

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 sequência fonte é apanhado e transferido para a sequência destino, sendo inserido no seu lugar apropriado.

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 $v[1..k]$

Para j de $k+1$ até n faça

Inserir $v[j]$ na sua posição em $v [1..k]$

Classificação por Inserção - insertion sort

Para encontrar o lugar de $v[k]$, basta comparar as chaves de índices $[1..k-1]$ 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 $v[1..k - 1]$ 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 $v[1..1]$, e passando-se a inserir os elementos de índice $2..n$.

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 $v[i \in 1..k-1] > v[k]$.

Se a chave $v[k]$ é menor que qualquer chave $v[1.. k - 1]$, acaba-se percorrendo o vetor até o seu início, correndo-se o risco de um índice inválido (0). Por isso, o critério de parada deve incluir uma segunda condição: $k_ins \geq 1$.

Classificação por Inserção - insertion sort

Observe no exemplo a seguir o comportamento de um vetor submetido à classificação por inserção direta. Em cada linha é listado um vetor depois de uma passagem completa sobre o mesmo, estando sublinhado o subvetor $[1..k-1]$ 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	<u>25</u> 75 95 87 64 59 86 40 16 49

passagem

vetor resultante

2	<u>25 75 95</u> 87 64 59 86 40 16 49
3	<u>25 75 87 95</u> 64 59 86 40 16 49
4	<u>25 64 75 87 95</u> 59 86 40 16 49
5	<u>25 59 64 75 87 95</u> 86 40 16 49
6	<u>25 59 64 75 86 87 95</u> 40 16 49
7	<u>25 40 59 64 75 86 87 95</u> 16 49
8	<u>16 25 40 59 64 75 86 87 95</u> 49
9	<u>16 25 40 49 59 64 75 86 87 95</u>

Com base no que foi discutido, codifique um módulo que receba como parâmetros um vetor (de inteiros), com no máximo cem elementos, e o número de elementos no mesmo. Através da inserção direta o módulo deve ordenar de forma crescente os elementos contidos no vetor.

```
procedimento insercao_direta(var vet:vetor[1..100] de
    inteiro; n: inteiro)
var k_ins, k, elemento: inteiro
inicio
    para k de 2 ate n faca
        k_ins<-k-1
        elemento<-vet[k]
        enquanto ((k_ins>=1) e (vet[k_ins]>elemento)) faca
            vet[k_ins+1] <- vet[k_ins]
            k_ins <- k_ins-1
        fimenquanto
        vet[k_ins+1]<-elemento
    fimpara
fimprocedimento
```

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*, de forma 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).

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 se 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.

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.

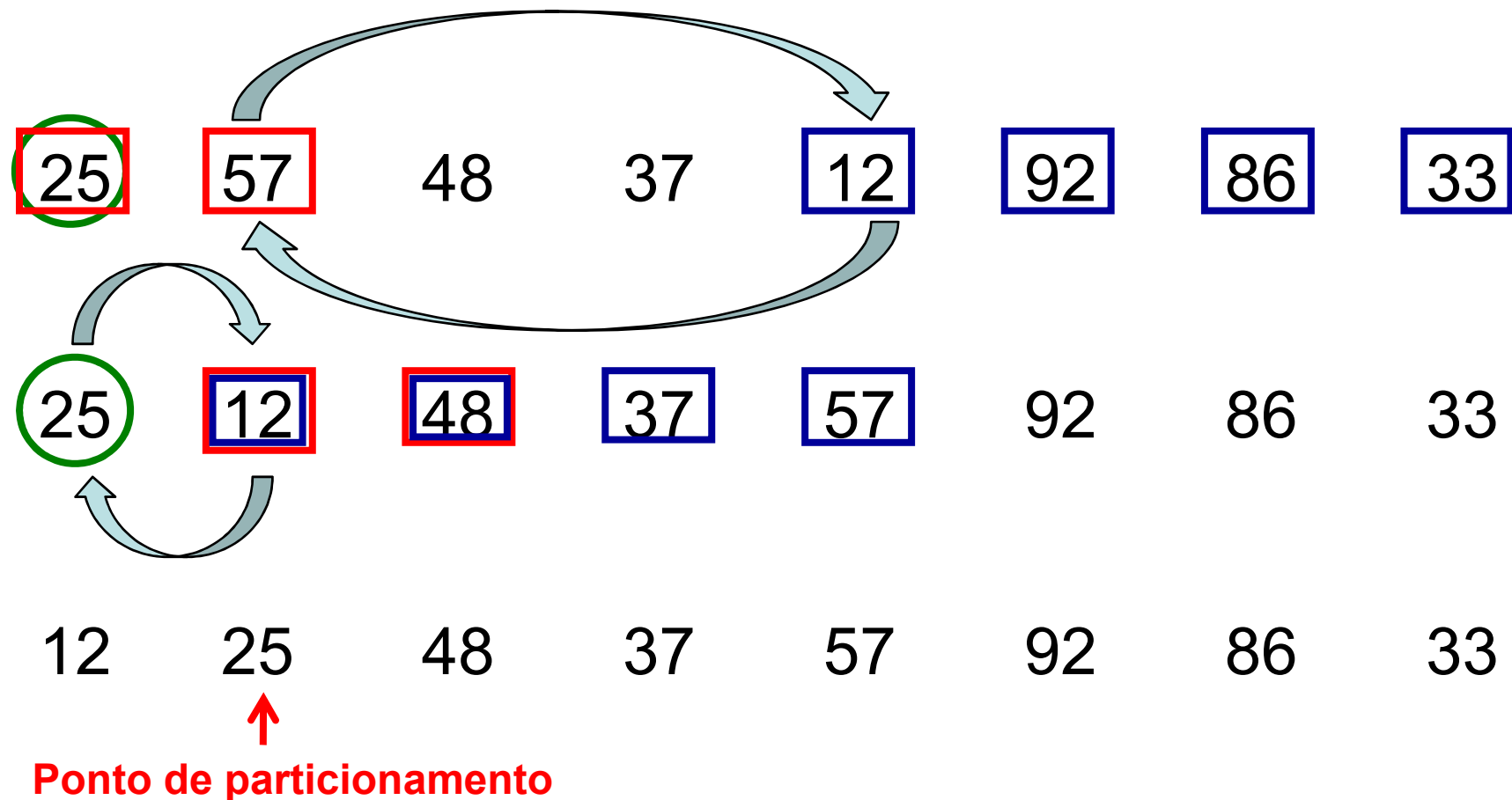
Com outro cursor, “dir”, percorre-se o vetor da direita para a esquerda, até se encontrar um elemento menor ou igual ao pivô, esses valores são intercambiados caso “dir” seja maior que “esq”.

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:



Classificação por Particionamento - quicksort

Exercício 56:

Com base no que foi discutido implemente um módulo denominado `particionar`. O qual recebe, como parâmetros, um vetor de inteiros com no máximo 100 elementos, os índices do limite inferior e superior de um subvetor pertencente a este e retorna o índice do ponto de particionamento.


```

funcao particionar (var v: vetor [1..100] de inteiro; ii: inteiro;
  is: inteiro): inteiro
var esq, dir, pivo, aux: inteiro
inicio
  esq <- ii
  dir <- is
  pivo <- v[ii]
  enquanto (esq<dir) faca
    enquanto ((v[esq]<=pivo) e (esq<is)) faca
      esq <- esq + 1
    fimenquanto
    enquanto (v[dir]>pivo) faca
      dir <- dir - 1
    fimenquanto
    se (esq<dir) entao
      aux <- v[esq]
      v[esq] <- v[dir]
      v[dir] <- aux
    fimse
  fimenquanto
  v[ii] <- v[dir]
  v[dir] <- pivo
  retorne (dir)

```

Classificação por Particionamento - quicksort

Exercício 57:

Agora, construa um módulo recursivo que receba, como parâmetros, um vetor de inteiros, com no máximo 100 elementos, e os índices do primeiro e do último elemento contidos no vetor. O módulo em questão deve ordenar o vetor implementando o método quicksort.

```
procedimento quicksort (var v: vetor
    [1..100] de inteiro; ind_pri_ele: inteiro;
    ind_ult_ele: inteiro)
var pont_part: inteiro
inicio
    se (ind_ult_ele-ind_pri_ele>0) entao
        pont_part <- particionar(v, ind_pri_ele,
            ind_ult_ele)
        quicksort (v, ind_pri_ele, pont_part-1)
        quicksort (v, pont_part+1, ind_ult_ele)
    fimse
fimprocedimento
```