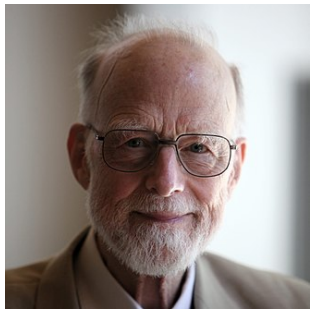
Sir Charles Antony Richard Hoare

- Cientista da Computação Britânico nascido em 1934
- Ganhador de diversos prêmios na Área da Computação
- I call it my billion-dollar mistake. It was the invention of the null reference in 1965. At that time, I was designing the first comprehensive type system for references in an object oriented language (ALGOL W). My goal was to ensure that all use of references should be absolutely safe, with checking performed automatically by the compiler. But I couldn't resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.
- Inventou em 1959 (e publicou em 1961) o QuickSort.



QuickSort

Porque ver e rever o QuickSort?

- Um dos Greatest Hits da ciência da computação.
- Muito usado na prática.
- Memória extra constante.
- Análise muito interessante.

Problema da Ordenação

Dado um arranjo de n inteiros distintos, encontrar o arranjo $\pi = (\pi_1, \pi_2, \ldots, \pi_n)$ que contenha os mesmos elementos mas ordenados de maneira não decrescente, ou seja, $\pi_i \leq \pi_j$ para qualquer $i < j$ e $i, j \in \{1, \ldots, n\}$.

Partição

O QuickSort depende fortemente da operação de Partição cuja a ideia é particionar o arranjo em torno de um pivô:

- Escolha um pivô.
- Reorganize os elementos do arranjo de forma que:
  - A esquerda do pivô tenha os elementos menores que o pivô.
  - A direita do pivô tenha os elementos maiores que o pivô.

Exemplo

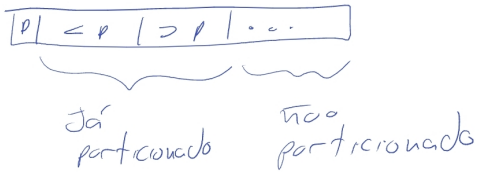
No vetor (3, 8, 2, 5, 1, 4, 7, 6), se escolhemos 3 como pivô.
Uma partição seria: (2, 1, 3, 6, 7, 4, 5, 8)

Observe que depois de uma partição o pivô estará na sua posição correta. Note também que não nos importamos com a posição relativa de cada uma das partes esquerda e direita.

- Como particionar em $O(n)$ usando memória extra?
- Podemos fazer a Partição em tempo $O(n)$ sem usar memória extra.

Ideia:

- Manter uma porção do arranjo já particionado, e o restante ainda não particionado



- Percorrer uma única vez o arranjo

Algoritmo 1: Partição

Entrada: Um arranjo A, índices l e r
Saída: O mesmo arranjo mas particionado

```
1 p = A[l];
2 i = l + 1;
3 para j = l + 1 até r faça
4     se A[j] < p então
5         Troca A[j] e A[i];
6         i = i + 1;
7 Troca A[l] e A[i − 1];
8 devolva A;
```


Provas por Indução - revisão

Vamos revisar o que é uma prova por indução, que utiliza o seguinte axioma:

Principio da Indução Completa

Seja $P(n)$ uma sentença aberta sobre $\mathbb{N}$. Suponha que:

1. $P(0)$ é verdade, e
2. para todo k em $\mathbb{N}$, $((\forall i \in \mathbb{N}) i \leq k \rightarrow P(i)) \rightarrow P(k+1)$

Então $P(n)$ é verdade para todo $n \in \mathbb{N}$.

Ou seja se $P(0), P(1), \ldots, P(k)$ é verdade então $P(k+1)$ também é verdade.

Para usar esse método podemos utilizar o seguinte roteiro:

- *Base da Indução*: Provar que $P(0)$ é verdade.
- *Hipótese de Indução*: Supor que $P(i)$ é verdade para todo $i \leq k \in \mathbb{N}$.
- *Passo da Indução*: Provar que $P(k+1)$ é verdade.

Exemplo: Provar que, para todo $n \geq 0$:

$$1 + 3 + 5 + \ldots + (2n + 1) = (n + 1)^2$$

- *Base da Indução*: $P(0)$ é verdade pois

$$1 = (0 + 1)^2 = 1$$

- *Hipótese de Indução*: Suponha que $P(i)$ é verdade para todo $i \leq k \in \mathbb{N}$.
- *Passo da Indução*: Provar que $P(k + 1)$ é verdade, ou seja que:

$$1 + 3 + 5 + \ldots + (2k + 1) + (2(2k + 1) + 1) = ((k + 1) + 1)^2$$

$$[1 + 3 + 5 + \ldots + (2k + 1)] + (2(k + 1) + 1)) = [(k + 1)^2 + (2(k + 1) + 1))$$
$$= k^2 + 2k + 1 + 2k + 3 = k^2 + 4k + 4 = (k + 2)^2 = ((k + 1) + 1)^2 \quad \square$$

Corretude do algoritmo partição

Invariante de laço: Os elementos $A[l + 1], \ldots, A[i - 1]$ são todos menores que o pivô, e os elementos $A[i], \ldots, A[j - 1]$ são todos maiores que o pivô.

- **Inicialização:** Por vacuidade, antes da primeira iteração a invariante vale.
- **Manutenção:** Em cada iteração verificamos se $A[j]$ é menor que p e nesse caso passamos ele para a primeira porção, incrementando o i. E dessa forma a invariante de mantêm.
- **Termino:** Ao final do laço trocamos o pivô pelo ultimo elemento da primeira porção (ou ele mesmo caso a primeira porção seja vazia) e o arranjo estará particionado.

QuickSort

Algoritmo 2: QuickSort

Entrada: Um arranjo A, o comprimento do arranjo n
Saída: Um arranjo com os mesmos elementos de A porém ordenados

1 **se** $n \leq 1$ **então** devolva A;
2 p = EscolhePivo(A, n);
3 Particionar A em torno de p;
4 Recursivamente ordenar a parte esquerda;
5 Recursivamente ordenar a parte direita;
6 devolva A;

QuickSort

Algoritmo 3: QuickSort

Entrada: Um arranjo A, o comprimento do arranjo n
Saída: Um arranjo com os mesmos elementos de A porém ordenados

1 **se** $n \leq 1$ **então** devolva A;
2 p = EscolhePivo(A, n);
3 Particionar A em torno de p;
4 Recursivamente ordenar a parte esquerda;
5 Recursivamente ordenar a parte direita;
6 devolva A;

Qual o tempo de execução da fase de combinação?

Teorema

O algoritmo QuickSort ordena corretamente um vetor.

Iremos provar a corretude do QuickSort por indução no comprimento n do arranjo. Considere a seguinte sentença aberta $P(n)$: "O QuickSort corretamente ordena arranjos de comprimento n.

- *Base da Indução*: Quando $n = 1$ o algoritmo não faz nada e o vetor está trivialmente ordenado. Portando $P(1)$ é verdade.
- *Hipótese de Indução*: $P(k)$ é verdade para $k < n$
- *Passo da Indução*: Agora queremos provar $P(k + 1)$ é verdade.
  - O algoritmo escolhe algum pivô p e particiona o arranjo
  - O pivô termina já em seu lugar correto
  - O algoritmo recursivamente ordena a primeira e a segunda porção que necessariamente são menores que k, logo pela hipótese de indução, são ordenadas corretamente. □