QuickSort

Prof. Juliano Foleis

Partition

O algoritmo `PARTITION` usado no `QuickSort` rearranja os elementos de um vetor V[p,...,r], tal que $p \leq r$ de forma que

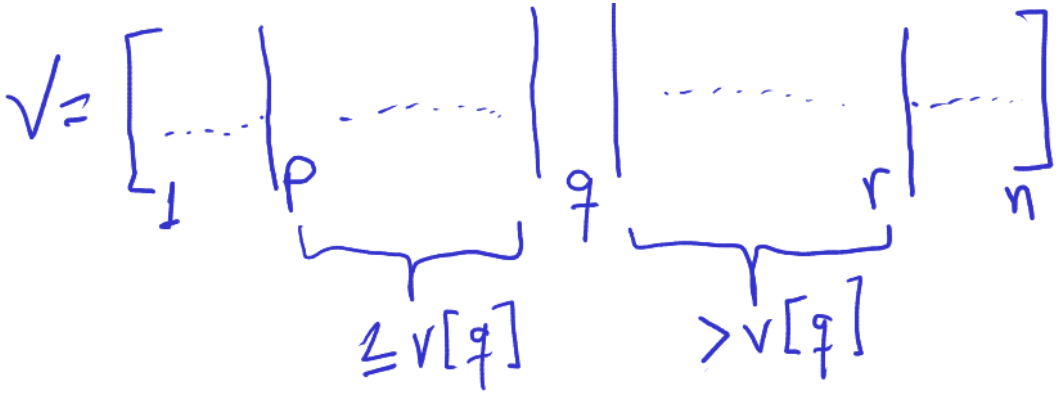
$$\forall k(p \leq k < q \rightarrow v[k] \leq v[q])$$
$$\wedge$$
$$\forall k(q+1 \leq k \leq r \rightarrow v[k] > v[q])$$

Partition

O algoritmo `PARTITION` usado no `QuickSort` rearranja os elementos de um vetor V[p,…,r], tal que $p \leq r$ de forma que

$$\forall k(p \leq k < q \rightarrow v[k] \leq v[q]) \wedge \forall k(q + 1 \leq k \leq r \rightarrow v[k] > v[q])$$

Graficamente,



PARITITON (Lomuto)

Existem vários algoritmos possíveis que estabelecem as propriedades de `PARTITION`.

PARITITON (Lomuto)

Existem vários algoritmos possíveis que estabelecem as propriedades de `PARTITION`.

Vamos estudar a implementação Lomuto:

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

PARITITON (Lomuto)

Existem vários algoritmos possíveis que estabelecem as propriedades de `PARTITION`.

Vamos estudar a implementação Lomuto:

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

Exercício: Prove que este algoritmo tem custo $\Theta(n)$ tal que $n = r - p + 1$.

Corretude do PARTITION

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

Este algoritmo está correto somente se ao final da execução os elementos de um vetor V[p,...,r], tal que $p \leq r$, respeitarem as seguintes propriedades:

$$\forall k(p \leq k \leq i \rightarrow v[k] \leq v[i+1]) \wedge \forall k(i+1 < k \leq r \rightarrow v[k] > v[i+1])$$

Corretude do PARTITION

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

Este algoritmo está correto somente se ao final da execução os elementos de um vetor V[p,...,r], tal que $p \leq r$, respeitarem as seguintes propriedades:

$$\forall k(p \leq k \leq i \rightarrow v[k] \leq v[i+1]) \wedge \forall k(i+1 < k \leq r \rightarrow v[k] > v[i+1])$$

Invariante:

No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

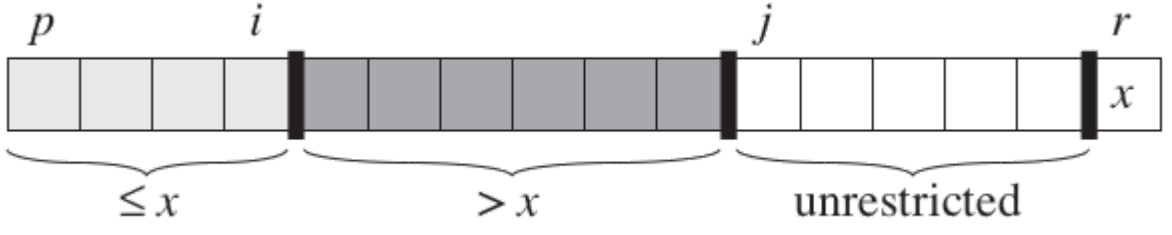
Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i + 1 \leq k \leq j - 1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Graficamente:



Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i + 1 \leq k \leq j - 1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Inicialização: Antes da primeira iteração, i=p-1, j=p e x=V[r]. Vamos verificar as proposições da invariante:

Proposição 1: Pode ser escrita como $\forall k(p \leq k \leq i \rightarrow V[k] \leq x)$.

$$\forall k(p \leq k \leq i \rightarrow V[k] \leq x)$$
$$\forall k(p \leq k \leq (p-1) \rightarrow V[k] \leq x) \text{ (substituindo i=p-1)}$$

Como o domínio de k é vazio uma vez que $((p-1) - p + 1) = 0$, então, pela definição do quantificador universal, a proposição é verdadeira. Em outras palavras, como não há elementos no vetor v[p..p-1] não há como violar a proposição.

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Inicialização (cont.): Antes da primeira iteração, i=p-1, j=p e x=V[r].

Proposição 2: Pode ser escrita como $\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x)$.

$$\begin{aligned} &\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x) \\ &\forall k((p-1)+1 \leq k \leq p-1 \rightarrow V[k] > x) \text{ (substituindo } i = p-1 \text{ e } j = p) \\ &\forall k(p \leq k \leq p-1 \rightarrow V[k] > x) \end{aligned}$$

Como o domínio de k é vazio uma vez que $((p-1)-p+1) = 0$, então, pela definição do quantificador universal, a proposição é verdadeira. Em outras palavras, como não há elementos no vetor v[p..p-1] não há como violar a proposição.

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Inicialização (cont.): Antes da primeira iteração, i=p-1, j=p e x=V[r].

Proposição 3: A proposição diz que p/ $k = r$, V[k] = x. Substituindo $k = r$, temos que V[r] = x, que é a atribuição realizada na linha 1. Portanto, a proposição 3 é verdadeira antes da primeira iteração.

Como mostramos que as proposições 1, 2 e 3 são verdadeiras antes da primeira iteração do laço, a invariante é verdadeira após a inicialização.

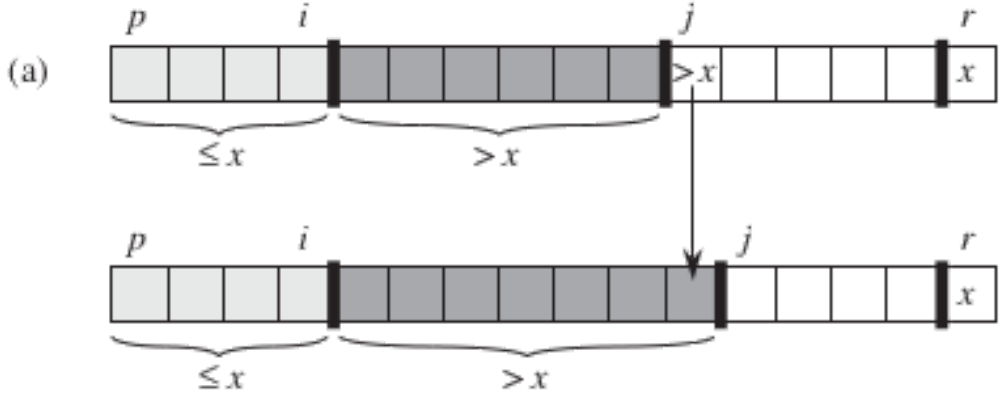
Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i + 1 \leq k \leq j - 1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção: Consideramos 2 casos: o caso que a condicional da linha 5 é verdadeira e o caso que é falsa. Quando a condicional é falsa ($V[j] > x$), temos a seguinte situação:



Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção (cont.): No caso que a condicional da linha 5 é falsa, sabemos que $V[j] > x$ e que não há alteração nas variáveis i, j e em qualquer elemento de V[p..r]. Isto é verdade pois a única instrução que é executada é o incremento de j.

Proposição 1: No início da iteração supomos que a invariante é verdadeira, e portanto esta proposição também é. Como i não é alterado e não há troca dos elementos que estão entre as posições p e i, a proposição 1 é mantida verdadeira.

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção (cont.): No caso que a condicional da linha 5 é falsa, sabemos que $V[j] > x$ e que não há alteração nas variáveis i, j e em qualquer elemento de V[p..r]. Isto é verdade pois a única instrução que é executada é o incremento de j.

Proposição 2:

$$\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x) \text{ (invariante de laço)}$$
$$\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x) \wedge V[j] > x \text{ (como } V[j] > x)$$
$$\forall k(i+1 \leq k \leq \mathbf{j} \rightarrow V[k] > x) \text{ (def. de } \forall)$$
$$\forall k(i+1 \leq k \leq \mathbf{j}-1 \rightarrow V[k] > x) \text{ (após j++ do FOR)}$$

Conforme mostrado acima, a invariante é mantida verdadeira antes da próxima iteração.

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i + 1 \leq k \leq j - 1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção (cont.): No caso que a condicional da linha 5 é falsa, sabemos que $V[j] > x$ e que não há alteração nas variáveis i, j e em qualquer elemento de V[p..r]. Isto é verdade pois a única instrução que é executada é o incremento de j.

Proposição 3: Pela invariante de laço, sabemos que no início da iteração p/ $k = r$, V[k] = x. Como r não é usado para indexar o vetor e nenhum elemento de V[p..r] é alterado neste caso, não é possível que $V[k] \neq x$.

Como mostramos que as proposições 1, 2 e 3 são mantidas verdadeiras após a execução do laço quando a condicional da linha 5 é falsa, a invariante é mantida verdadeira neste caso. Vamos verificar o que acontece quando a condicional da linha 5 é verdadeira.

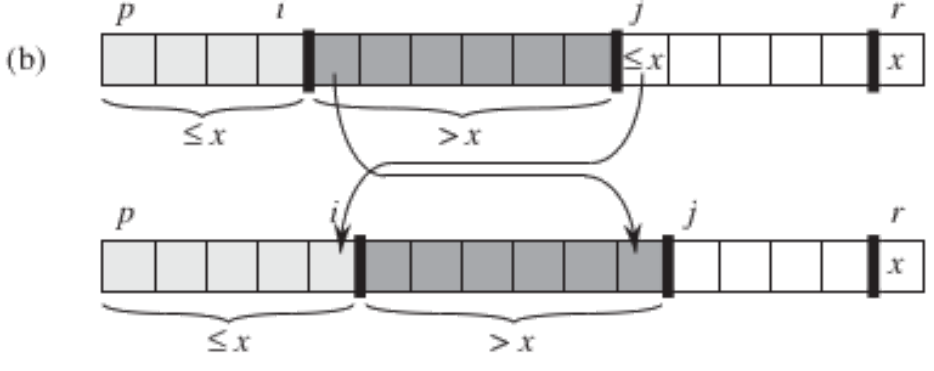
Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i + 1 \leq k \leq j - 1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção (cont.): Quando a condicional da linha 5 é verdadeira ($V[j] \leq x$), temos a seguinte situação:



Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção (cont.): No caso que a condicional da linha 5 é verdadeira, sabemos que $V[j] \leq x$.

Proposição 1:

$$\begin{aligned}
&\forall k(p \leq k \leq i \rightarrow V[k] \leq x)\ (\text{invariante de laço}) \\
\forall k(p \leq k \leq i-1 \rightarrow V[k] \leq x) \wedge (V[i] > x) \wedge (V[j] \leq x)\ &(\text{após a l.6, pelo if e prop. 2}) \\
\forall k(p \leq k \leq i-1 \rightarrow V[k] \leq x) \wedge (V[i] \leq x) \wedge (V[j] > x)\ &(\text{após a l.7}) \\
\forall k(p \leq k \leq i-1 \rightarrow V[k] \leq x) \wedge (V[i] \leq x)\ &(\text{eq. simplificação}) \\
&\forall k(p \leq k \leq i \rightarrow V[k] \leq x)\ (\text{def. } \forall)
\end{aligned}$$

Pelo argumento acima, a proposição é mantida verdadeira.

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção (cont.): No caso que a condicional da linha 5 é verdadeira, sabemos que $V[j] \leq x$.

Proposição 2:

$$\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x) \text{ (invariante de laço)}$$
$$\forall k(i \leq k \leq j-1 \rightarrow V[k] > x) \wedge V[j] \leq x \text{ (após a l.6 e pelo if)}$$
$$\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x) \wedge (V[j] > x) \wedge (V[i] \leq x) \text{ (após a linha 7)}$$
$$\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x) \wedge (V[j] > x) \text{ (eq. simplificação)}$$
$$\forall k(i+1 \leq k \leq j \rightarrow V[k] > x) \text{ (def. de } \forall)$$
$$\forall k(i+1 \leq k \leq j-1 \rightarrow V[k] > x) \text{ (após j++ do FOR)}$$

Conforme mostrado acima, a invariante é mantida verdadeira antes da próxima iteração.

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Manutenção (cont.): No caso que a condicional da linha 5 é verdadeira, sabemos que $V[j] \leq x$.

Proposição 3: Pela invariante de laço, sabemos que no início da iteração p/ $k = r$, V[k] = x. r não é usado para indexar o vetor, e portanto a $V[r]$ não é alterado diretamente. Além disto, o vetor é alterado apenas na linha 7, usando i e j como índices, que nunca recebem o valor r, uma vez que j só varia de p a $r-1$ e i é sempre menor que j. Desta forma, não é possível que $V[k] \neq x$.

Como mostramos que as proposições 1, 2 e 3 são mantidas verdadeiras após a execução do laço quando a condicional da linha 5 é verdadeira, a invariante é mantida verdadeira neste caso. Como mostramos que a invariante é mantida em ambos casos, podemos concluir que a invariante se mantém entre duas iterações quaisquer.

Corretude do PARTITION

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

Invariante: No início de cada iteração do laço FOR das linhas 4–9, para qualquer índice k,

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq j-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Término: No término, temos $j = r$. Substituindo na invariante, temos:

1. Se $p \leq k \leq i$, então $V[k] \leq x$
2. Se $i+1 \leq k \leq r-1$, então $V[k] > x$
3. Se $k = r$, então $V[k] = x$

Após a troca da linha 10 temos:

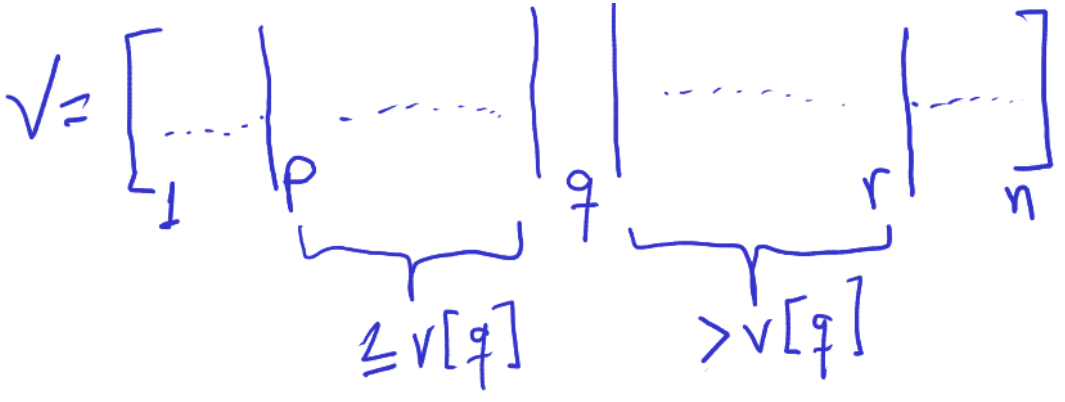
1. Se $p \leq k \leq i$, então $V[k] \leq V[i+1]$ (pois V[r]=x (pela 3ª prop.) foi trocado com V[i+1])
2. Se $i+1 < k \leq r$, então $V[k] > V[i+1]$ (pois V[r]=x (pela 3ª prop.) foi trocado com V[i+1])

Corretude do PARTITION

Término (cont.):

1. Se $p \leq k \leq i$, então $V[k] \leq V[i+1]$
2. Se $i+1 < k \leq r$, então $V[k] > V[i+1]$

Considerando $q = i+1$, estas duas proposições descrevem exatamente as propriedades que queríamos assegurar ao vetor pelo algoritmo PARTITION. Relembrando a figura apresentada no início:



Portanto, o algoritmo está correto.

QuickSort

```
1 QUICKSORT(V, p, r)
2   IF p < r THEN
3     q = PARTITION(V, p, r)
4     QUICKSORT(V, p, q-1)
5     QUICKSORT(V, q+1, r)
6   END IF
```

Baseado em divisão e conquista, o algoritmo para ordenar $V[p, \ldots, r]$ pode ser descrito por:

- ***DIVISÃO*** – Particionar o vetor $V[p, \ldots, r]$ em dois subvetores potencialmente vazios $V[p, \ldots, q-1]$ e $V[q+1, \ldots, r]$ de forma que os elementos $V[p, \ldots, q-1]$ sejam menores ou iguais a $V[q]$, que, por sua vez, é menor ou igual a todos os elementos em $V[q+1, \ldots, r]$.
- ***CONQUISTA*** – Ordenar $A[p, \ldots, q-1]$ e $A[q+1, \ldots, r]$ recursivamente, usando `QUICKSORT`.
- ***COMBINAÇÃO*** – Como os subvetores $A[p, \ldots, q-1]$ e $A[q+1, \ldots, r]$ já estão ordenados, trabalho adicional não é necessário para combiná-los: o subvetor $A[p, \ldots, r]$ está ordenado.

Desempenho do QuickSort

O desempenho do QUICKSORT depende do balanceamento obtido por PARTITION, que depende da escolha do pivô e da relação dos valores dos elementos. Se o particionamento é balanceado, ou seja, o pivô “divide” o subvetor em 2 partições com quantidades praticamente iguais de elementos, o algoritmo possui complexidade assintótica idêntica ao HEAPSORT e MERGESORT.

Caso contrário, a complexidade assintótica é a mesma que do ineficiente BUBBLESORT. Primeiramente, analisamos informalmente o desempenho do QUICKSORT em relação ao balanceamento do particionamento.

Custo do QuickSort no Pior Caso

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

$$\begin{aligned} T(n) &= T(n-1) + T(0) + \Theta(n) \\ &= T(n-1) + \Theta(n) \end{aligned}$$

Intuitivamente, ao somarmos o custo de todos os níveis da recursão teríamos n níveis, cada um com um elemento a menos que o nível anterior de custo $\Theta(n)$ teríamos uma série aritmética ($cn + c(n-1) + c(n-2) + \cdots + c1$), que é $\Theta(n^2)$. Desta forma, se o particionamento for totalmente desbalanceado em todo nível da recursão, o tempo de execução é $\Theta(n^2)$, que não é melhor que o INSERTIONSORT, por exemplo. Além disto, o pior caso ocorre quando o vetor de entrada já está ordenado, um caso que o INSERTIONSORT executa em tempo $\Theta(n)$.

Custo do QuickSort no Melhor Caso

No corte (particionamento) mais bem distribuído possível, dois subproblemas devem ser resolvidos recursivamente, um com $\left\lceil \frac{n}{2} \right\rceil$ elementos e outro com $\left\lceil \frac{n}{2} \right\rceil - 1$ elementos. Neste caso, o tempo necessário para QUICKSORT executar é dado pela recorrência

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n)$$

Ignorando o chão e teto e a subtração de 1, sem perda de generalidade. Pelo caso 2 do teorema mestre, a recorrência tem solução $T(n) = \Theta(n \lg n)$. Portanto, se o particionamento for balanceado em todos os níveis, temos um algoritmo mais eficiente, comparável ao HEAPSORT, com constantes escondidas potencialmente menores.

Particionamento Desbalanceado

Uma das vantagens práticas do QUICKSORT é que a complexidade do caso médio se assemelha mais ao melhor caso do que ao pior caso. Para entender melhor esta relação é necessário entender como o balanço do particionamento reflete na recorrência que descreve o tempo de execução.

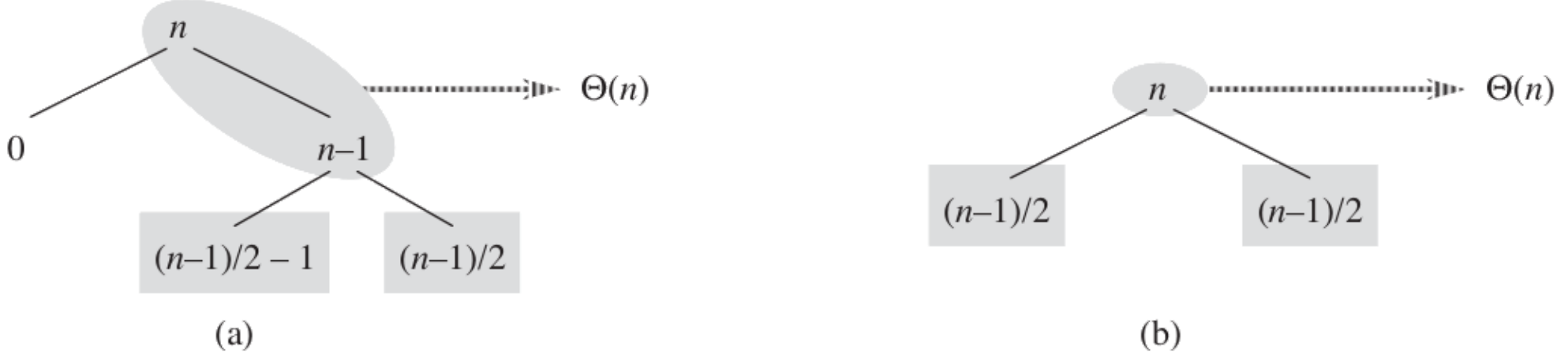
Suponha que PARTITION sempre produza subproblemas em proporção 9 pra 1 em todos os níveis da recursão. Note que esta aproximação é grosseira, uma vez que em uma execução média todo tipo de corte seria realizado por PARTITION, nas mais variadas proporções. No entanto, como veremos adiante, esta aproximação nos permite ter uma boa noção do tempo de execução no caso médio. Portanto, supondo a proporção 9 pra 1, a recorrência de QUICKSORT seria

$$T(n) = T\left(\frac{9n}{10}\right) + T\left(\frac{n}{10}\right) + \Theta(n)$$

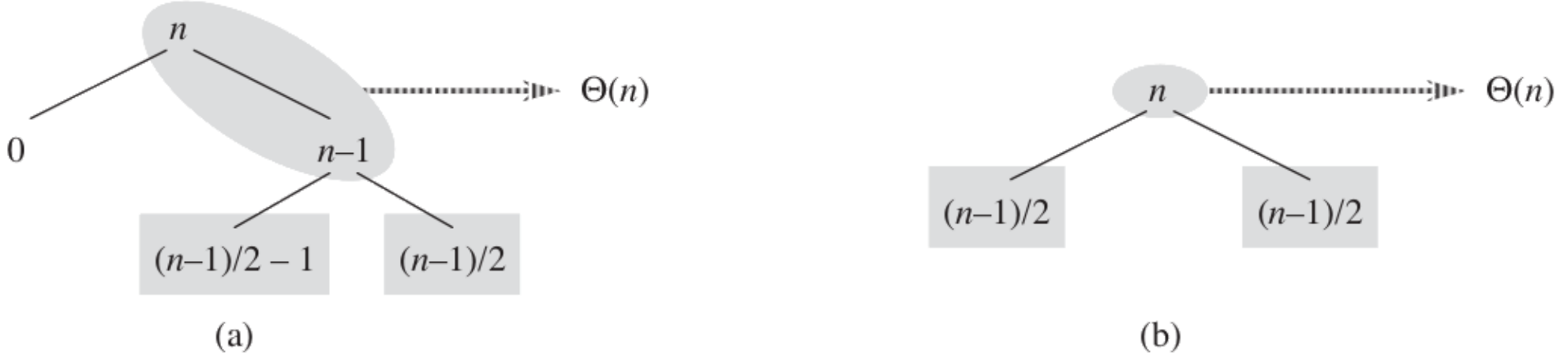
Como exercício, construa a árvore de recursão para esta recorrência. Note que todo nível da árvore tem custo cn até o nível de altura $\log_1 0 = \Theta(\lg n)$. A partir deste nível, todos os demais possuem custo $O(cn)$, ou seja, no máximo cn. A recursão termina na altura $\log_{\frac{10}{9}} n = \Theta(\lg n)$. É possível confirmar, pelo método da substituição que $T(n) = \Theta(n \lg n)$.

Intuição para o Caso Médio

Quando `QUICKSORT` é executado em um vetor qualquer, o particionamento dificilmente ocorre da mesma forma em todos os níveis, como a análise anterior assumiu. Esperamos que alguns cortes sejam bem balanceados e outros não. No caso médio, `PARTITION` produz uma quantidade mista de cortes “bons” e “ruins”. Em uma árvore de recursão para o caso médio de `QUICKSORT`, há cortes bons e ruins distribuídos pela árvore. Suponha, por questão de simplicidade de cálculos, que cortes bons e ruins estão em níveis alternados na árvore, como apresentado na Figura a seguir:



Intuição para o Caso Médio



Intuitivamente, o custo do corte “ruim” é absorvido no custo $\Theta(n)$ do corte “bom”, resultando em um corte bom. Assim, o custo do `QUICKSORT`, quando os níveis alternam entre cortes “bons” e “ruins” é parecido com o custo envolvido quando há apenas cortes “bons”: $O(n \lg n)$, mas com uma constante maior, escondida na notação O.

Bibliografia

[CRLS] CORMEN, T. H. et al. Algoritmos: Teoria e Prática. Elsevier, 2012. 3a Ed. Capítulo 7 (*QuickSort*), Seções 7.1 e 7.2.