Quick Sort

- Frequentemente melhor opção prática!
- Baseado em **D&C:**
  - **Dividir**: Reparticionar (rearranjar) *A[p..r]* em dois subarranjos: *A[p..q-1]* e *A[q+1..r]* tal que:
    - Todos os elementos em $A[p..q-1] \leq A[q]$
    - Todos os elementos em $A[q+1..r] \geq A[q]$
      - O índice ***q*** é calculado no procedimento "**partição**"
  - **Conquistar**: Os subarranjos *A[p..q-1]* e *A[q+1..r]* são ordenados por chamadas recursivas a *QuickSort*
  - **Combinar**: Uma vez que os subarranjos são ordenados localmente, não é necessário trabalho para combina-los. Assim, *A[p..r]* está ordenado

Quick Sort
Algoritmo partição

- 1. Selecionar um **pivô** (elemento de partição) que dividirá o subarranjo
- 2. Rearranjar a lista de modo que todos os elementos nas posições:
  - **a)** antes do pivô sejam menores ou iguais ao pivô
  - **b)** após o pivô sejam maiores ou iguais ao pivô
    - Diversos métodos para rearranjar os elementos
      - Método/algoritmo conhecido como **Partição**

Quick Sort
Algoritmo (II)

```
   QUICKSORT(A, p, r)
1) If (p<r)
2)     q ← PARTITION(A, p, r)
3)     QUICKSORT(A, p,   q-1)
4)     QUICKSORT(A, q+1, r)

   Início: QUICKSORT(A, 1, lenght[A])
```

Quick Sort
Sub-algoritmo: Partição (II)

- Ao final, o vetor *A[p..r]* está particionado de tal forma que:
  - Os itens em *A[p], A[p+1], . . . , A[i]* são menores ou iguais ao pivô *p*
  - Os itens em *A[i+1], A[i+2], . . . , A[j]* são maiores ou iguais ao pivô *p*

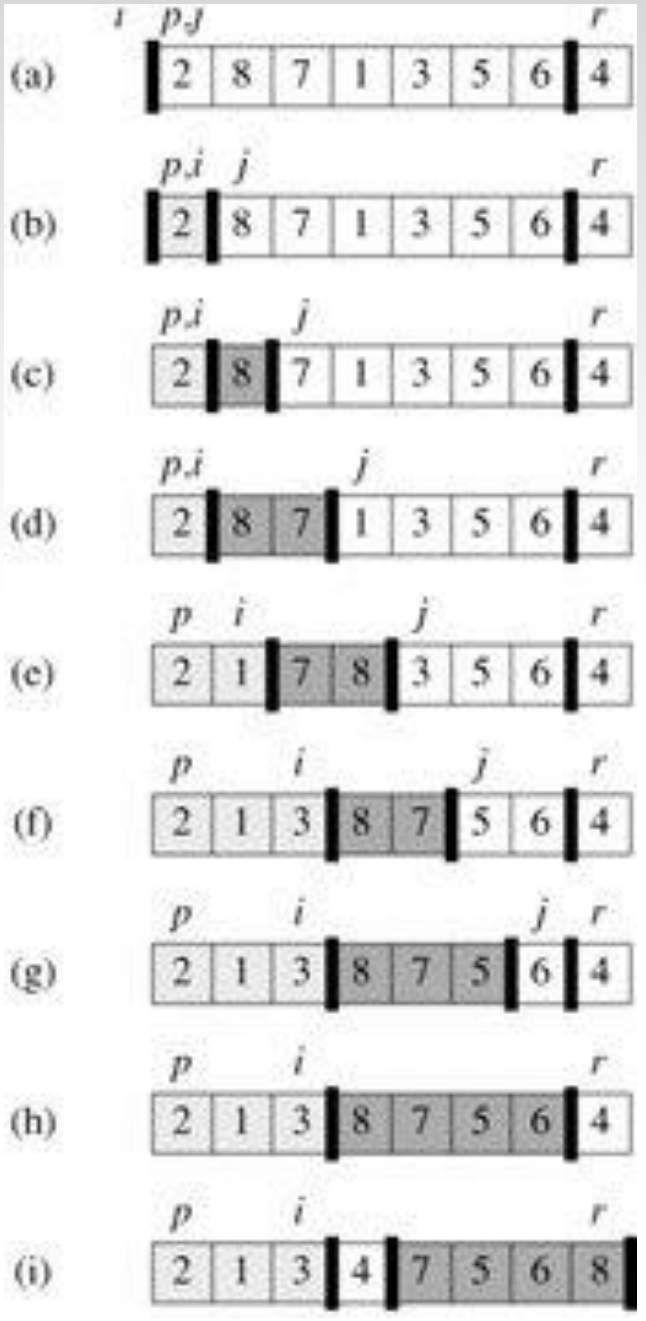
Quick Sort
Sub-algoritmo: *Partition* (III)

```
   PARTITION(A, p, r)
1) x = A[r] //obtem o pivo x
2) i ← p-1
3) for j ← p to r-1
4)     if A[j] ≤ x
5)                 i ← i+1
6)                 swap(A[i], A[j])
7) swap(A[i+1], A[r])
8) return (i+1)
```

Sub-algoritmo: *Partition*

Exemplo

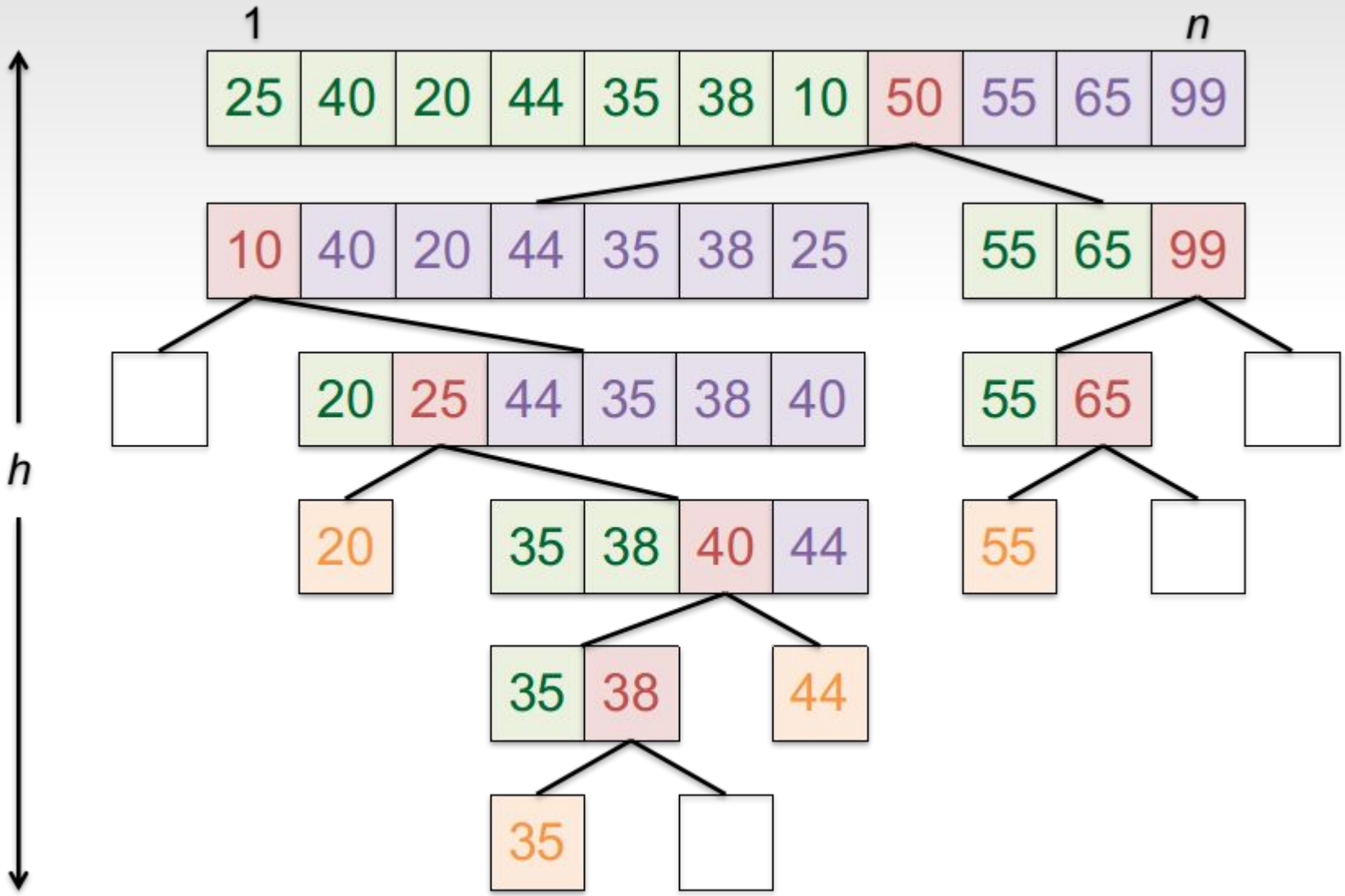
- Sombra leve: menor que pivô
- Sombra forte: maior que pivô
- Sem sombra: a serem analisados
- Final branco: pivô
- (a) arranjo inicial
- (b) elemento A[1] analisado
- (c)-(d) elementos A[2..3] analisados
- (e) elemento A[4] analisado e pemutado com A[2]
- (f) A[5] analisado e permutado com A[3]
- (g)-(h) Partição maior cresce
- (i) Pivô inserido na posição correta



Quick Sort
Análise

- A execução pode ser facilmente descrita por uma estrutura de árvore
  - Cada nó representa uma chamada recursiva
  - O nó raiz é a chamada inicial
  - Os nós folhas são conjuntos de *0* ou *1* elemento (casos bases)

Quick Sort
Análise

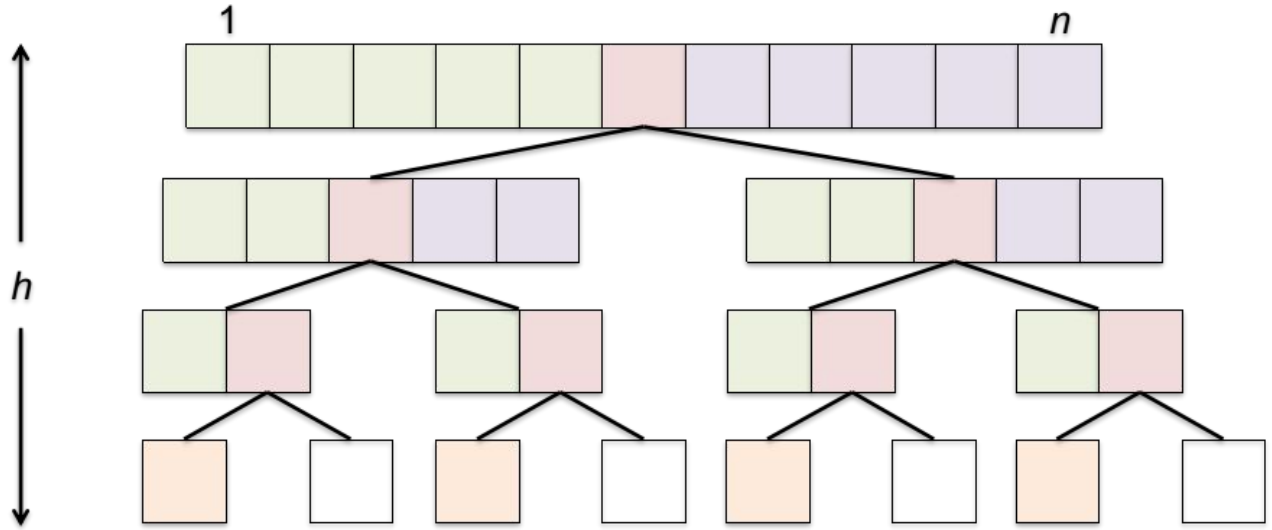


Quick Sort
Análise

- A árvore de execução possui uma altura máxima *h*
- Partição – executa *O(n)* operações
- QuickSort – executa no máximo *h x Partição*
- Logo, QuickSort executa *h x O(n)* = *O(n . h)* operações no pior caso

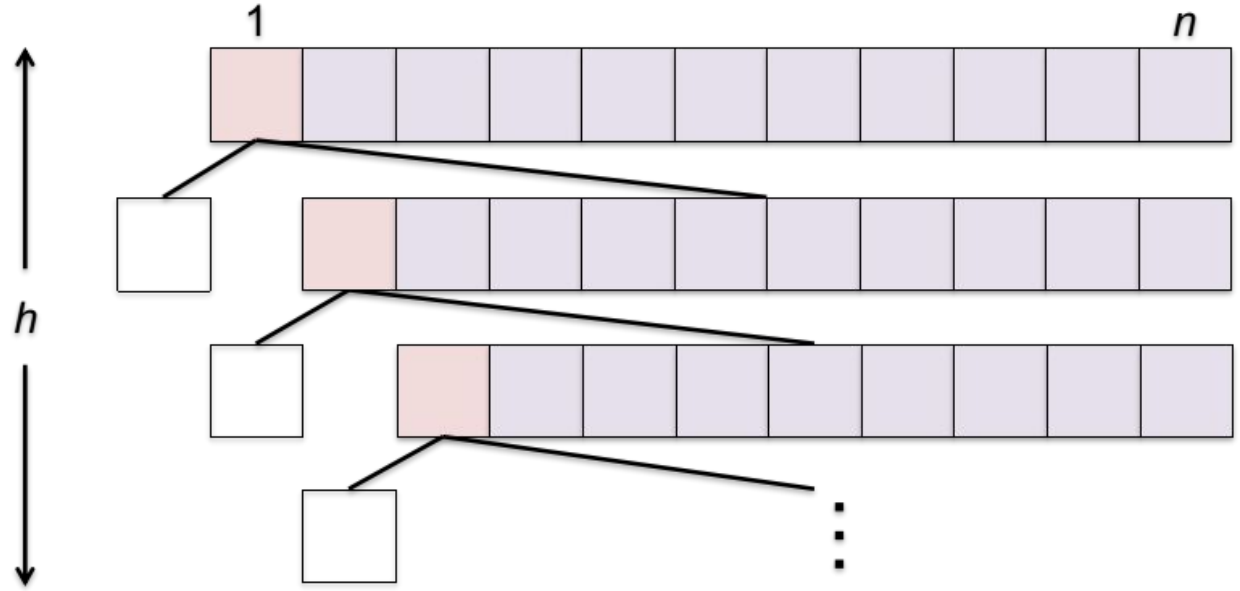
Quick Sort
Análise

- Melhor caso: $h \approx log_2 n$ $O(n\ log_2\ n)$ operações
  - Ocorre quando o problema é sempre dividido em subproblemas de igual tamanho após a partição



Quick Sort
Análise

- Pior caso: $h \approx n$ $O(n^2)$ operações
  - Ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos do arquivo já ordenado



Quick Sort
Análise

- Caso médio de acordo com Sedgewick e Flajolet (1996, p. 17):
  - $C(n) \approx 1{,}386\ n \log_2 n - 0{,}846\ n$

- Isso significa que, em média, o QuickSort executa $O(n \log_2 n)$ operações

Quick Sort
Melhorias

- O pior caso pode ser evitado empregando pequenas modificações no algoritmo
  - Para isso, basta escolher três itens quaisquer do vetor e usar a **mediana dos três** como pivô
  - Mais genericamente: mediana de *k* elementos

- Interromper as partições para vetores pequenos, utilizando um dos algoritmos básicos para ordená-los
  - Inserção ou Seleção