DCC302 – ESTRUTURA DE DADOS I

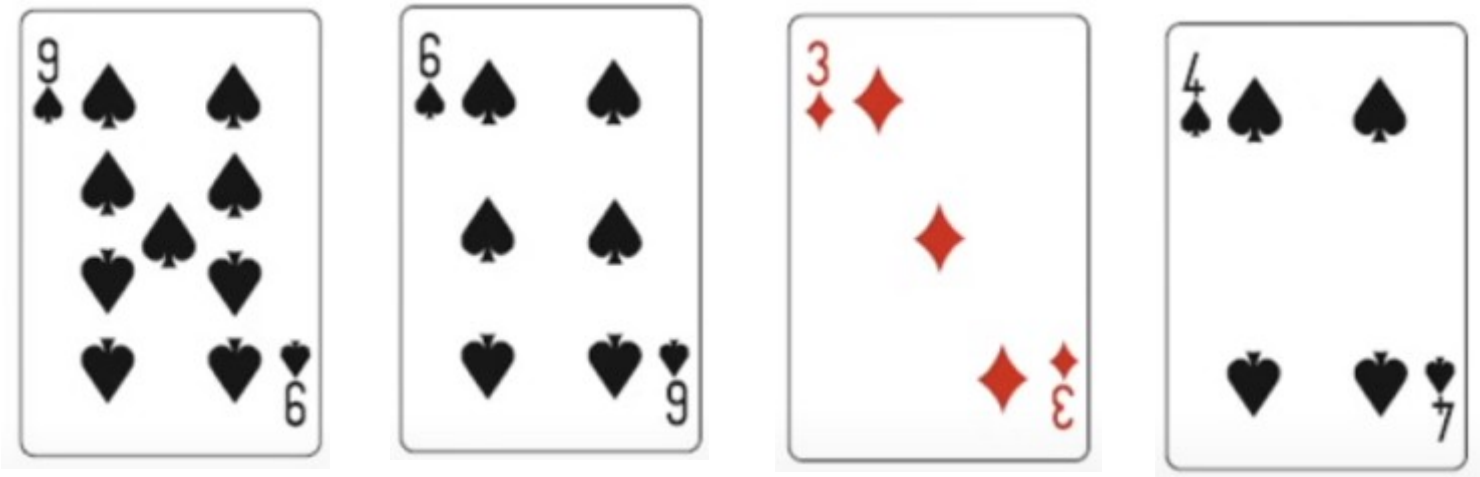
Aula 15.3 – Insertion Sort

Insertion Sort

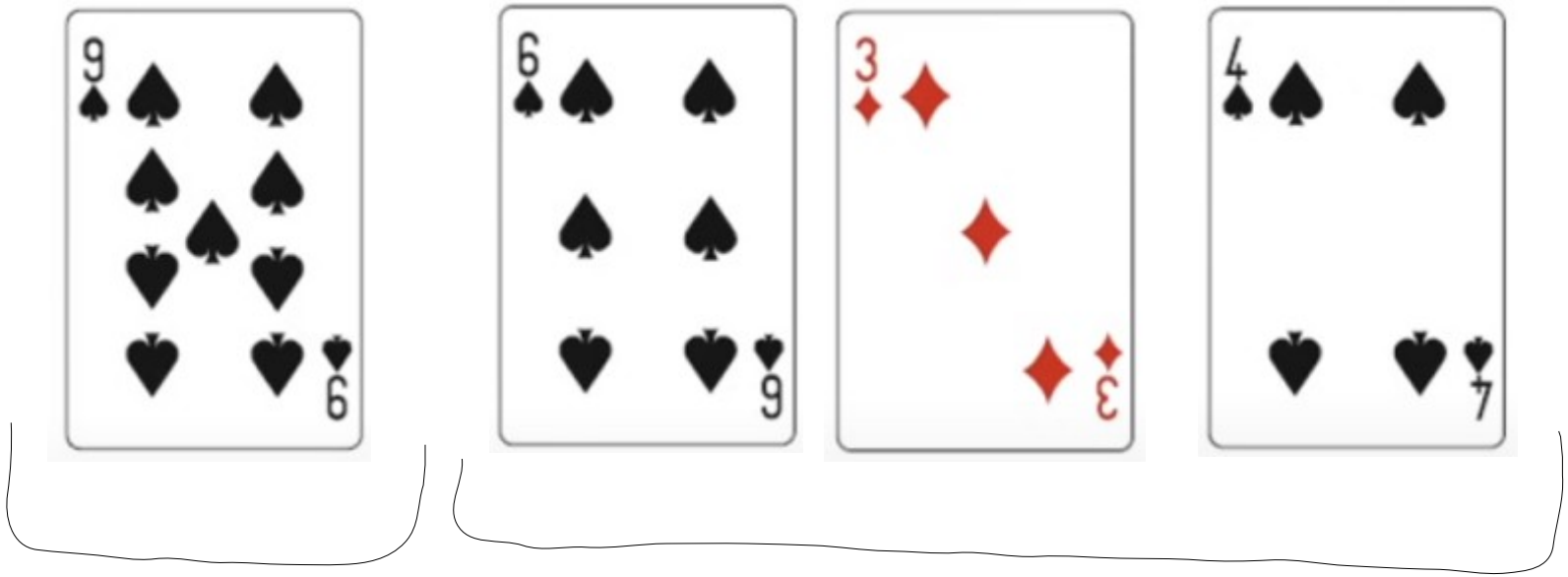
- Este é um algoritmo de ordenação baseado em comparação **in-place**. Aqui, uma sublista é mantida, a qual está sempre ordenada. Por exemplo, a parte inferior de um array é mantida ordenada. Um elemento que é para ser '**inserido**' nesta sub-lista ordenada, tem que encontrar seu lugar apropriado e então tem que ser inserido lá. Daí o nome, ordenação por **inserção**.

- O array é percorrido sequencialmente e os itens não ordenados são movidos e inseridos na sub-lista ordenada (no mesmo array). Este algoritmo não é adequado para grandes conjuntos de dados, pois sua complexidade média e de pior caso são de $O(n^2)$, onde n é o número de itens.

Insertion Sort - Exemplo



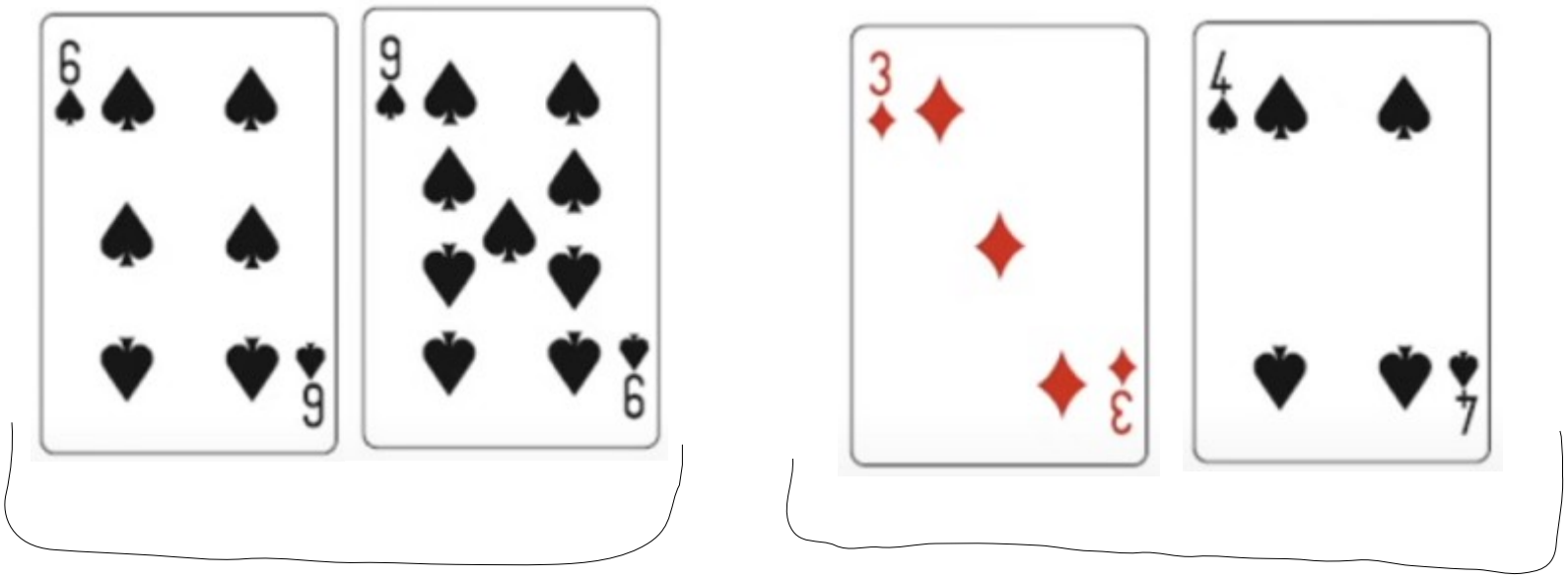
Insertion Sort - Exemplo



Subconjunto Ordenado

Subconjunto Desordenado

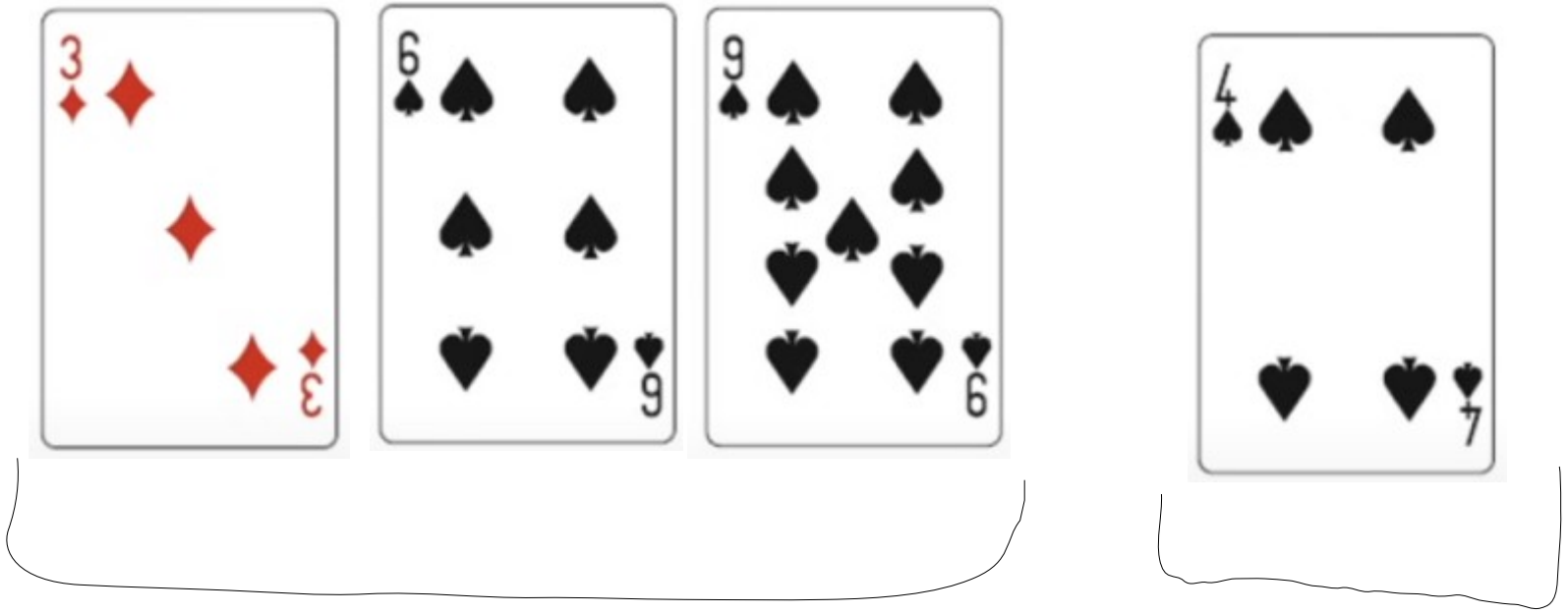
Insertion Sort - Exemplo



Subconjunto Ordenado

Subconjunto Desordenado

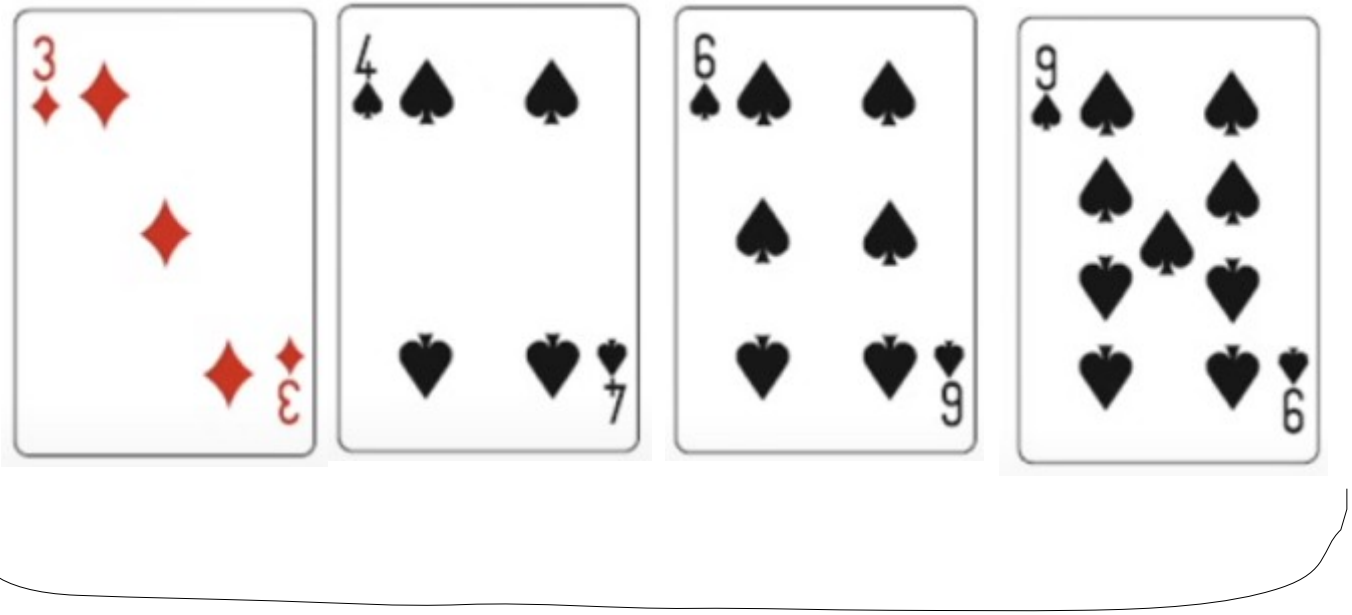
Insertion Sort - Exemplo



Subconjunto Ordenado

Subconjunto Desordenado

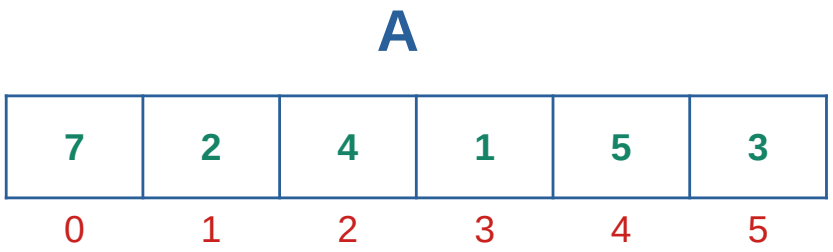
Insertion Sort - Exemplo



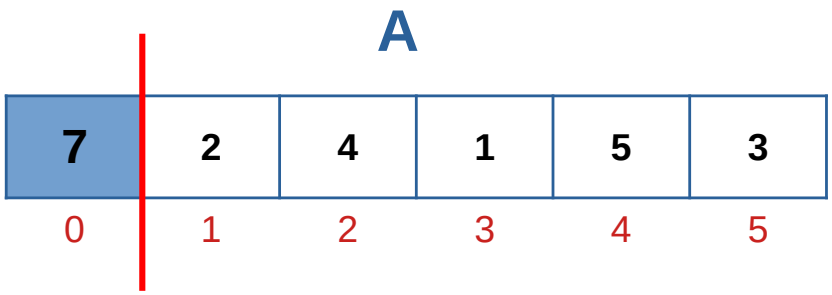
Subconjunto Ordenado

Subconjunto Desordenado

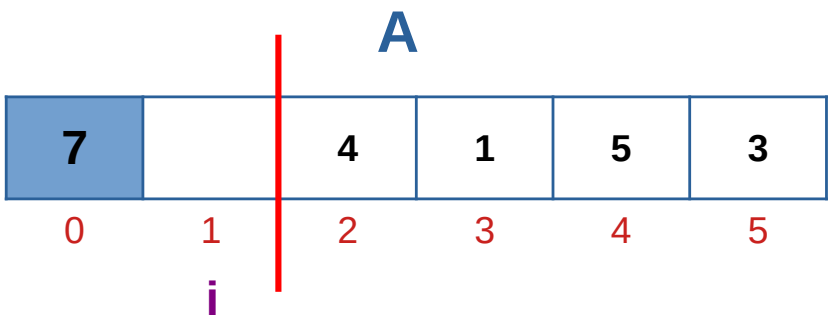
Insertion Sort



Insertion Sort



Insertion Sort

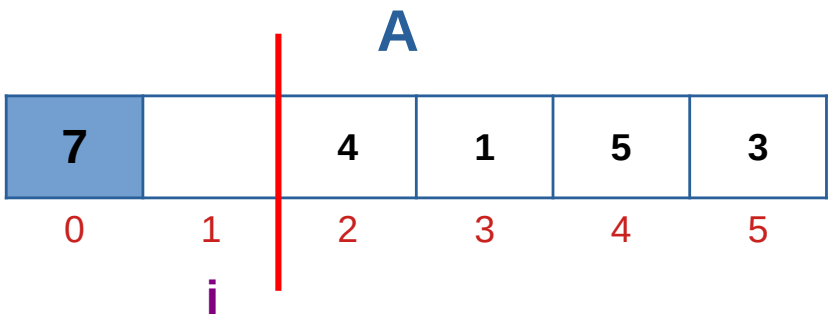


ValueToInsert ← 2 (A[i])

Algoritmo:

1) Vamos ter um i que vai iniciar do índice 1, esse vai ser o valor que vai ser inserido no conjunto ordenado. (O conjunto ordenado na primeira passada é somente o índice 0.)

Insertion Sort



valueToInsert ← 2 (A[i])

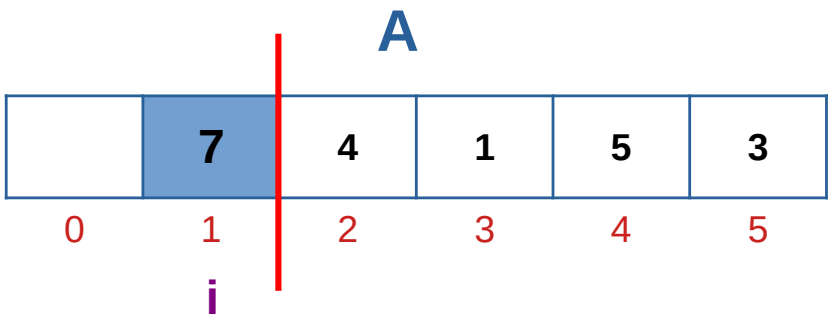
holePosition ← i

Algoritmo:

1) Vamos ter um **i** que vai iniciar do **índice 1**, esse vai ser o valor que vai ser inserido no conjunto ordenado. (O conjunto ordenado na primeira passada é somente o índice 0.)

2) No índice i foi criado um buraco (**holePosition**) onde agora vai ser possível aumentar o tamanho da sublista ordenada.

Insertion Sort



valueToInsert ← 2 (A[i])

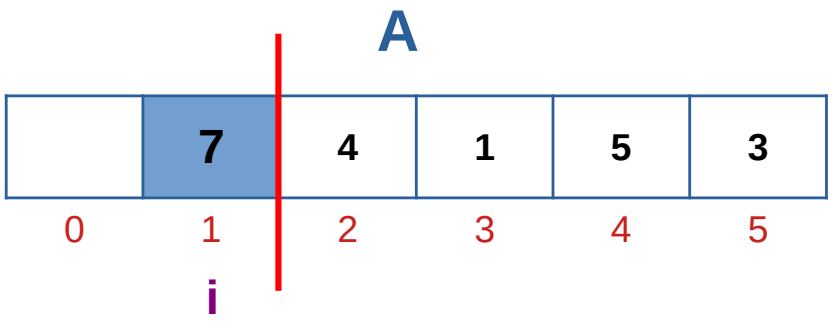
HolePosition ← i

Algoritmo:

1) Vamos ter um i que vai iniciar do índice 1, esse vai ser o valor que vai ser inserido no conjunto ordenado. (O conjunto ordenado na primeira passada é somente o índice 0.)

2) No índice i foi criado um buraco (**holePosition**) onde agora vai ser possível aumentar o tamanho da sublista ordenada.

Insertion Sort



valueToInsert ← 2 (A[i])

HolePosition ← i

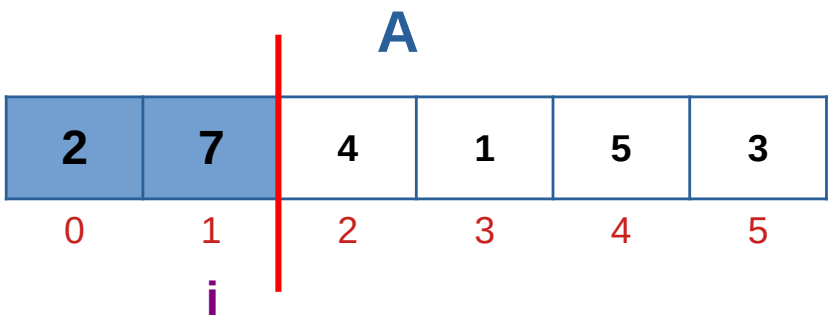
Algoritmo:

1) Vamos ter um i que vai iniciar do índice 1, esse vai ser o valor que vai ser inserido no conjunto ordenado. (O conjunto ordenado na primeira passada é somente o índice 0.)

2) No índice i foi criado um buraco (**holePosition**) onde agora vai ser possível aumentar o tamanho da sublista ordenada.

3) O holeposition é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que valueToInsert, troca-se eles de lugar

Insertion Sort



valueToInsert ← 2 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

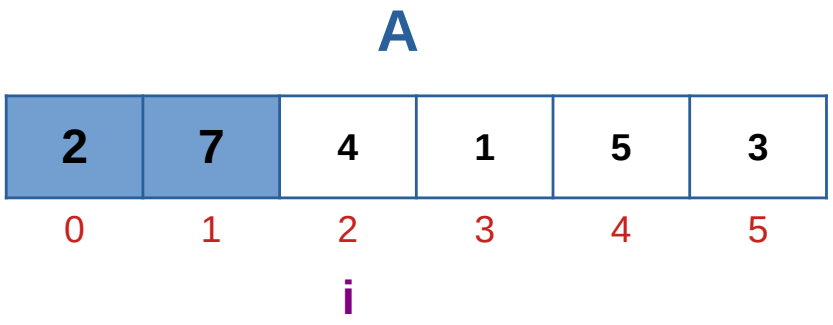
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 4 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

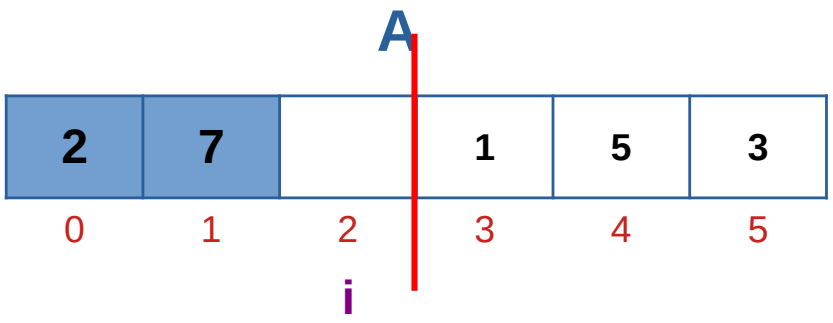
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 4 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

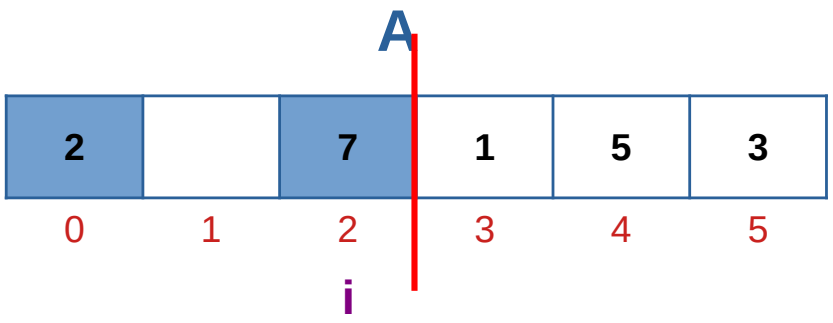
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 4 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

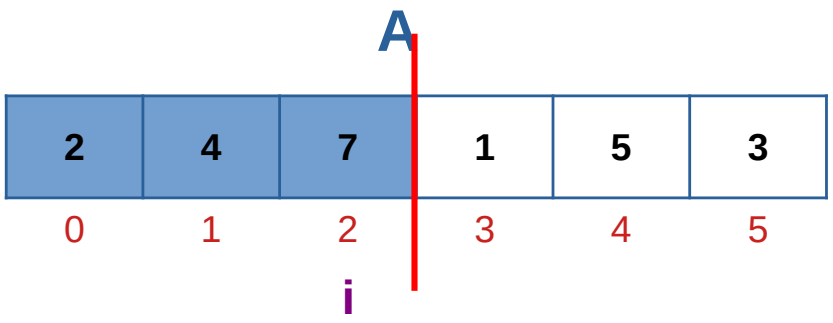
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 4 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

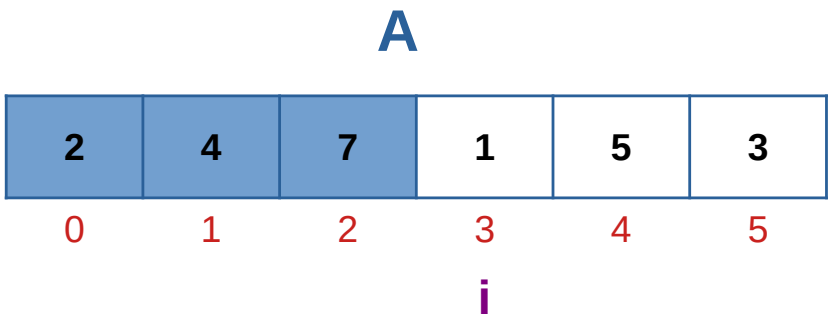
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 1 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

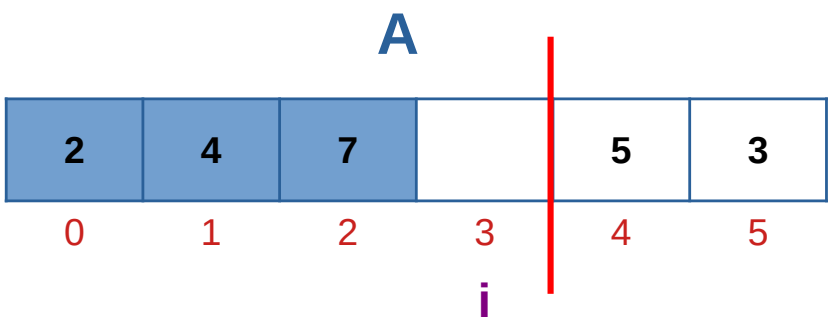
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 1 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

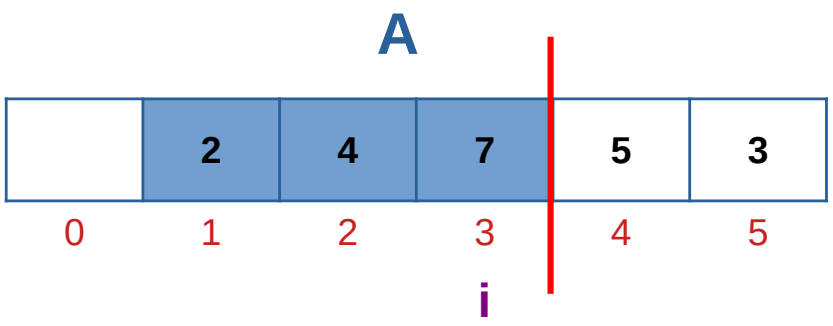
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 1 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

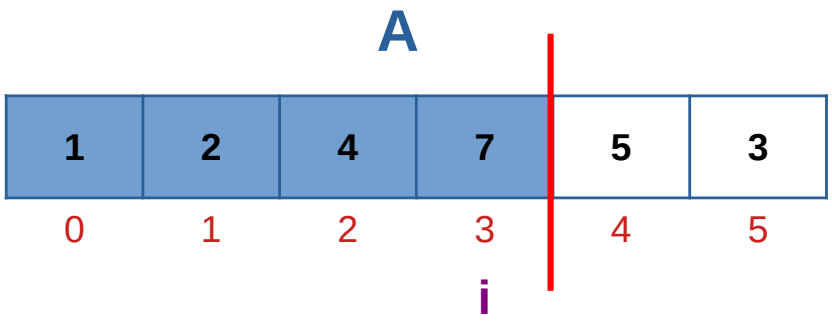
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 1 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

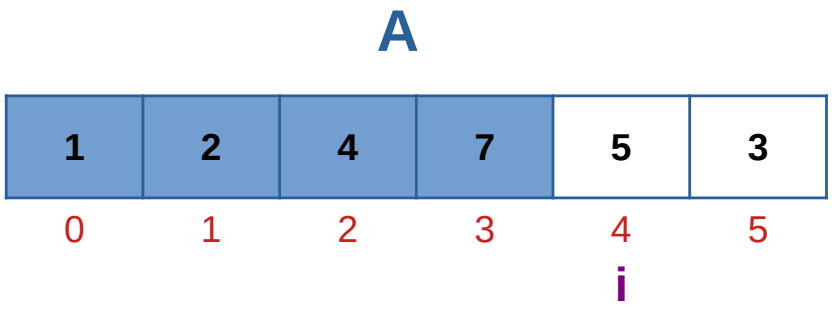
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

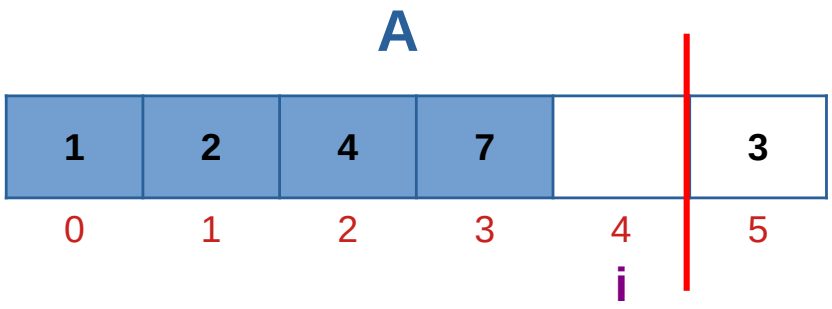
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

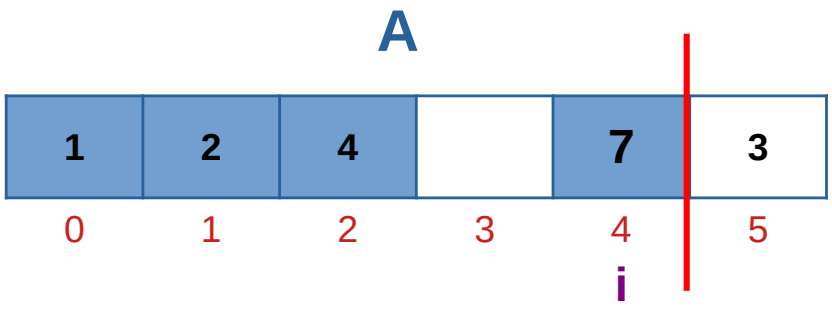
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

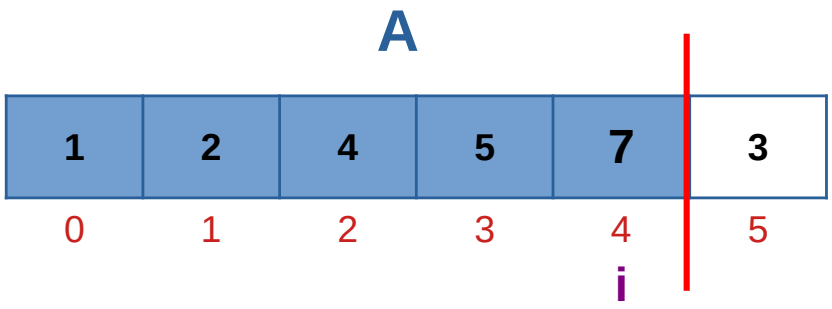
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

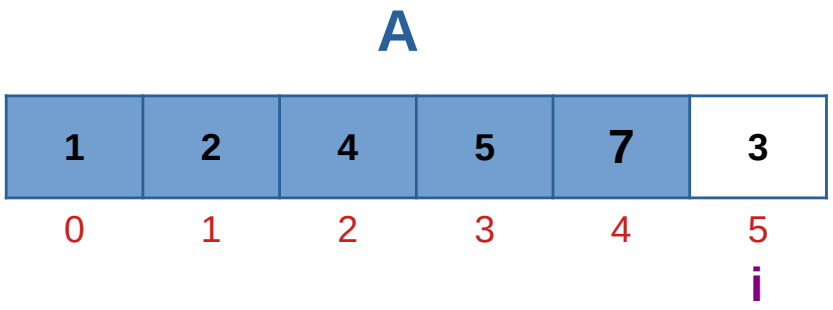
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

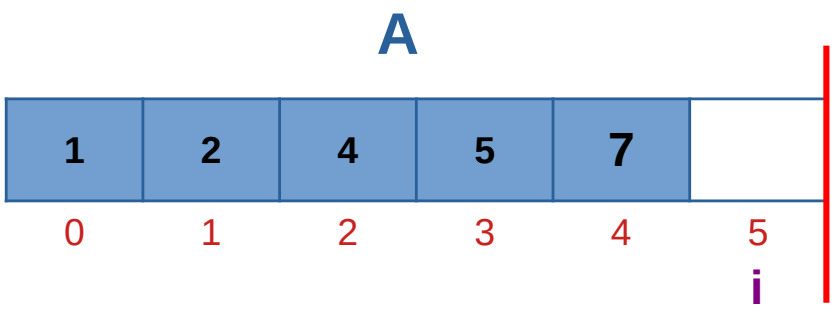
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

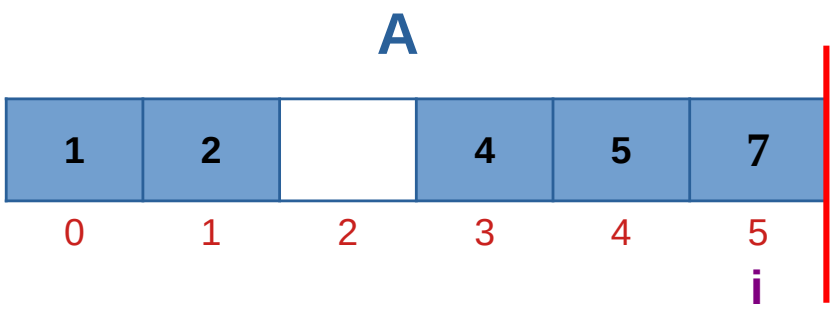
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

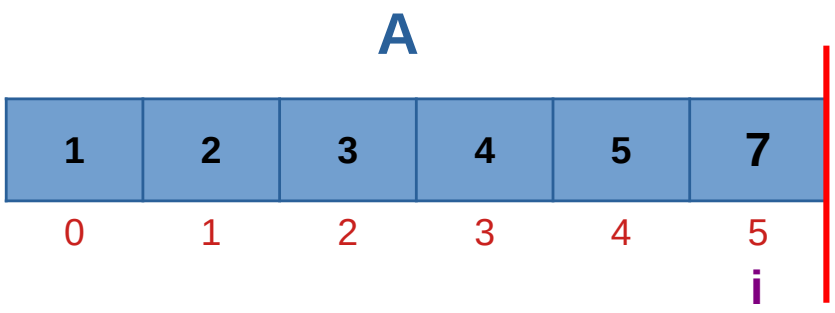
Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

Algoritmo:

...

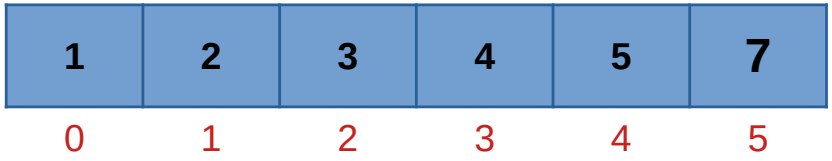
3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort

Ordenado!

A



valueToInsert ← 5 (A[i])

HolePosition ← i

A[holePosition] = valueToInsert

Algoritmo:

...

3) O **holeposition** é um índice de controle. Eu vou pegar ele e ir comparando com o seu anterior (hole – 1) se o valor que esta em A[hole-1] for maior que **valueToInsert**, troca-se eles de lugar

4) O que foi feito no ponto 3) foi encontrar o lugar para onde o valor a ser inserido vai entrar. Ou seja abre-se um buraco na posição onde vai ser inserido o **valueToInsert**

Insertion Sort - Pseudocode

```
procedure insertionSort( A : array of items )
   int holePosition
   int valueToInsert

   for i = 1 to length(A) inclusive do:

      /* select value to be inserted */
      valueToInsert = A[i]
      holePosition = i

      /*locate hole position for the element to be inserted */

      while holePosition > 0 and A[holePosition-1] > valueToInsert do:
         A[holePosition] = A[holePosition-1]
         holePosition = holePosition -1
      end while

      /* insert the number at hole position */
      A[holePosition] = valueToInsert

   end for

end procedure
```

Insertion Sort

- Implementar em c

Insertion Sort

- Teste