Algoritmos e Estruturas de Dados

Quicksort

Prof. Maiquel de Brito
maiquel.b@ufsc.br

Quicksort

Proposto por Hoare em 1960 e publicado em 1962

Apresenta bom desempenho em várias situações

Algoritmo

Dividir para conquistar:

Particionar o conjunto de N itens em problemas menores e ordenar as várias partes independentemente

- Uma vez efetuada a partição, cada uma das partes pode ser ordenada pelo mesmo algoritmo (de forma recursiva)
- A parte mais delicada do quicksort é o processo de partição
- O vetor v é rearranjado por meio da escolha arbitrária de um pivô *p*
- Os elementos do vetor v são reposicionados de forma que:
  - ***Partição esquerda: chaves ≤ p***
  - ***Partição direita: chaves ≥ p***

Particionamento

Algoritmo

- Escolha arbitrariamente o pivô **p**
- Percorra a partir da esquerda encontrar v[**e**] ≥ **p**
- Percorra a partir da direita encontrar v[**d**] ≤ **p**
- Troque v[**e**] com v[**d**]
- Repita os passos anteriores até que **e** e **d** se cruzem (**d** < **e**)

Particionamento

Ao final do particionamento

- Pelo menos um elemento (pivô) está em sua posição final
- Elementos na partição esquerda são menores
- Elementos na partição da direita são maiores que o pivô

- Selecionando o pivô como v[(esq+dir)/2] = 5 (esq = 0, dir = 8)

	esq				p				dir
		e					d		
Passo 1	2	7	4	1	5	3	0	8	6
		e					d		
Passo 2	2	0	4	1	5	3	7	8	6
					e	d			
Passo 3	2	0	4	1	5	3	7	8	6
					d	e			
Passo 4	2	0	4	1	3	5	7	8	6

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

e = 0 v[e] = 2 pivô = 5
d = 8 v[d] = 6

esq				p				dir
e								d
2	7	4	1	5	3	0	8	6

v[e] é menor que o pivô → verifica v[e+1]

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

$e = 1$ $v[e] = 7$ pivô $= 5$

$d = 8$ $v[d] = 6$

esq				p				dir
	e							d
2	7	4	1	5	3	0	8	6

v[e] é maior que o pivô → procura elemento menor que o pivô à direita

Procurando o primeiro número à direita do pivô que seja menor ou igual a ele

$e = 1$ $v[e] = 7$ pivô = 5
$d = 8$ $v[d] = 6$

esq				p				dir
	e							d
2	7	4	1	5	3	0	8	6

v[d] é maior que o pivô → verifica v[d-1]

Procurando o primeiro número à direita do pivô que seja menor ou igual a ele

$e = 1$ $v[e] = 7$ pivô = 5

$d = 7$ $v[d] = 8$

esq				p				dir
	e						d	
2	7	4	1	5	3	0	8	6

v[d] é maior que o pivô → verifica v[d-1]

Procurando o primeiro número à direita do pivô que seja menor ou igual a ele

$e = 1$ $v[e] = 7$ pivô = 5
$d = 6$ $v[d] = 0$

esq				p				dir
	e					d		
2	7	4	1	5	3	0	8	6

v[d] é menor que o pivô → troca v[e] com v[d] e recomeça o ciclo

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

$e = 1$ $v[e] = 0$ pivô = 5
$d = 6$ $v[d] = 7$

esq				**p**				**dir**
	e					d		
2	0	4	1	5	3	7	8	6

v[e] é menor que o pivô → verifica v[e+1]

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

e = 2 v[e] = 4 pivô = 5
d = 6 v[d] = 7

esq				p				dir
		e				d		
2	0	4	1	5	3	7	8	6

v[e] é menor que o pivô → verifica v[e+1]

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

e = 3 v[e] = 1 pivô = 5
d = 6 v[d] = 7

esq				p				dir
			e			d		
2	0	4	1	5	3	7	8	6

v[e] é menor que o pivô → verifica v[e+1]

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

e = 4 v[e] = 5 pivô = 5
d = 6 v[d] = 7

esq				p				dir
				e		d		
2	0	4	1	5	3	7	8	6

v[e] igual ao pivô → procura elemento menor que o pivô à direita

Procurando o primeiro número à direita do pivô que seja menor ou igual a ele

e = 4 v[e] = 5 pivô = 5

d = 6 v[d] = 7

esq				p				dir
				e		d		
2	0	4	1	5	3	7	8	6

v[d] é maior que o pivô → verifica v[d-1]

Procurando o primeiro número à direita do pivô que seja menor ou igual a ele

e = 4 v[e] = 5 pivô = 5
d = 5 v[d] = 3

esq				p				dir
				e	d			
2	0	4	1	5	3	7	8	6

v[d] é menor que o pivô → troca v[e] com v[d] e recomeça o ciclo

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

$e = 4$ $v[e] = 3$ $pivô = 5$

$d = 5$ $v[d] = 5$

esq				**p**				**dir**
				e	d			
2	0	4	1	3	5	7	8	6

v[e] é menor que o pivô → verifica v[e+1]

Procurando o primeiro número à esquerda do pivô que seja maior ou igual a ele

$e = 5$ $v[e] = 5$ pivô = 5
$d = 5$ $v[d] = 5$

esq				p				dir
					e d			
2	0	4	1	3	5	7	8	6

Esquerda e direita se encontraram (i.e. e==d):

- O pivô está em seu local definitivo;
- Todos os elementos anteriores ao pivô são menores que ele;
- Todos os elementos posteriores ao pivô são maiores que ele.

- Selecionando o pivô como v[(esq+dir)/2] = 4 (esq = 0, dir = 4)

	esq		p		dir				
			e		d				
Passo 4	2	0	4	1	3	5	7	8	6
					e d				
Passo 5	2	0	3	1	4	5	7	8	6

- Selecionando o pivô como v[(esq+dir)/2] = 0 (esq = 0, dir = 3)

	esq	p		dir					
	e	d							
Passo 6	2	0	3	1	4	5	7	8	6
	e d								
Passo 7	0	2	3	1	4	5	7	8	6

- Selecionando o pivô como v[(esq+dir)/2] = 3 (esq = 1, dir = 3)

		esq	p	dir					
			e	d					
Passo 8	0	2	3	1	4	5	7	8	6
				e d					
Passo 9	0	2	1	3	4	5	7	8	6

- Selecionando o pivô como v[(esq+dir)/2] = 2 (esq = 1, dir = 2)

		esq	dir						
		e	d						
Passo 10	0	2	1	3	4	5	7	8	6
			e d						
Passo 11	0	1	2	3	4	5	7	8	6

- Selecionando o pivô como v[(esq+dir)/2] = 7 (esq = 5, dir = 8)

					esq	**p**		**dir**	
						e		d	
0	1	2	3	4	5	7	8	6	Passo 12
							e	d	
0	1	2	3	4	5	6	8	7	Passo 13
							d	e	
0	1	2	3	4	5	6	7	8	

- Selecionando o pivô como v[(esq+dir)/2] = 5 (esq = 5, dir = 6)

					esq	dir			
					e	d			
0	1	2	3	4	5	6	7	8	Passo 14
					e	d			
0	1	2	3	4	5	6	7	8	Passo 15

Implementação

```
void quicksort(int *v, int left, int right) {
	int i = left, j = right;
	int tmp;
	int pivot = v[(left + right) / 2];

	while (i <= j) {
		while (v[i] < pivot)
			i++;
		while (v[j] > pivot)
			j--;
		if (i <= j) {
			tmp = v[i];
			v[i] = v[j];
			v[j] = tmp;
			i++;    j--;
		}
	}
	if (left < j)
		quicksort(v, left, j);
	if (i < right)
		quicksort(v, i, right);
}
```

Eficiência do QuickSort

A eficiência do processo de ordenação depende de como a partição divide os dados.

Escolha pivô determina eficiência

- *pior caso*: pivô é o maior ou menor elemento $O(N^2)$
- *melhor caso*: pivô é o elemento médio O(N logN)
- *caso médio*: pivô corta vetor arbitrariamente O(N logN)

Escolha do pivô

- um dos elementos extremos do vetor:
  má escolha: $O(N^2)$ se vetor ordenado
- elemento aleatório: boa escolha
- mediana de três elementos (extremos do vetor e ponto médio): boa escolha

Vantagens e desvantagens

Vantagens

- Melhor caso é $O(n.log^n)$ e o caso médio tende a isso
- Boa probabilidade de evitar o pior caso, dependendo da escolha do pivô
  (escolha aleatória tende a ser uma boa opção)
- Não requer memória extra porque não faz cópias do vetor
  (isso não é capturado pela análise assintótica)

Desvantagens

- Pior caso é $O(n^2)$;
- Desempenho pode ser ruim em grandes volumes de dados: o quicksort faz muito acesso à memória, que pode ser lento no caso de memória virtual;
  (isso não é capturado pela análise assintótica)