


Introdução:

O quicksort é um algoritmo de ordenação que possui como base a estratégia de divisão e conquista, dividindo o problema de ordenação em sub-problemas menores e resolvendo-os recursivamente.

Funcionamento teórico:


Para iniciarmos o quicksort, devemos definir um valor dentro de nosso vetor que receberá o papel de pivot, ou seja, será um valor base que utilizaremos para dividir nosso problema de ordenação em um ou dois problemas menores. Uma boa escolha do pivot é essencial, pois pode alterar completamente a velocidade do algoritmo, mas isso será abordado a frente, neste momento, o pivot será sempre o último valor do vetor a ser ordenado.

Vetor						
índice	0	1	2	3	4	5
Valor	4	2	7	8	5	3



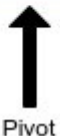
Para o próximo passo, vamos admitir duas variáveis auxiliares, min, que deverá ser posicionado no início do vetor analisado e max, que deverá ser posicionado no final.

Vetor						
índice	0	1	2	3	4	5
Valor	4	2	7	8	5	3



Agora, vamos varrer o vetor e fazer com que todos os valores maiores que o pivot fiquem a direita de max e todos os valores menores ou iguais ao pivot, fiquem a esquerda de min. Quando min e max se cruzarem, trocamos esta posição com o pivot, gerando um novo vetor como o a seguir:

Vetor						
índice	1	5	4	3	2	0
Valor	2	3	5	8	7	4



Por fim, com os dados separados, basta chamarmos o processo do quicksort recursivamente nos vetores gerados a direita e a esquerda do pivot.

<<<< GIF DO QUICKSORT >>>>

	Pseudo-código:	Complexidade:
1	QuickSort(vetor, min, max):	
2	Se min < max então:	O(1)
3	q ← particiona(vetor, min, max)	Θ(n)
4	QuickSort(vetor, min, q-1)	T(k)
5	QuickSort(vetor, q+1, max)	T(n-k)
6		
7	Particiona(vetor, min, max):	
8	pivot ← vetor[max]	
9	i ← min - 1	
10	para j de min até max então:	
11	se vetor[j] < pivot então:	
12	i ← i + 1	
13	vetor[i] ↔ vetor[j]	
14	se pivot < vetor[i+1] então:	
15	vetor[i+1] ↔ vetor[max]	
16	retorna i+1	

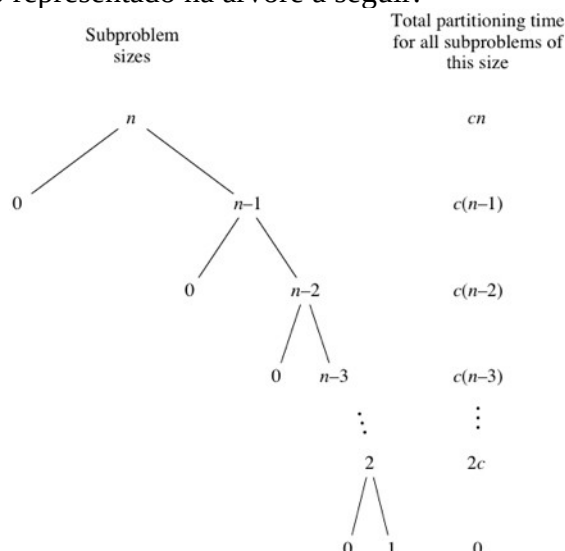
Análise em árvore:

Analisando o algoritmo temos que $T(n)$ é a soma das linhas 3, 4 e 5, ou seja:

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

1. Pior caso:

Quando o pivot não divide o vetor, retirando apenas 1 elemento (o pivot) → $k=1$, deixando todo o peso da árvore para um dos lados. Ou seja, $T(n) = \Theta(n) + T(1) + T(n-1) \rightarrow T(n) = \Theta(n) + T(n-1)$, como representado na árvore a seguir:



Com isso: $T(n) = n + n-1 + n-2 + \dots + 1$
Podemos deduzir que:

$$T(n) = \sum_{i=0}^{n-1} n - i$$

$$T(n) = \sum_{i=0}^{n-1} n - \sum_{i=0}^{n-1} i \rightarrow T(n) = n^2 - \frac{(n-1).n}{2}$$

$$T(n) = \frac{n^2 - n}{2}$$

$$T(n) = T(n-1) + n \quad e \quad T'(n) = \frac{n^2 - n}{2}$$

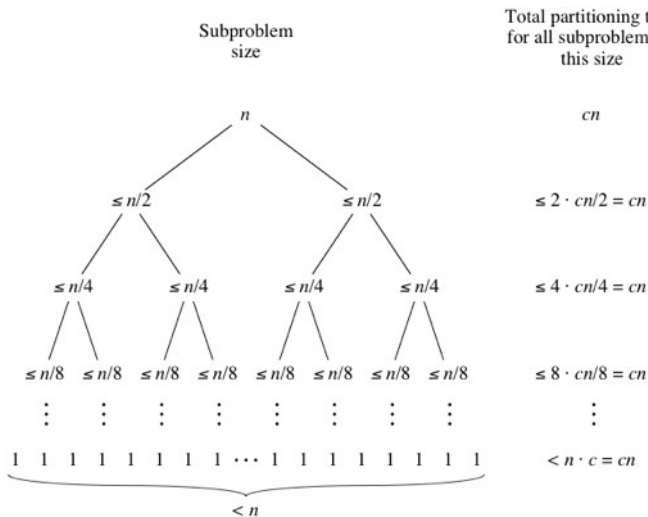
$$\text{Logo: } T(n) = \frac{(n-1)^2 + n - 1 + n}{2} = \frac{n^2 + n}{2} = T'(n)$$

Portanto concluímos que para o pior caso a ordem do quick sort é $(n^2+n)/2$, ou seja, $O(n^2)$

2. Melhor caso:

Quando o pivot divide igualmente os valores, tendo x valores a sua esquerda e no máximo $x+1$ valores a sua direita (ou o contrário), por tanto $k = n/2$.

Ou seja, $T(n) = \Theta(n) + T(n/2) + T(n/2) \rightarrow T(n) = \Theta(n) + 2.T(n/2)$, como representado na árvore a seguir:



Podemos perceber que este problema se encaixa no segundo caso do teorema mestre, logo a complexidade é: $\Theta(n^{\log_2(2)} \cdot \log(n))$, ou seja, a complexidade é: $O(n \cdot \log(n))$

Observações:

Podemos perceber o como a escolha de um bom pivot impacta muito este algoritmo, pois decide se cairemos no melhor ou no pior caso, e portanto se a complexidade será $O(n^2)$ ou $O(n \cdot \log(n))$.

Autor: Pedro Domingues