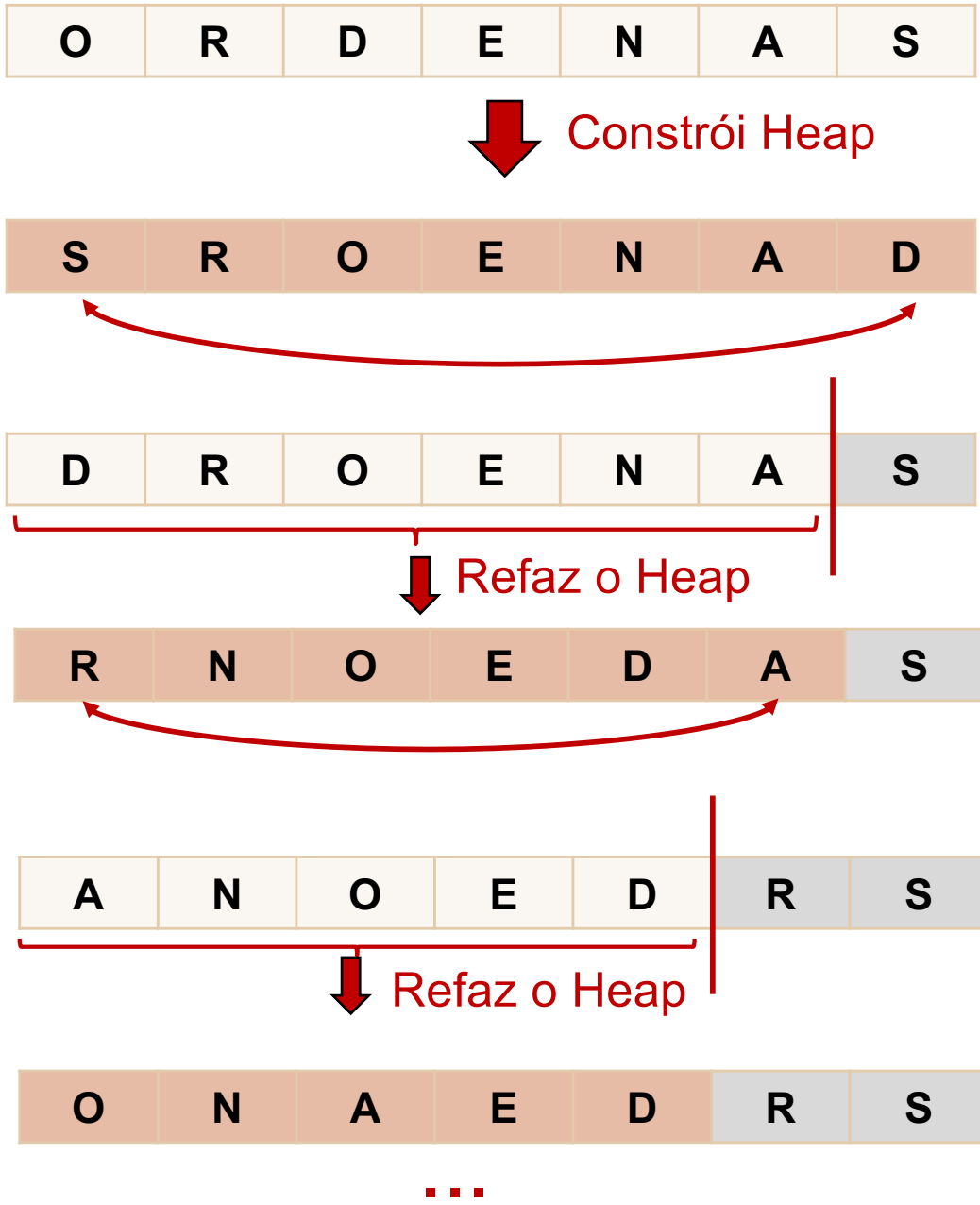
HEAPSORT – ORDENANDO COM O HEAP

Heapsort

- Algoritmo:
  1. Construir o *heap.*
  2. Troque o item na posição 1 do vetor (raiz do *heap) com o item da posição n.*
  3. Use o procedimento Refaz para reconstituir o *heap para os itens* A[1], A[2], ..., A[n - 1].
  4. Repita os passos 2 e 3 com os n - 1 itens restantes, depois com os n - 2, até que reste apenas um item.

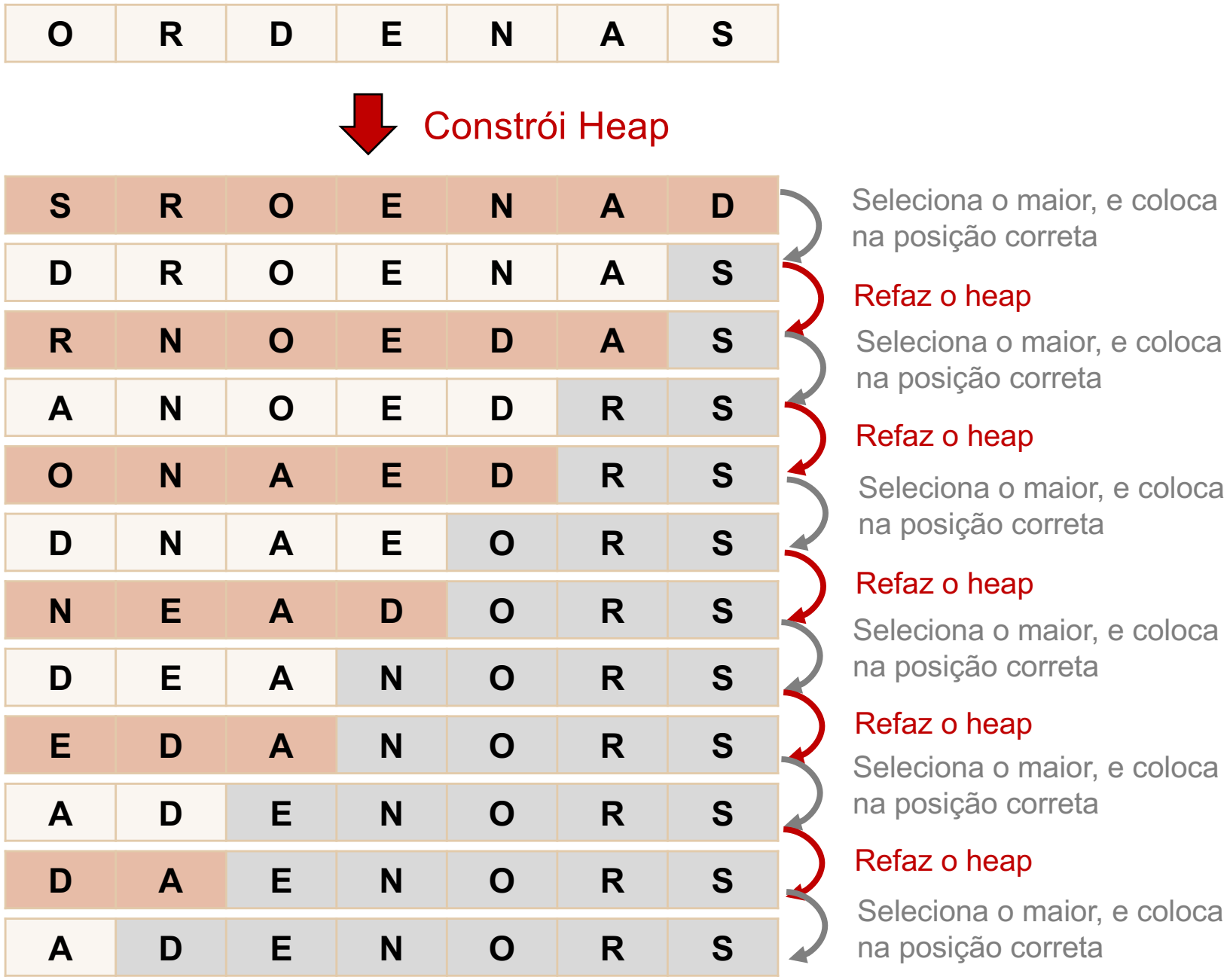
Heapsort - Exemplo



Seleciona o maior, e troca com o que está na última posição

Seleciona o 2º. maior, e troca com o que está na penúltima posição

Heapsort - Exemplo

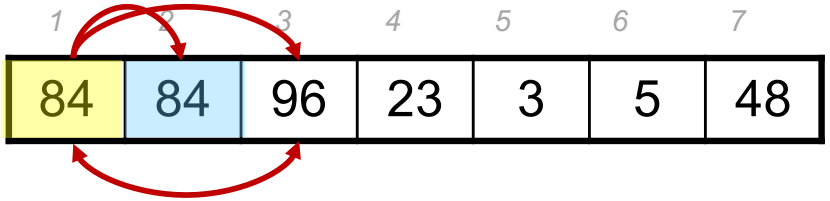


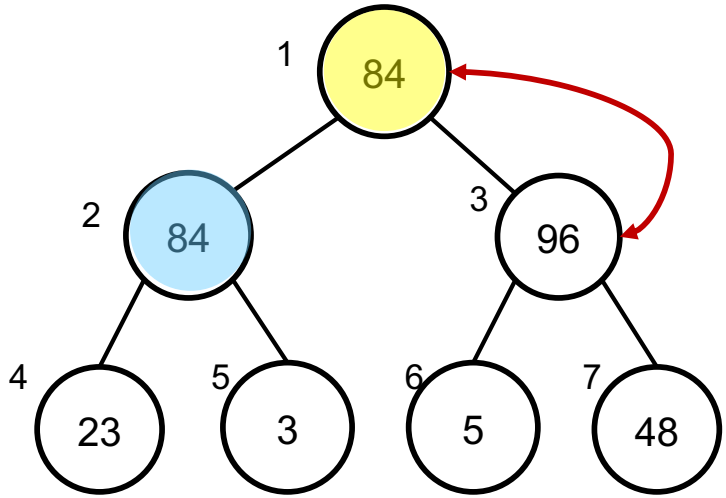
Heapsort

```
void Heapsort(Item *A, int *n) {
  int Esq, Dir;
  Item x;
  Constroi(A, n); /* constroi o heap */
  Esq = 1; Dir = *n;
  while (Dir > 1)
  { /* ordena o vetor */
    x = A[1];
    A[1] = A[Dir];
    A[Dir] = x;
    Dir--;
    Refaz(Esq, Dir, A);
  }
}
```

HEAPSORT - ANÁLISE

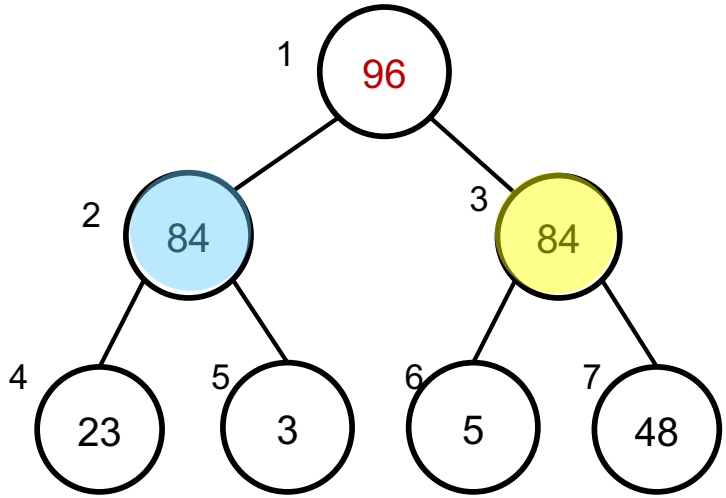
O método é estável?





O método é estável?

1	2	3	4	5	6	7
96	84	84	23	3	5	48



Inverteu a posição dos '84's

NÃO É ESTÁVEL!

Heapsort – Análise de Complexidade

- Refaz:
  - No pior caso, percorre todo um galho da árvore binária, ou seja, executa log n operações ➡ ***C(n) = O(log n)***
- Constrói:
  - Para os nós internos (n/2 elementos), chama refaz, logo executa: n/2 log n ➡ ***C(n) = O(n log n)***
- Heapsort
  - Chama Constroi – uma vez
  - Chama Refaz n-1 vezes

  } ➡ ***C(n) = O(n log n)***

Heapsort

- Vantagens:
  - O comportamento do Heapsort é sempre O(n log n), qualquer que seja a entrada.
- Desvantagens:
  - O anel interno do algoritmo é bastante complexo se comparado com o do Quicksort.
  - O Heapsort não é **estável**.
- Recomendado:
  - Para aplicações que não podem tolerar eventualmente um caso desfavorável.
  - Não é recomendado para arquivos com poucos registros, por causa do tempo necessário para construir o *heap.*