Estratégia de Divisão e Conquista

Algoritmo de Ordenação: Quick-Sort

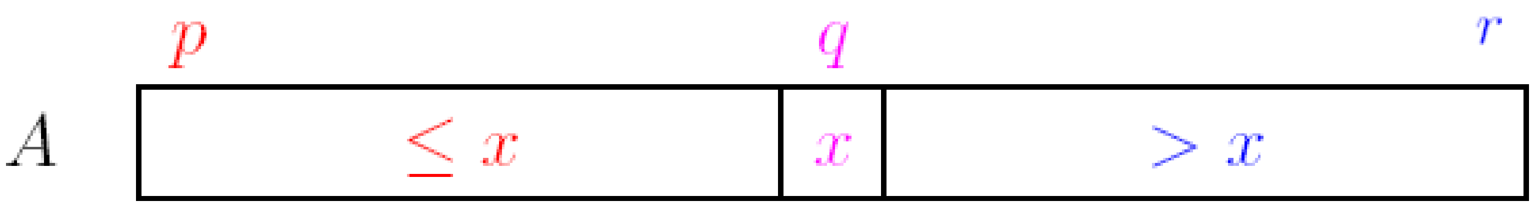
Divisão e Conquista

- Motivação
  - Pegar um problema de entrada grande.
  - Quebrar a entrada em pedaços menores (DIVISÃO).
  - Resolver cada pedaço separadamente. (CONQUISTA)
  - Como resolver os pedaços?
  - Combinar os resultados

- Estratégia
  - Divisão
    - Divida o problema em duas ou mais partes, criando subproblemas menores.
  - Conquista
    - Os subproblemas são resolvidos recursivamente usando divisão e conquista.
    - Caso os subproblemas sejam suficientemente pequenos resolva-os de forma direta.
  - Combina
    - Tome cada uma das partes e junte-as todas de forma a resolver o problema original.

Dividir e Conquistar

- O algoritmo QUICKSORT segue o paradigma de divisão-e-conquista:
  - **Dividir:** divida o vetor em dois subvetores A[p ... q-1] e A[q+1 … r] tais que:



- **Conquistar:** ordene os dois subvetores recursivamente usando o QUICKSORT;
- **Combinar:** nada a fazer, o vetor está ordenado.

Partition

- **Partition**: “dado um vetor A[p..r], rearranjar A[p..r] de modo que todos os elementos pequenos fiquem na parte esquerda do vetor e todos os elementos grandes fiquem na parte direita.”
- Mas o que é ser pequeno? O que é ser grande?
- O ponto de partida, então, é a escolha de um "pivô", digamos x: os elementos do vetor que forem maiores que x serão considerados grandes e os demais (ou seja, os que forem menores que x ou iguais a x) serão considerados pequenos.

Partition

- **Problema**: Rearranjar um dado vetor A[p … r] e devolver um índice q, p ≤ q ≤ r, tais que:

  $$A[p \ldots q-1] \leq A[q] < A[q+1 \ldots r]$$

Entra:

	p									r
A	99	33	55	77	11	22	88	66	33	44

Sai:

	p				q					r
A	33	11	22	33	44	55	99	66	77	88

Partition

i	j									x
A	99	33	55	77	11	22	88	66	33	44

Partition

i	j									x
A	99	33	55	77	11	22	88	66	33	44

i		j								x
A	99	33	55	77	11	22	88	66	33	44

Partition

i	j									x
A	99	33	55	77	11	22	88	66	33	44

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

Partition

i	j									x
A	99	33	55	77	11	22	88	66	33	44

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i			j						x
A	33	99	55	77	11	22	88	66	33	44

Partition

i	j									x
A	99	33	55	77	11	22	88	66	33	44

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i			j						x
A	33	99	55	77	11	22	88	66	33	44

	i				j					x
A	33	99	55	77	11	22	88	66	33	44

Partition

i	j									x
A	99	33	55	77	11	22	88	66	33	44

i		j								x
A	99	33	55	77	11	22	88	66	33	44

	i		j							x
A	33	99	55	77	11	22	88	66	33	44

	i			j						x
A	33	99	55	77	11	22	88	66	33	44

	i				j					x
A	33	99	55	77	11	22	88	66	33	44

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

Partition

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

Partition

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

Partition

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

Partition

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

			i						j	x
A	33	11	22	77	99	55	88	66	33	44

Partition

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

			i						j	x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

Partition

		i				j				x
A	33	11	55	77	99	22	88	66	33	44

			i				j			x
A	33	11	22	77	99	55	88	66	33	44

			i					j		x
A	33	11	22	77	99	55	88	66	33	44

			i						j	x
A	33	11	22	77	99	55	88	66	33	44

				i						j
A	33	11	22	33	99	55	88	66	77	44

	p				q					r
A	33	11	22	33	44	55	88	66	77	99

Partition: Algoritmo

PARTITION(A, p, r)

```
x = A[r];    /* O pivô é o último elemento do vetor */
i = p-1;
for(j = p; j <= r-1; j++)
    if A[j] <= x
        i = i + 1;
        aux = A[i];
        A[i] = A[j];
        A[j] = aux;
i = i + 1;
aux = A[r];
A[r] = A[i];
A[i] = aux;
return i;
```

Partition: Funcionamento

- Para entender como ele funciona, vamos ilustrar a operação PARTITION(A, 1, 9), sobre o vetor A = [13, 4, 9, 5, 12, 7, 19, 6, 7] (observe que o vetor está numerado a partir de 1).
- O pivô é 7. Vamos marcar de vermelho os elementos do vetor que forem maiores do que 7 e de azul os que forem menores ou iguais a 7.

Partition: Funcionamento

p = 1

r = 9

x = A[r] = 7

i = p-1 = 1 – 1 = 0

for(j = p = 1; j <= r-1 = 8; j++)

j = 1: A[1] = 13 <= 7 (falso) [13, 4, 9, 5, 12, 7, 19, 6, 7]

j = 2: A[2] = 4 <=7 (Verdadeiro)

i = i + 1 = 1

aux = A[1] = 13

A[1] = A[2] = 4

A[2] = aux = 13 [4, 13, 9, 5, 12, 7, 19, 6, 7]

Partition: Funcionamento

j = 3:	A[3] = 9 <= 7 (falso)	[4, 13, 9, 5, 12, 7, 19, 6, 7]
j = 4:	A[4] = 5 <= 7 (Verdadeiro)	
	i = i + 1 = 2	
	aux = A[2] = 13	
	A[2] = A[4] = 5	
	A[4] = aux = 13	[4, 5, 9, 13, 12, 7, 19, 6, 7]
j = 5	A[5] = 12 <= 7 (Falso)	[4, 5, 9, 13, 12, 7, 19, 6, 7]

Partition: Funcionamento

j = 6	A[6] = 7 <= 7 (Verdadeiro)	
	i = i + 1 = 3	
	aux = A[3] = 9	
	A[3] = A[6] = 7	
	A[6] = aux = 9	[4, 5, 7, 13, 12, 9, 19, 6, 7]
j = 7	A[7] = 19 <= 7 (Falso)	[4, 5, 7, 13, 12, 9, 19, 6, 7]
j = 8	A[8] = 6 <= 7 (Verdadeiro)	
	i = i+1 = 4	
	aux = A[4] = 13	
	A[4] = A[8] = 6	
	A[8] = aux = 13	[4, 5, 7, 6, 12, 9, 19, 13, 7]

aux = A[9] = 7
A[9] = A[5] = 12
A[5] = aux = 7 [4, 5, 7, 6, **7** , 9, 19, 13, 12]
return i+1 = 5

Observação sobre o Partition

- O PARTITION não ordena o vetor. O único elemento que, ao final, estará na posição definitiva, é o pivô.

- Quanto tempo o algoritmo PARTITION consome?

QuickSort: Algoritmo

- O algoritmo Quicksort recebe um vetor A[p..r] e rearranja o vetor em ordem crescente.

QUICKSORT(A, p, r)

```
if (p < r )
        { q = PARTITION(A, p, r);
          QUICKSORT(A, p, q-1);
          QUICKSORT(A, q+1, r); }
```

QuickSort: Funcionamento

- **Exemplo: Vamos aplicar ao vetor A = [18, 4, 10, 15, 5, 16, 3, 7].**

- A possui n = 8 elementos. Supondo que estejam indexados de 0 a 7.

- A chamada será **QUICKSORT(A, 0, 7).** Notem que os parâmetros passados são índices, e não valores.

QuickSort: Funcionamento