A ideia que nós acabamos de ver também pode ser adaptada para outros casos.

Por exemplo, suponha que o vetor `V[1..n]` contém apenas números inteiros positivos (todos distintos), e nós sabemos que o maior número em `V` é igual a $m > n$.

Então, nós podemos ordenar o vetor `V` com o auxílio de uma lista auxiliar `L`, através do seguinte procedimento

1. inicializar todas as posições de `L` com o valor 0
2. percorrer o vetor `V`, colocadando cada elemento k na k-ésima posição de `L`
3. percorrer a lista `L`, movendo os seus elementos maiores do que 0 de volta para o vetor `V`

Não é difícil ver que esse algoritmo executa em tempo

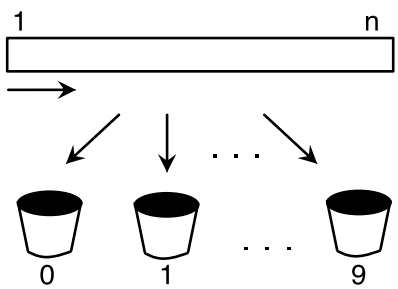
$$O(m) \quad + \quad O(n) \quad + \quad O(m) \qquad = \qquad O(n+m)$$

Logo, ele pode ser uma boa opção, quando $m < n \log n$.

Agora, considere o caso em que o vetor `V[1..n]` possui muitos elementos repetidos.

Por exemplo, suponha que os únicos números que aparecem em `V` são: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

Nesse caso, nós podemos adaptar o algoritmo acima como indicado na figura abaixo



Quer dizer, ao invés da lista auxiliar `L`, nós vamos utilizar uma estrutura de dados que funciona como uma coleção de baldes.

A ideia então é percorrer o vetor `V` colocando cada elemento no balde correspondente.

Uma vez que isso tenha sido feito, basta mover os elementos do primeiro balde de volta para o vetor `V`, em seguida os elementos do segundo balde, e assim por diante.

Qual é o tempo de execução desse algoritmo?

Bom, fazendo a suposição de que a inserção de um elemento na estrutura de dados dos baldes leva tempo $O(1)$, não é difícil ver que a primeira etapa executa em tempo $O(n)$.

E, fazendo a suposição de que a remoção de um elemento de um balde leva tempo $O(1)$, nós vemos que a segunda etapa também leva tempo $O(n)$.

Portanto, o algoritmo como um todo executa em tempo

$$O(n) \quad + \quad O(n) \qquad = \qquad O(n)$$

Ou seja, melhor impossível!