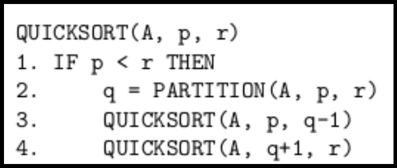
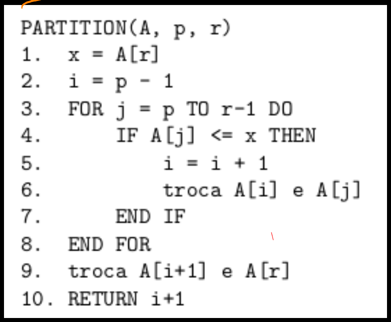
1959 (C.A.R. HOARE)

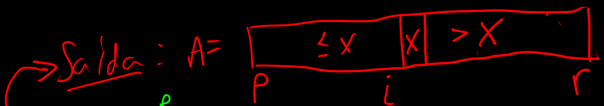


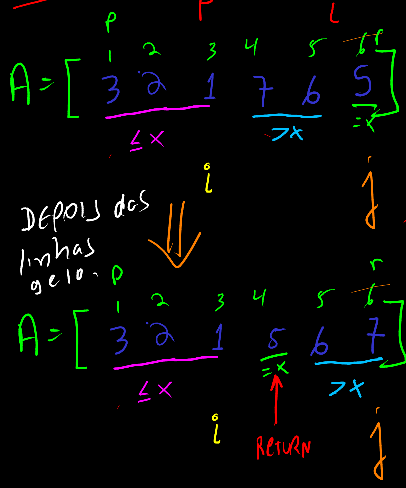
```
QUICKSORT(A, p, r)
1. IF p < r THEN
2.      q = PARTITION(A, p, r)
3.      QUICKSORT(A, p, q-1)
4.      QUICKSORT(A, q+1, r)
```

Partition (Nico Lomuto)

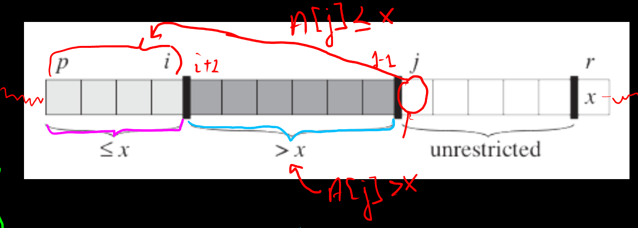


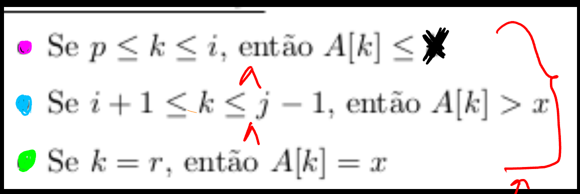
```
PARTITION(A, p, r)
1.  x = A[r]
2.  i = p - 1
3.  FOR j = p TO r-1 DO
4.      IF A[j] <= x THEN
5.          i = i + 1
6.          troca A[i] e A[j]
7.      END IF
8.  END FOR
9.  troca A[i+1] e A[r]
10. RETURN i+1
```





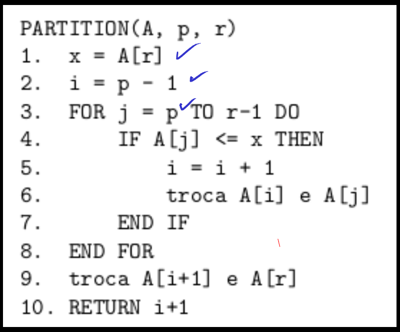
$X = 5$





- Se $p \le k \le i$, então $A[k] \le \cancel{x}$
- Se $i + 1 \le k \le j - 1$, então $A[k] > x$
- Se $k = r$, então $A[k] = x$

Invariante de laço.



```
PARTITION(A, p, r)
1.  x = A[r] ✓
2.  i = p - 1 ✓
3.  FOR j = p✓TO r-1 DO
4.      IF A[j] <= x THEN
5.          i = i + 1
6.          troca A[i] e A[j]
7.      END IF
8.  END FOR
9.  troca A[i+1] e A[r]
10. RETURN i+1
```

Inicialização Antes da primeira iteração $i = p-1$ e $j = p$. Como não há elementos no subvetor $A[p..i] = A[p..p-1]$, então a primeira proposição da invariante é verdadeira. Como não há elementos no subvetor $A[i+1..j-1] = A[(p-1)+1..p-1] = A[p..p-1]$, então a segunda proposição da invariante é verdadeira. A atribuição da linha 1 $(x = A[r])$ satisfez a terceira proposição trivialmente. Como todas as proposições são verdadeiras a invariante de laço é verdadeira antes da primeira iteração.

$X = A[r]$

$i = p - 1$

$j = p$,

$p \le k \le i \rightarrow A[k] \le x$

$\underbrace{p \le k \le p-1}_{\text{vazio!}} \rightarrow A[k] \le x$ ✓

$i+1 \le k \le j-1 \rightarrow A[k] > x$

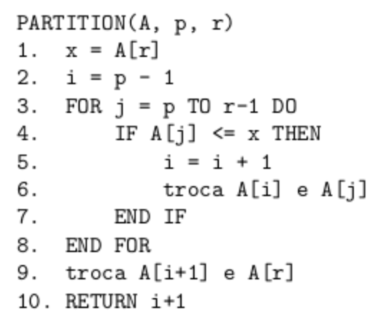
$(p-1)+1 \le k \le p-1 \rightarrow A[k] > x$ ✓

$\underbrace{p \le k \le p-1}_{\text{subvetor vazio}} \rightarrow A[k] > x$

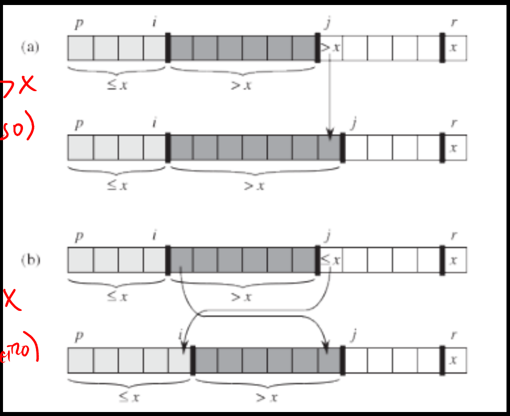
$k = r \rightarrow A[k] = x$ ✓

$k = r \rightarrow A[r] = x$

$k = r \rightarrow x = x$

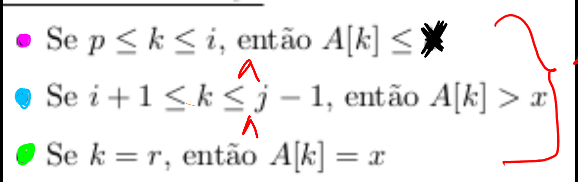


```
PARTITION(A, p, r)
1.  x = A[r]
2.  i = p - 1
3.  FOR j = p TO r-1 DO
4.      IF A[j] <= x THEN
5.          i = i + 1
6.          troca A[i] e A[j]
7.      END IF
8.  END FOR
9.  troca A[i+1] e A[r]
10. RETURN i+1
```



$A[j] > x$ (IF FALSO)

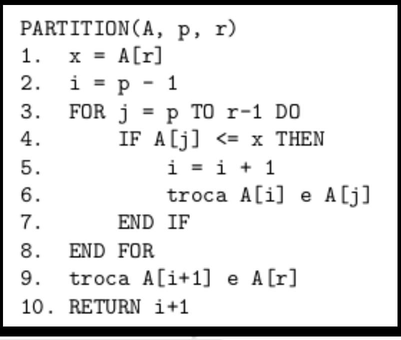
$A[j] \le x$ (IF verdadeiro)



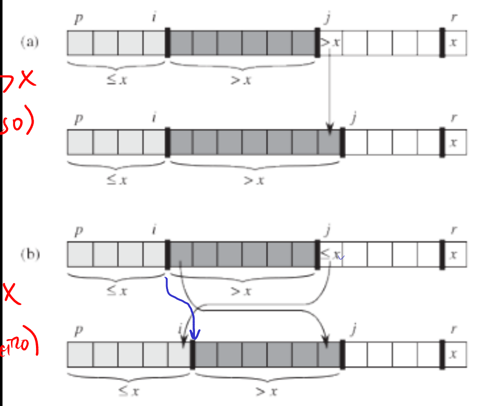
- Se $p \le k \le i$, então $A[k] \le x$
- Se $i + 1 \le k \le j - 1$, então $A[k] > x$
- Se $k = r$, então $A[k] = x$

INVARIANTE DE LAÇO.

Manutenção Consideramos 2 casos: o caso que a condicional da linha 4 é verdadeira ($A[j] \le x$) e o caso que é falsa ($A[j] > x$). Quando $A[j] > x$, o elemento na posição j já está na posição certa uma vez que os elementos maiores que x estão, pela invariante de laço, entre i+1 e j-1. Quando j é incrementado a segunda proposição se mantém verdadeira p/ o início da próxima iteração. A primeira proposição é mantida verdadeira pois, pela invariante de laço ela é verdadeira e nenhum elemento A[p..i] foram alterados, nem p e i foram alterados. A terceira proposição também se mantém verdadeira pois nem x nem A[r] mudaram de valor nesta iteração.

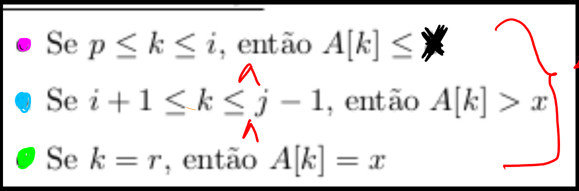


```
PARTITION(A, p, r)
1.   x = A[r]
2.   i = p - 1
3.   FOR j = p TO r-1 DO
4.       IF A[j] <= x THEN
5.           i = i + 1
6.           troca A[i] e A[j]
7.       END IF
8.   END FOR
9.   troca A[i+1] e A[r]
10.  RETURN i+1
```



$A[j] > x$ (IF FALSO)

$A[j] \leq x$ (IF verdadeiro)

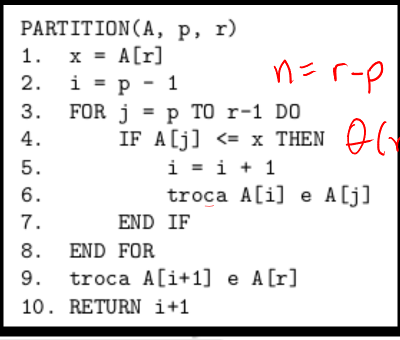


- ● Se $p \leq k \leq i$, então $A[k] \leq x$
- ● Se $i + 1 \leq k \leq j - 1$, então $A[k] > x$
- ● Se $k = r$, então $A[k] = x$

INVARIANTE DE LAÇO.

Manutenção (cont.) No caso $A[j] \leq x$ (if verdadeiro), i é incrementado, $A[i]$ e $A[j]$ são trocados e j é incrementado. Após a troca, temos que $A[i] \leq x$, portanto temos um elemento a mais em $A[p..i]$. Portanto a primeira proposição é mantida verdadeira. Da mesma forma, $A[j-1] > x$, uma vez que o item que foi trocado p/ a posição $j-1$ (que estava em $A[i]$ antes do incremento de j) é, por invariante de laço, maior que x, satisfazendo a segunda proposição. A terceira proposição continua verdadeira uma vez que x nunca recebe valor após a linha 1 e j nunca assume o valor R.

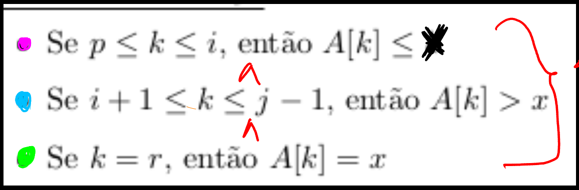
Portanto a invariante se mantém verdadeira em todos os casos.



```
PARTITION(A, p, r)
1.  x = A[r]
2.  i = p - 1
3.  FOR j = p TO r-1 DO
4.      IF A[j] <= x THEN
5.          i = i + 1
6.          troca A[i] e A[j]
7.      END IF
8.  END FOR
9.  troca A[i+1] e A[r]
10. RETURN i+1
```

$n = r - p$

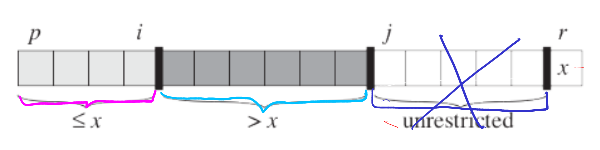
$\Theta(n)$.



- ● Se $p \le k \le i$, então $A[k] \le x$
- ● Se $i + 1 \le k \le j - 1$, então $A[k] > x$
- ● Se $k = r$, então $A[k] = x$

Invariante de laço.

Término No término, $j = r$. Assim, como a partição $A[j..r-1] = A[j..j-1]$ é vazia, todo elemento do vetor $A[p..r-1]$ está em suas respectivas partições conforme a invariante. Portanto, $A[p..i]$ contém os elementos menores ou iguais a X e $A[i+1..j-1] = A[i+1..r-1]$ contém os elementos maiores que X. Para obter o vetor particionado com o pivô em sua posição adequada, a linha 9 troca o pivô ($A[r]$) com o primeiro elemento da partição com o elementos maiores que X (pela invariante de laço, o elemento $A[i+1]$). Portanto, as modificações realizadas pelo algoritmo partition particionam o vetor conforme esperado. Portanto, o algoritmo está correto!

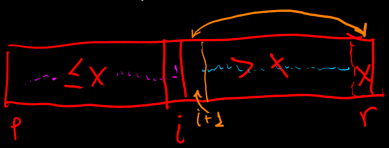


quando o for termina,

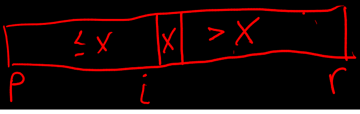
$j = r$.

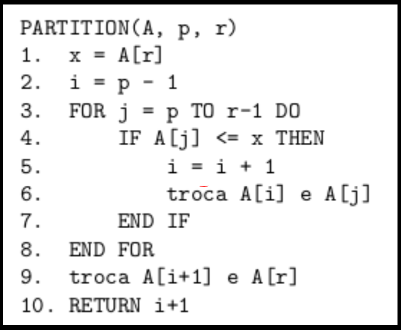
- Se $p \le k \le i$ então $A[k] \le X$
- Se $i+1 \le k \le j-1$ então $A[k] > X$

  Se $i+1 \le k \le r-1$ então $A[k] > X$
- Se $k = r$, então $A[k] = X$

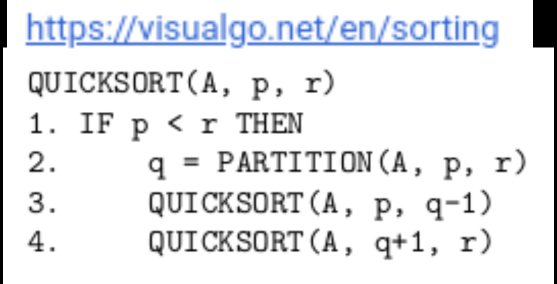


⇓ Após a linha 9





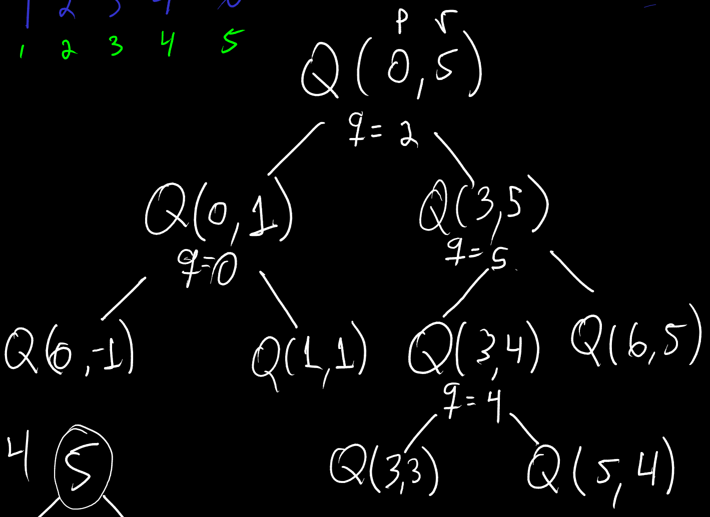
```
PARTITION(A, p, r)
1.  x = A[r]
2.  i = p - 1
3.  FOR j = p TO r-1 DO
4.      IF A[j] <= x THEN
5.          i = i + 1
6.          troca A[i] e A[j]
7.      END IF
8.  END FOR
9.  troca A[i+1] e A[r]
10. RETURN i+1
```



https://visualgo.net/en/sorting

```
QUICKSORT(A, p, r)
IF p < r THEN
      q = PARTITION(A, p, r)
      QUICKSORT(A, p, q-1)
      QUICKSORT(A, q+1, r)
```

V =	0	1	2	3	4	6
	0	1	2	3	4	5



pior caso

Custo

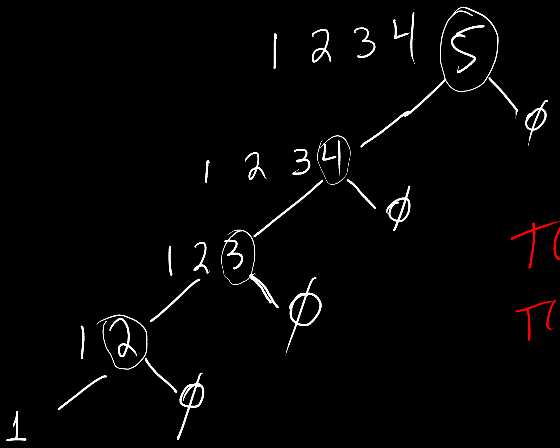
5

4

3

2

1



$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2} = \Theta(n^2).$$

$$T(n) = T(n-1) + T(0) + \Theta(n)$$

custo do partition

$$T(n) = T(n-1) + \Theta(n)$$
$$= \Theta(n^2).$$

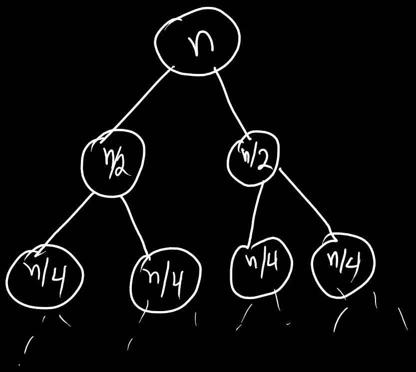
Análise

O pior caso acontece quando Partition produz dois subproblemas, um com 0 elementos e outro com n - 1 em todas as chamadas recursivas ao quicksort. Como o particionamento tem custo $\Theta(n)$ e $T(0)$ tem custo constante, a recorrência fica:

$$T(n) = T(n-1) + T(0) + \Theta(n)$$
$$= T(n-1) + \Theta(n)$$

Pelo método a árvore de recursão, teremos uma árvore com n níveis, cada uma com custo $\Theta(n)$. Portanto, o custo é $\Theta(n^2)$, que pode ser verificado c/ o método da substituição.

Melhor Caso



$T(n) = 2T(n/2) + \Theta(n).$

$= \Theta(n \lg n)$

Análise O melhor caso acontece quando há o particionamento de forma que exatamente metade dos elementos fiquem a esquerda do pivô e a outra metade fiquem a direita do pivô. Isto produz a árvore de recursão binária com a menor altura possível. Isto implica no menor custo total de particionamento possível. Neste caso a recorrência é $T(n) = 2T(n/2) + \Theta(n)$. Pelo método mestre, $T(n) = \Theta(n \lg n)$.