Quicksort

Marcelo K. Albertini

20 de Novembro de 2013

Quicksort simplificado

```
1  int[] quicksort(vetor) {
2    if (tamanho(vetor) <= 1) {
3      return(vetor) // vetor já ordenado
4    }
5  // pivot é elemento de referencia para ordenar
6    pivot = escolher_E_remover(vetor)
7  // criar vetores de elementos menores e maiores
8    vetorMenores = new int[tamanho(vetor)]
9    vetorMaiores = new int[tamanho(vetor)]
10
11   for (x in vetor) {
12       if (x <= pivot)
13           insereNoFim(x, vetorMenores)
14       else
15           insereNoFim(x, vetorMaiores)
16   }
17   quicksort(vetorMenores) // ordenar os menores
18   quicksort(vetorMaiores) // ordenar os maiores
19   return(juntar(vetorMenores, pivot, vetorMaiores));
20 }
```

Escolha do pivot

```
1  pivot = escolher_E_remover(vetor)
```

- custo do algoritmo depende da escolha do pivot
- possibilidades
  - o primeiro/último elemento
  - aleatório
  - o elemento do meio
  - a mediana entre o primeiro, meio e último

Quicksort simplificado: problemas

- uso de mais memória: vetores auxiliares
- escolha do pivot: qual escolha leva a menor complexidade?

Quicksort sem memória extra

Ideia

1. **Desordenar** o vetor
2. **Particionar** tal que para algum elemento na posição `j` (pivot)
   - valor em `v[j]` está na posição correta
   - todos os valores à esquerda de j são menores que `v[j]`
   - todos os valores à direita de j são maiores que `v[j]`
3. **ordenar** cada pedaço **recursivamente**, mas sem cópia do vetor

entrada	Q	U	I	C	K	S	O	R	T
desordenado	K	R	T	Q	S	O	I	U	C
partição	I	C	K	Q	U	R	T	S	O
ordena esq.	C	I	K	Q	U	R	T	S	O
ordena dir.	C	I	K	O	Q	R	S	T	U
resultado	C	I	K	O	Q	R	S	T	U

Desordenação

Desordenação com complexidade de tempo $\Theta(n)$ e espaço $\Theta(n)$ pode ser feita com o algoritmo de desordenação de Knuth.

```
1  public static void shuffle(int[] v) {
2
3    Random r = new Random(System.currentTimeMillis());
4    int aux;
5
6    // desordena um elemento por vez
7    for (int i = 0; i < v.length; i++) {
8      int ir = r.nextInt(i+1); // sorteio aleatorio
9
10     aux = v[i]; // troca com posicao aleatoria
11     v[i] = v[ir];
12     v[ir] = aux;
13    }
14 }
```

Partição

Objetivo

Dividir vetor em duas regiões separadas pelo `pivot`.

- A região anterior ao `pivot` consiste de elementos menores ou iguais a ele.
- A região posterior ao `pivot` consiste de elementos maiores a ele.

Quardamos duas variáveis de índice: da esquerda `i` e da direita `j`.

$v[p]$ é o elemento pivot.

Antes:

$v[p]$	$v[\ldots]$

Durante:

$v[p]$	$v[\ldots] <= v[p]$	$v[i] \ldots v[j]$	$v[\ldots] > v[p]$

Depois:

$v[\ldots] <= v[p]$	$v[p]$	$v[\ldots] > v[p]$

Partição

Região do vetor a ser particionada é delimitada por `inf` e `sup`

```
1  particao(int v[], int inf, int sup) {
2    int i = inf, j = sup+1, aux;
3
4    while(true) {
5      while (v[++i] < v[inf]) // movimento da esq.
6          if (i == sup)
7              break;
8
9      while (v[inf] < v[--j]) // mov. da direita
10         if (j == inf)
11             break;
12
13     if (i >= j) break;
14
15     troca(v, i, j);
16   }
17
18   troca(v, inf, j); // troca posicao do pivot
19
20   return j; // retorna posicao do pivot
21 }
```

```
1  int particao(int v[], int
       inf, int sup) {
2  int i=inf, j=sup+1;
3
4  while(true) {
5   while (v[++i] < v[inf])
6    if (i == sup) break;
7
8   while (v[inf] < v[--j])
9    if (j == inf) break;
10
11  if (i >= j) break;
12
13  troca(v, i, j);
14 }
15
16 troca(v, inf, j);// pivot
17
18 return j; // retorna pivot
19 }
```

$p = 0$, $v[p] = 5$, $inf = 0$, $sup = 8$

5	6	7	8	5	4	3	2	9
5	6	7	8	5	4	3	2	9
5	2	7	8	5	4	3	6	9
5	2	7	8	5	4	3	6	9
5	2	3	8	5	4	7	6	9
5	2	3	8	5	4	7	6	9
5	2	3	4	5	8	7	6	9
5	2	3	4	5	8	7	6	9

Quicksort

```
1  public static void quicksort(int[] v, int n) {
2    shuffle(v); // desordenar é rápido
3    sort(v, 0, v.length-1);
4  }
5
6  public static void sort(int[] v, int inf, int sup) {
7    if (inf >= sup) {
8      return;
9    } else {
10     int j = particao(v, inf, sup); // ordena pivot
11     sort(v, inf, j-1); // ordena menores
12     sort(v, j+1, sup); // ordena maiores
13   }
14 }
```

$p = 0,\ v[p] = 5,\ inf = 0,\ sup = 8$

5	6	7	8	5	4	3	2	9
5	6	7	8	5	4	3	2	9
5	2	7	8	5	4	3	6	9
5	2	7	8	5	4	3	6	9
5	2	3	8	5	4	7	6	9
5	2	3	8	5	4	7	6	9
5	2	3	4	5	8	7	6	9
5	2	3	4	5	8	7	6	9

$p = 0,\ v[p] = 5,\ inf = 0,\ sup = 3$

5	2	3	4	5	8	7	6	9
4	2	3	5	5	8	7	6	9

$p = 0,\ v[p] = 4,\ inf = 0,\ sup = 2$

4	2	3	5	5	8	7	6	9
3	2	4	5	5	8	7	6	9

$p = 0,\ v[p] = 3,\ inf = 0,\ sup = 1$

3	2	4	5	5	8	7	6	9
2	3	4	5	5	8	7	6	9

$p = 5,\ v[p] = 8,\ inf = 5,\ sup = 8$

2	3	4	5	5	8	7	6	9
2	3	4	5	5	6	7	8	9

$p = 5,\ v[p] = 6,\ inf = 5,\ sup = 6$

2	3	4	5	5	6	7	8	9
2	3	4	5	5	6	7	8	9

Análise de complexidade

O custo de usar o quicksort em n é o custo de particionar esses elementos com αn operações mais o custo de aplicar o quicksort nos dois subvetores resultantes, com k e $n - k$ elementos.

Relação de recorrência

$$T(n) = T(k) + T(n - k) + \alpha n$$

Análise de pior caso

Pior caso ocorre quando o `pivot` for sempre o menor elemento do vetor. Ou seja, $k = 1$ em $T(n) = T(k) + T(n-k) + \alpha n$

Relação de recorrência: pior caso k=1

Divide array com $k = 1$	$T(n) = T(n-1) + T(1) + \alpha n$
Obtém eq. para $n-1$	$T(n) = [T(n-2) + T(1) + \alpha(n-1)] + \alpha n$
Reorganiza	$T(n) = T(n-2) + 2T(1) + \alpha(n-1+n)$

Análise de pior caso k=1: continuando

Para $k = 1$ $\qquad\qquad$ $T(n) = T(n-1) + T(1) + \alpha n$

Análise de pior caso k=1: continuando

Para $k = 1$

$$T(n) = T(n-1) + T(1) + \alpha n$$

Eq. de $n-1$

$$= [T(n-2) + T(1) + \alpha(n-1)] + \alpha n$$

Análise de pior caso k=1: continuando

Para $k = 1$	$T(n) = T(n-1) + T(1) + \alpha n$
Eq. de $n-1$	$= [T(n-2) + T(1) + \alpha(n-1)] + \alpha n$
Organiza	$= T(n-2) + 2T(1) + \alpha(n-1+n)$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k = 1 \quad & T(n) = T(n-1) + T(1) + \alpha n \\
\text{Eq. de } n-1 \quad & = [T(n-2) + T(1) + \alpha(n-1)] + \alpha n \\
\text{Organiza} \quad & = T(n-2) + 2T(1) + \alpha(n-1+n) \\
\text{Eq. de } n-2 \quad & = [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1) + \alpha n
\end{aligned}
$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k = 1 \quad & T(n) = T(n-1) + T(1) + \alpha n \\
\text{Eq. de } n-1 \quad & = [T(n-2) + T(1) + \alpha(n-1)] + \alpha n \\
\text{Organiza} \quad & = T(n-2) + 2T(1) + \alpha(n-1+n) \\
\text{Eq. de } n-2 \quad & = [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1) + \alpha n \\
\text{Organiza} \quad & = T(n-3) + 3T(1) + \alpha[(n-2) + (n-1) + n]
\end{aligned}
$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k = 1 \quad & T(n) = T(n-1) + T(1) + \alpha n \\
\text{Eq. de } n-1 \quad & = [T(n-2) + T(1) + \alpha(n-1)] + \alpha n \\
\text{Organiza} \quad & = T(n-2) + 2T(1) + \alpha(n-1+n) \\
\text{Eq. de } n-2 \quad & = [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1) + \alpha n \\
\text{Organiza} \quad & = T(n-3) + 3T(1) + \alpha[(n-2) + (n-1) + n] \\
\text{Eq. de } n-i \quad & = T(n-i) + iT(1) + \alpha[(n-i+1) + \ldots + (n-1) + n]
\end{aligned}
$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k=1 \quad & T(n) = T(n-1) + T(1) + \alpha n \\
\text{Eq. de } n-1 \quad & = [T(n-2) + T(1) + \alpha(n-1)] + \alpha n \\
\text{Organiza} \quad & = T(n-2) + 2T(1) + \alpha(n-1+n) \\
\text{Eq. de } n-2 \quad & = [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1) + \alpha n \\
\text{Organiza} \quad & = T(n-3) + 3T(1) + \alpha[(n-2) + (n-1) + n] \\
\text{Eq. de } n-i \quad & = T(n-i) + iT(1) + \alpha[(n-i+1) + \ldots + (n-1) + n] \\
\text{Soma} \quad & = T(n-i) + iT(1) + \alpha \textstyle\sum_{j=0}^{i-1}(n-j)
\end{aligned}
$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k=1 \quad & T(n) = T(n-1) + T(1) + \alpha n \\
\text{Eq. de } n-1 \quad & = [T(n-2) + T(1) + \alpha(n-1)] + \alpha n \\
\text{Organiza} \quad & = T(n-2) + 2T(1) + \alpha(n-1+n) \\
\text{Eq. de } n-2 \quad & = [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1) + \alpha n \\
\text{Organiza} \quad & = T(n-3) + 3T(1) + \alpha[(n-2) + (n-1) + n] \\
\text{Eq. de } n-i \quad & = T(n-i) + iT(1) + \alpha[(n-i+1) + \ldots + (n-1) + n] \\
\text{Soma} \quad & = T(n-i) + iT(1) + \alpha \textstyle\sum_{j=0}^{i-1}(n-j) \\
\text{Vai até } i=n-1 \quad & = T(n-n+1) + (n-1)T(1) + \alpha \textstyle\sum_{j=0}^{n-1-1}(n-j)
\end{aligned}
$$

Análise de pior caso k=1: continuando

$$
\begin{aligned}
\text{Para } k=1 \quad & T(n) = T(n-1) + T(1) + \alpha n \\
\text{Eq. de } n-1 \quad & = [T(n-2) + T(1) + \alpha(n-1)] + \alpha n \\
\text{Organiza} \quad & = T(n-2) + 2T(1) + \alpha(n-1+n) \\
\text{Eq. de } n-2 \quad & = [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1) + \alpha n \\
\text{Organiza} \quad & = T(n-3) + 3T(1) + \alpha[(n-2) + (n-1) + n] \\
\text{Eq. de } n-i \quad & = T(n-i) + iT(1) + \alpha[(n-i+1) + \ldots + (n-1) + n] \\
\text{Soma} \quad & = T(n-i) + iT(1) + \alpha \textstyle\sum_{j=0}^{i-1}(n-j) \\
\text{Vai até } i=n-1 \quad & = T(n-n+1) + (n-1)T(1) + \alpha \textstyle\sum_{j=0}^{n-1-1}(n-j) \\
\text{Resultado} \quad & = nT(1) + \alpha[(\textstyle\sum_{j=1}^{n} j) - 1]
\end{aligned}
$$

Análise de pior caso k=1: continuando

Para $k = 1$	$T(n) = T(n-1) + T(1) + \alpha n$
Eq. de $n-1$	$= [T(n-2) + T(1) + \alpha(n-1)] + \alpha n$
Organiza	$= T(n-2) + 2T(1) + \alpha(n-1+n)$
Eq. de $n-2$	$= [T(n-3) + T(1) + \alpha(n-2)] + 2T(1) + \alpha(n-1) + \alpha n$
Organiza	$= T(n-3) + 3T(1) + \alpha[(n-2) + (n-1) + n]$
Eq. de $n-i$	$= T(n-i) + iT(1) + \alpha[(n-i+1) + \ldots + (n-1) + n]$
Soma	$= T(n-i) + iT(1) + \alpha \sum_{j=0}^{i-1}(n-j)$
Vai até $i = n-1$	$= T(n-n+1) + (n-1)T(1) + \alpha \sum_{j=0}^{n-1-1}(n-j)$
Resultado	$= nT(1) + \alpha[(\sum_{j=1}^{n} j) - 1]$
Só o somatório	$\sum_{j=1}^{n} j = (n+1)n/2$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$T(n) = 2T(n/2) + \alpha n$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$T(n) = 2T(n/2) + \alpha n$$

Para $n/4$ $$= 2(2T(n/4) + \alpha n/2) + \alpha n$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$
\begin{aligned}
& T(n) = 2T(n/2) + \alpha n \\
\text{Para } n/4 \quad & = 2(2T(n/4) + \alpha n/2) + \alpha n \\
\text{Organizar} \quad & = 4T(n/4) + 2\alpha n/2 + \alpha n
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$
\begin{aligned}
& T(n) = 2T(n/2) + \alpha n \\
\text{Para } n/4 \quad & = 2(2T(n/4) + \alpha n/2) + \alpha n \\
\text{Organizar} \quad & = 4T(n/4) + 2\alpha n/2 + \alpha n \\
& = 2^2 T(n/2^2) + 2\alpha n
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$
\begin{aligned}
& T(n) = 2T(n/2) + \alpha n \\
\text{Para } n/4 \quad & = 2(2T(n/4) + \alpha n/2) + \alpha n \\
\text{Organizar} \quad & = 4T(n/4) + 2\alpha n/2 + \alpha n \\
& = 2^2 T(n/2^2) + 2\alpha n \\
\text{Para } n/8 \quad & = 2^2[2(T(n/8) + \alpha n/8)] + 2\alpha n
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$
\begin{aligned}
& T(n) = 2T(n/2) + \alpha n \\
\text{Para } n/4 \quad & = 2(2T(n/4) + \alpha n/2) + \alpha n \\
\text{Organizar} \quad & = 4T(n/4) + 2\alpha n/2 + \alpha n \\
& = 2^2 T(n/2^2) + 2\alpha n \\
\text{Para } n/8 \quad & = 2^2[2(T(n/8) + \alpha n/8)] + 2\alpha n \\
\text{Organizar} \quad & = 2^3 T(n/2^3) + 3\alpha n
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$
\begin{aligned}
& & T(n) &= 2T(n/2) + \alpha n \\
\text{Para } n/4 & & &= 2(2T(n/4) + \alpha n/2) + \alpha n \\
\text{Organizar} & & &= 4T(n/4) + 2\alpha n/2 + \alpha n \\
& & &= 2^2 T(n/2^2) + 2\alpha n \\
\text{Para } n/8 & & &= 2^2[2(T(n/8) + \alpha n/8)] + 2\alpha n \\
\text{Organizar} & & &= 2^3 T(n/2^3) + 3\alpha n \\
\text{Para } n/2^k & & &= 2^k T(n/2^k) + k\alpha n
\end{aligned}
$$

Análise de melhor caso

Melhor caso ocorre quando o `pivot` for sempre o elemento que divide o vetor na metade. Ou seja, $k = n/2$ em $T(n) = 2T(n/2) + \alpha n$

Relação de recorrência: melhor caso k=n/2

$$
\begin{aligned}
& & T(n) &= 2T(n/2) + \alpha n \\
\text{Para } n/4 & & &= 2(2T(n/4) + \alpha n/2) + \alpha n \\
\text{Organizar} & & &= 4T(n/4) + 2\alpha n/2 + \alpha n \\
& & &= 2^2 T(n/2^2) + 2\alpha n \\
\text{Para } n/8 & & &= 2^2[2(T(n/8) + \alpha n/8)] + 2\alpha n \\
\text{Organizar} & & &= 2^3 T(n/2^3) + 3\alpha n \\
\text{Para } n/2^k & & &= 2^k T(n/2^k) + k\alpha n \\
\text{Até } n = 2^k, \text{ com } k = \log_2 n & & T(n) &= nT(1) + \alpha n \log_2 n
\end{aligned}
$$

Tratamento de números repetidos

Problema

O que acontece quando temos um grande quantidade de números repetidos?

É comum usar ordenação para juntar elementos iguais. E.g. para eliminá-los, para comparar dados relacionados, para contar repetidos.

Problemas de desempenho

Em 1990, um programador de C percebeu que o `qsort()` estava rodando no pior caso $O(n^2)$. Erro comum em livros didáticos que acontece ao particionar números repetidos.

$p = 0,\ v[p] = 1,\ inf = 0,\ sup = 11$

1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	2	1	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1

$p = 0,\ v[p] = 1,\ inf = 0,\ sup = 4$

1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1

1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1

$p = 0,\ v[p] = 1,\ inf = 0,\ sup = 1$

1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1

$p = 3,\ v[p] = 1,\ inf = 3,\ sup = 4$

1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1

$p = 6,\ v[p] = 2,\ inf = 6,\ sup = 11$

1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	2	1	1	1	1
1	1	1	1	1	1	2	1	1	1	1	2
1	1	1	1	1	1	1	1	1	1	2	2

$p = 6,\ v[p] = 1,\ inf = 6,\ sup = 9$

1	1	1	1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	1	1	1	1	2	2

$p = 6,\ v[p] = 1,\ inf = 6,\ sup = 7$

1	1	1	1	1	1	1	1	1	1	2	2
1	1	1	1	1	1	1	1	1	1	2	2

Tratamento de números repetidos

- Método de particionamento em três partes de Dijkstra
  - Valores entre posições `me` e `ma` são iguais a `p`
  - Valores antes da posição `me` são todos menores que `p`
  - Valores depois da posição `ma` são todos maiores que `p`

Antes:

$v[inf] = p$	$\ldots v[sup]$

Depois:

$v[inf \ldots me-1] < p$	$v[me \ldots ma] = p$	$p < v[ma+1 \ldots sup]$

p é o elemento pivot. *inf* e *sup* delimitam região do vetor a ser particionada. *me* e *ma* identificam região de elementos iguais ao pivot p

Método de particionamento de Dijkstra

- Seja o pivot p o item na posição $v[me]$
- Ir da esquerda para a direita com um índice i
  - se $(v[i] < p)$: trocar $v[me]$ com $v[i]$ e incrementar me e i
  - se $(v[i] > p)$: trocar $v[ma]$ com $v[i]$ e decrementar ma
  - se $(v[i] = p)$: incrementar i

Invariante

$v[inf \ldots me-1] < p$	$v[me \ldots i-1] = p$	$v[i \ldots ma]?$	$p < v[ma+1 \ldots sup]$

Método de particionamento de Dijkstra

```
1  sort(int[] v, int inf, int
       sup) {
2
3    if (sup <= inf) return;
4
5    int me = inf;
6    int ma = sup;
7    int p = v[me];
8    int i = me;
9
10   while (i <= ma) {
11     if (v[i] < p)
12       troca(v, me++, i++);
13     else if (v[i] > p)
14       troca(v, i, ma--);
15     else i++;
16   }
17   sort(v, inf, me-1);
18   sort(v, ma+1, sup);
19 }
```

1	2	1	1	1	2	1	1	2

inf = 0, sup = 8, p = 1

v[1] > p, troca v[8] com v[1]

1	2	2	1	1	1	2	1	1
1	1	2	1	1	1	2	1	2

v[2] > p, troca v[7] com v[2]

1	1	2	1	1	1	2	1	2
1	1	1	1	1	1	2	2	2

v[6] > p, troca v[6] com v[6]

1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	2	2	2
1	1	1	1	1	1	2	2	2

inf = 6, sup = 8, p = 2

1	1	1	1	1	1	2	2	2

1	2	3	1	1	2	1	1	2

inf = 0, sup = 8, p = 1

v[1] > p, troca v[8] com v[1]

1	2	1	3	1	1	1	2	2

1	2	1	3	1	1	1	2	2

v[1] > p, troca v[7] com v[1]

1	2	1	3	1	1	1	2	2

1	2	1	3	1	1	1	2	2

v[1] > p, troca v[6] com v[1]

1	2	1	3	1	1	1	2	2

1	1	1	3	1	1	2	2	2

v[3] > p, troca v[5] com v[3]

1	1	1	3	1	1	2	2	2

1	1	1	1	1	3	2	2	2

1	1	1	1	1	3	2	2	2

inf = 5, sup = 8, p = 3

v[6] < p, troca v[5] e v[6]

1	1	1	1	1	3	2	2	2

1	1	1	1	1	2	3	2	2

v[7] < p, troca v[6] e v[7]

1	1	1	1	1	2	3	2	2

1	1	1	1	1	2	2	3	2

v[8] < p, troca v[7] e v[8]

1	1	1	1	1	2	2	3	2

1	1	1	1	1	2	2	2	3

1	1	1	1	1	2	2	2	3

inf = 5, sup = 7, p = 2

1	1	1	1	1	2	2	2	3