QuickSort - Explicação

Temos um vetor [0…n-1] com n elementos. Escolhemos arbitrariamente um destes elementos para ser o pivô. Por exemplo, podemos definir o pivô como sendo sempre o último elemento, ou o primeiro, ou o segundo… enfim, qualquer elemento. Mas sem repetir a escolha.

O que faremos basicamente é mandar todos os valores maiores ou iguais ao pivô para a direita dele e os menores para a esquerda. E faremos isso com todos os "subvetores" do vetor principal até todos estiverem ordenados.

Imagine o seguinte, temos o vetor:

7 4 5 2 27 -1 9 6 0

O pivô nesse caso, escolhido arbitrariamente, será o número 7. Confira a seguir a parte principal do código:

```
int particiona (int v[], int ini, int fim) {
 int i, j, x;

 i = ini;
 j = fim+1;
 x = v[ini];
 while (1){
   while (v[++i] < x)
     if (i == fim)
 break;

   while (v[--j] > x)
     if (j == ini)
 break;

   if (i >= j)
     break;
   troca(v, i, j);
 }
 troca(v, ini, j);
 return j;
}

void quicksort (int v[], int ini, int fim) {
 int x;
```

```
 if (ini < fim){
   x = particiona(v, ini, fim);
   quicksort(v, ini, x - 1);
   quicksort(v, x + 1, fim);
 }
}
```

Primeiro chamamos a função **particiona** que define o pivô (x) como sendo o primeiro elemento, que, nosso exemplo, é o 7. Utilizaremos duas variáveis aqui, i e j, com i recebendo inicialmente o valor 0 e j começando em n.

Entramos em um loop infinito (while(1) com outros while dentro. No primeiro, vamos percorrendo o vetor a partir da posição 1, enquanto o elemento daquela posição for menor que o valor do pivô. Quando for maior, i fica com o valor da posição deste elemento. Há uma quebra quando i chega no valor (n-1).

O mesmo ocorre para j, só que de maneira inversa, j começa no valor n-1 e vai decrescendo. Quando encontramos um valor menor que o pivô, j salva o número da posição deste elemento e o loop é quebrado.

Ao final, se i tiver se igualado ou tornado-se maior que o valor de j quebramos o loop infinito. Caso isso não aconteça, trocamos de posição os elementos da posição inicial, ou seja, o pivô, com o elemento da posição j.

Por fim, retornamos a função quicksort que repete esse procedimento mas separando o vetor original em 2: O subvetor anterior ao elemento x e o subvetor posterior ao elemento x;

Fazendo isso, vamos ordenando os elementos.