funcionamento:

1. Selecione o menor item do vetor (ou o maior).
2. Troque-o com o item que está na primeira posição do vetor.

Repita estas duas operações com os n-1 itens restantes, depois com os n-2 itens, até que reste apenas um elemento.

Classificação por Seleção - selection sort

A ordenação por seleção consiste, em cada etapa, em selecionar o maior (ou o menor) elemento e colocá-lo em sua posição correta dentro da futura lista ordenada.

Durante a aplicação do método de seleção a lista com *n* registros fica decomposta em duas sub listas, uma contendo os itens já ordenados e a outra com os restantes ainda não ordenados.

- No início a sub lista ordenada é vazia e a outra contém todos os demais.
- No final do processo a sub lista ordenada apresentará (n-1) itens e a outra apenas 1.

UNIVASF

Classificação por Seleção - selection sort

As etapas (ou varreduras) como já descrito consiste em buscar o maior (ou menor) elemento da lista não ordenada e colocá-lo na lista ordenada.

Para uma melhor compreensão trabalharemos com um exemplo, visando demonstrar o resultado das etapas da ordenação de um vetor de inteiros, consideraremos o ordenamento crescente dos elementos e selecionaremos em cada etapa o maior elemento do subvetor não ordenado.

UNIVASF

Classificação por Seleção - selection sort

Resultado das Etapas					
Etapa	X[1]	X[2]	X[3]	X[4]	X[5]
0	5	9	1	4	3
1	{5	3	1	4}	{9}
2	{4	3	1}	{5	9}
3	{1	3}	{4	5	9}
4	{1}	{3	4	5	9}

Classificação por Seleção - selection sort

Exemplo:

	1	2	3	4	5	6
Chaves Iniciais	O	R	D	E	N	A
I=1	A	R	D	E	N	O
I =2	A	D	R	E	N	O
I=3	A	D	E	R	N	O
I=4	A	D	E	N	R	O
I=5	A	D	E	N	O	R

Nota: As Chaves em vermelho sofreram uma troca entre si

Classificação por Seleção - selection sort

Com base no que foi visto, construa uma função, em C, que recebe um vetor de inteiros e o número de elementos neste vetor. Esta função deve ordenar o vetor implementando o selection sort.

Classificação por Seleção - selection sort

```
void Selecao(int m,int x[])
{
 int aux,j,i,maior;
 for(i=0;i<m-1;i++)
 {
        maior=0;
        for  (j=0;j<m-i;j++)
          if (x[j] > x[maior])
            maior=j;

        aux=x[m-i-1];
        x[m-i-1]=x[maior];
        x[maior]=aux;
 }
}
```

Qual a complexidade do algoritmo?
$O(n^2)$

Classificação por Seleção - selection sort

Comentários finais:

- Este é um algoritmo a ser utilizado para tabelas com poucos registros.
- O fato da tabela já estar ordenada não ajuda em nada pois o custo continua quadrático.
- Este é um exemplo de um método não estável, pois ele nem sempre deixa os registros com chaves iguais na mesma posição relativa.

UNIVASF

Classificação por Inserção - insertion sort

- Método preferido dos jogadores de **cartas.**
- O jogador vai recebendo cartas uma por uma, e inserindo-as na posição adequada em sua mão, fazendo com que as cartas permaneçam ordenadas durante todo o jogo.
- ou seja,
  - Em cada passo, a partir de i=2, o i-ésimo item da seqüência fonte é apanhado e transferido para a seqüência destino, sendo inserido no seu lugar apropriado.

UNIVASF

Classificação por Inserção - insertion sort

A proposta da ordenação por inserção é a seguinte:

Para cada elemento do vetor **faça**

Inseri-lo na posição que lhe corresponde;

Um processo *in situ*, chamado **inserção direta**, pode ser assim descrito:

Considerar o subvetor ordenado v[0..k - 1];

Para j de k até n-1 faça

Inserir v[j] na sua posição em v [0..k - 1];

Classificação por Inserção - insertion sort

Para encontrar o lugar de v[k], basta comparar as chaves de índices [0..k-1] até encontrar uma chave v[k_ins] que seja maior que ele. v[k_ins] será a sucessora de v[k] depois deste ser inserido no subvetor ordenado (i.e., depois de ser localizado). O problema que surge agora é: como arranjar lugar em v[0..k - 1] para o valor v[k]?

Bem, como v[k] vai deixar seu lugar, este pode ser ocupado pelo elemento v[k-1], ao se fazer avançar uma posição para frente todo o subvetor v[k_ins..k – 1].

Classificação por Inserção - insertion sort

O processo começa adotando-se o subvetor ordenado v[0..0], e passando-se a inserir os elementos de índice 1..n-1. Para melhorar um pouco o desempenho, inicia-se a pesquisa a partir da posição k, ao mesmo tempo que se vai deslocando os elementos v[k_ins..k - 1]. A busca da posição de inserção deve prosseguir enquanto v[i $\in$ 0..k-1] > v[k]. Se a chave v[k] é menor que qualquer chave v[0.. k -1], acaba-se percorrendo o vetor até o seu início, correndo-se o risco de um índice inválido (-1). Por isso, o critério de parada deve incluir uma segunda condição: k_ins >= 0.

Classificação por Inserção - insertion sort

Observe no exemplo a seguir o comportamento de um vetor submetido á classificação por inserção direta. Em cada linha é listado um vetor depois de uma passagem completa sobre o mesmo, estando sublinhado o subvetor [0..k-1] e em negrito o elemento nele inserido por último.

Vetor original:	75 25 95 87 64 59 86 40 16 49
passagem	**vetor resultante**
1	**25** 75 95 87 64 59 86 40 16 49

passagem	vetor resultante
2	25 75 **95** 87 64 59 86 40 16 49
3	25 75 **87** 95 64 59 86 40 16 49
4	25 **64** 75 87 95 59 86 40 16 49
5	25 **59** 64 75 87 95 86 40 16 49
6	25 59 64 75 **86** 87 95 40 16 49
7	25 **40** 59 64 75 86 87 95 16 49
8	**16** 25 40 59 64 75 86 87 95 49
9	16 25 40 **49** 59 64 75 86 87 95

Com base no que foi discutido, codifique uma função que receba um vetor (de inteiros) e o número de elementos no mesmo e através da inserção direta ordene de forma crescente os elementos do vetor.

```
void insercao_direta (int *v, int n)
{
   int k_ins, k, elemento;
   for (k=1;k<n;k++)
   {
      k_ins=k-1;
      elemento=v[k];
      while (k_ins>=0 && v[k_ins]>elemento)
         v[k_ins+1]=v[k_ins--];
      v[k_ins+1]=elemento;
   }
}
```

Classificação por Inserção - insertion sort

Uma análise do algoritmo de inserção direta confirma que ele é **O**(n^2) o laço externo força a execução n-1 vezes do laço interno que, por sua vez, poderá ser executado até n-1 vezes (quando o elemento a ser inserido é o menor que de todos os contidos no subvetor ordenado). O pior caso ocorre quando o vetor está invertido: os elementos extremos têm de atravessar todo o vetor para relocalização. O melhor caso corresponde ao vetor ordenado, para o qual não são necessários deslocamentos.

UNIVASF

Classificação por Inserção - insertion sort

O caso médio ocorre quando cada chave está a n/2 posições de sua posição final, exigindo então $n^2/2$ movimentos de chave.

O método da inserção é o método a ser utilizado quando o arquivo está “quase" ordenado.

É também um bom método quando se deseja adicionar uns poucos itens a um arquivo já ordenado e depois obter um outro arquivo ordenado: neste caso o custo é linear.

Observa-se também que o algoritmo proposto implementa o método *estável*.

Classificação por Inserção - insertion sort

O processo de inserção direta pode ser melhorado utilizando-se do processo de busca binária para localizar a posição de inserção de v[k] no subvetor v[0..k-1].

Outra forma de melhoria do processo de inserção direta é a aplicação do método sobre listas encadeadas, evitando assim a necessidade de deslocamento dos elementos para uma posterior inserção.

UNIVASF

Classificação por Particionamento

O método de particionamento é um caso de aplicação do princípio da divisão e conquista: classificar dois vetores de tamanho n/2 é mais fácil que classificar um vetor de tamanho n.

O método basicamente se resume em:

i. particionar o vetor;
ii. classificar cada partição.

Classificação por Particionamento - quicksort

Um processo que trabalha com base nessa proposta é o *quicksort*, assim denominado por seu inventor, C. A. R. Hoare.

É um processo *in situ* e seu caso trivial é a partição de tamanho 1.

Pode ser visto, como uma melhoria do método das trocas, buscando reposicionar primeiro as chaves mais distantes do seu lugar final.

Classificação por Particionamento - quicksort

Funda-se em separar o vetor em duas partições, delimitadas por uma certa chave, dita *pivô*, tais que numa das partições estejam todas as chaves menores ou iguais ao pivô e na outra todas as chaves maiores que ele. Em seguida, reclassifica-se cada partição.

Para uma melhor compreensão vamos analisar um exemplo (consideraremos o primeiro elemento da partição como o pivô, com o objetivo de facilitar o processo).

UNIVASF

Classificação por Particionamento - quicksort

Se um vetor inicial for dado como:

25 57 48 37 12 92 86 33

e o pivô (25) for colocado na posição correta o vetor resultante será:

12 25 57 48 37 92 86 33

agora, o problema inicial foi decomposto na classificação de dois subvetores:

(12) e (57 48 37 92 86 33)

O primeiro já está classificado, uma vez que tem apenas um elemento. O processo se repete para classificar o segundo subvetor.

Classificação por Particionamento - quicksort

Agora, o vetor pode ser visualizado como:

12 25 (57 48 37 92 86 33)

onde os parênteses encerram os subvetores que ainda serão classificados. Repetindo o processo sobre o subvetor a ordenar teremos:

12 25 (48 37 33) 57 (92 86)

e as aplicações posteriores resultam em:

12 25 (37 33) 48 57 (92 86)

12 25 (33) 37 48 57 (92 86)

12 25 33 37 48 57 (92 86)

12 25 33 37 48 57 (86) 92

12 25 33 37 48 57 86 92

Classificação por Particionamento - quicksort

Os observadores mais atentos devem ter percebido que o quicksort pode ser melhor compreendido definido recursivamente.

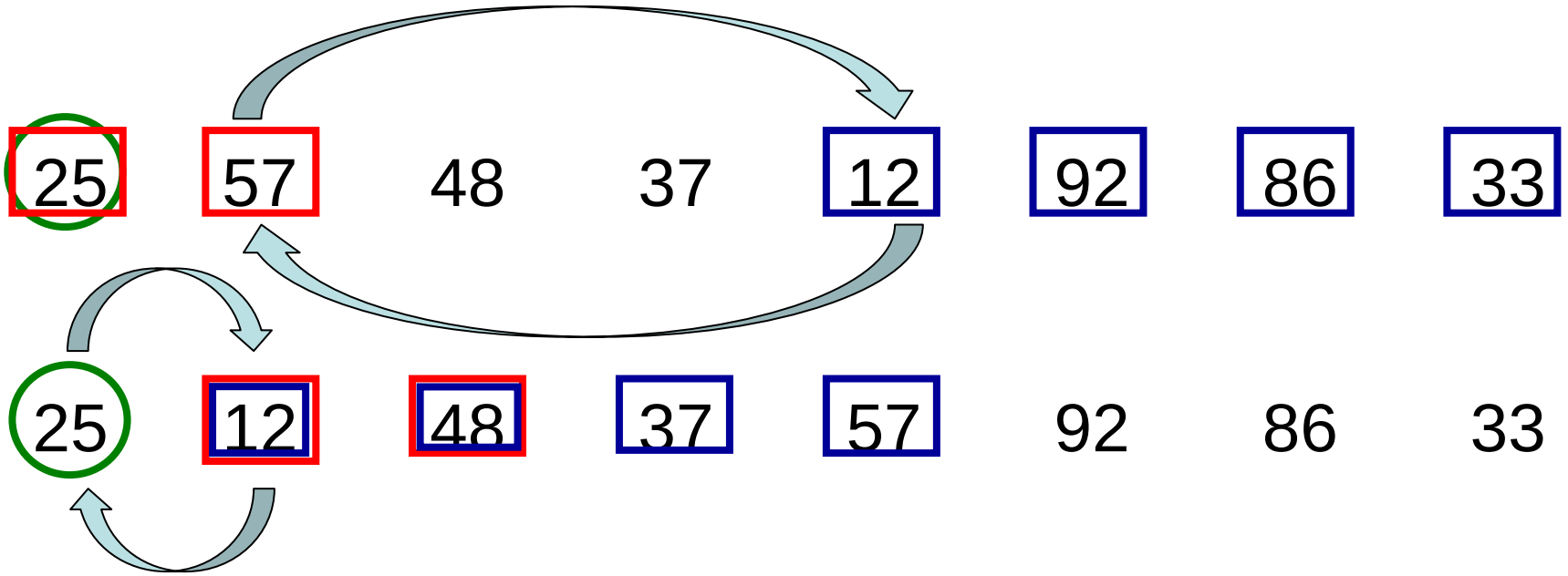
É interessante observar que no método apresentado os elementos após o particionamento aparecem na mesma ordem relativa em que apareciam no vetor original. Entretanto, este método de particionamento é ineficiente de se implementar. Veremos agora um método eficiente de se implementar o particionamento.

Classificação por Particionamento - quicksort

Para tanto, com um cursor, digamos “esq”, percorre-se o vetor da esquerda para a direita, até se localizar um elemento maior que o pivô. Com outro cursor, “dir”, percorre-se o vetor da direita para a esquerda, até se encontrar um elemento menor ou igual ao pivô, esses valores são intercambiados caso “dir” seja maior que “esq”. Repete-se este processo até os setores perscrutados esgotarem o vetor, o que se pode detectar pelo encontro dos cursores, ou seja, quando esq > dir. Neste momento, troca-se o valor do pivô pelo valor do elemento indexado por dir que é dito “ponto de partição”. O qual delimitará as partições a serem reclassificadas.

Classificação por Particionamento - quicksort

Este processo pode ser observado no exemplo:



Legenda: Pivô V[esq] V[dir]

Classificação por Particionamento - quicksort

Exercício:

Com base no que foi discutido implemente a função particionar. A qual recebe um vetor, os índices do limite inferior e superior de um subvetor pertencente a este e retorna o índice do ponto de particionamento.

UNIVASF