Programação Estruturada

Prof. Paulo Henrique Pisani

http://professor.ufabc.edu.br/~paulo.pisani/

novembro/2018

Tópicos

- Ordenação:
  - Selection sort
  - Insertion sort

Ordenação

- Ordenação é o processo de **rearranjar uma sequência de elementos em ordem ascendente ou descendente**, de acordo com a **chave** de cada elemento;
- Um dos principais objetivos de realizar a ordenação é **facilitar a recuperação** dos elementos por sua chave.

Algoritmos de ordenação

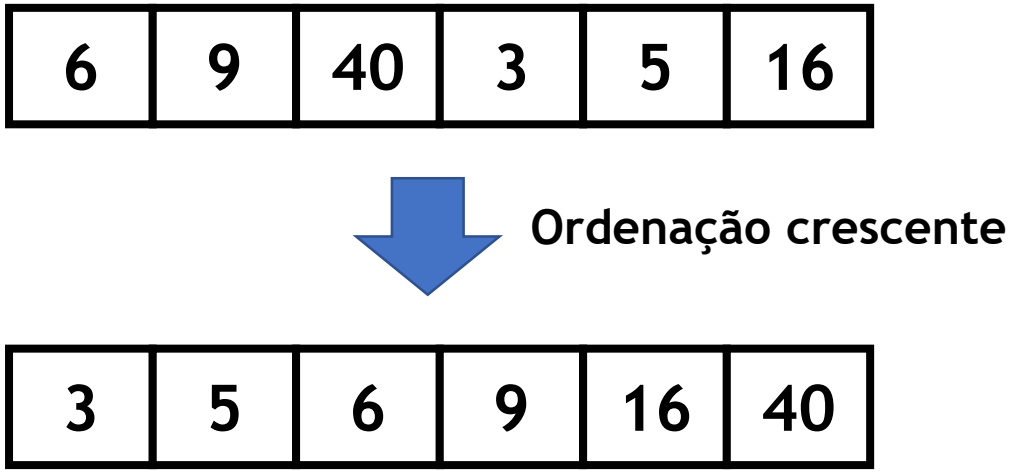
- Os algoritmos de ordenação podem ser divididos entre os baseados em comparação:
  - Bubble sort;
  - Selection sort;
  - Insertion sort;
  - Merge sort;
  - Quick sort;
  - Heap sort.

Veremos esses algoritmos na aula de hoje.

- E os baseados em distribuição:
  - Count sort;
  - Radix sort;
  - Bucket sort.

Ordenação

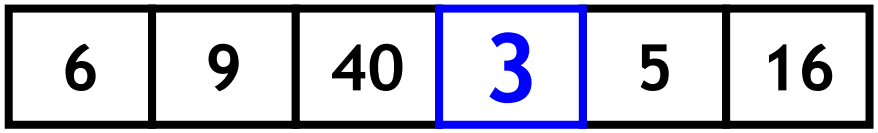
- Faremos a **ordenação de elementos em vetores/arrays** nesta aula, mas podemos aplicar os algoritmos de ordenação para outras estruturas (e.g. listas ligadas).



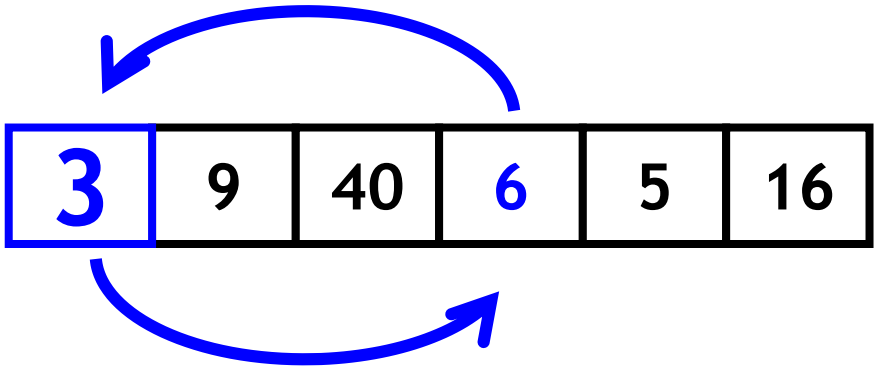
Selection sort

Selection sort

- Ideia geral:
  - Encontra menor elemento:

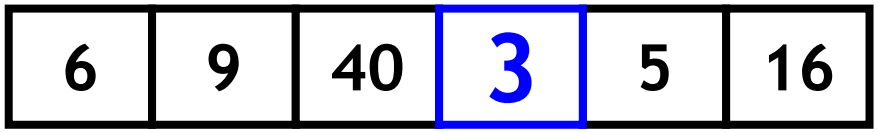


  - Troca menor elemento com o primeiro elemento do vetor:

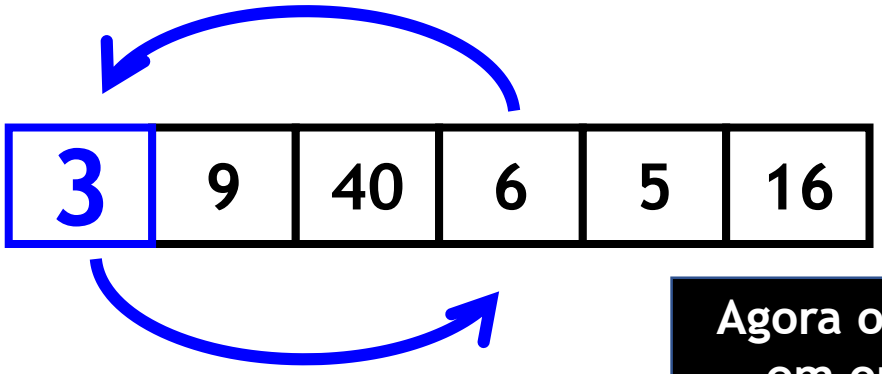


Selection sort

- Ideia geral:
  - Encontra menor elemento:



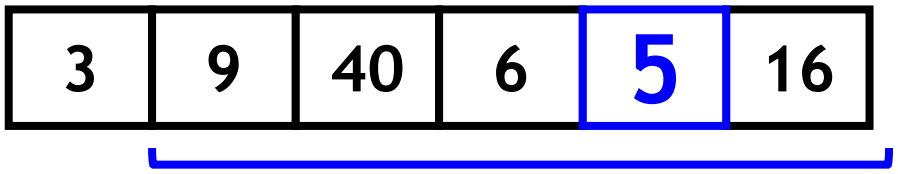
  - Troca menor elemento com o primeiro elemento do vetor:



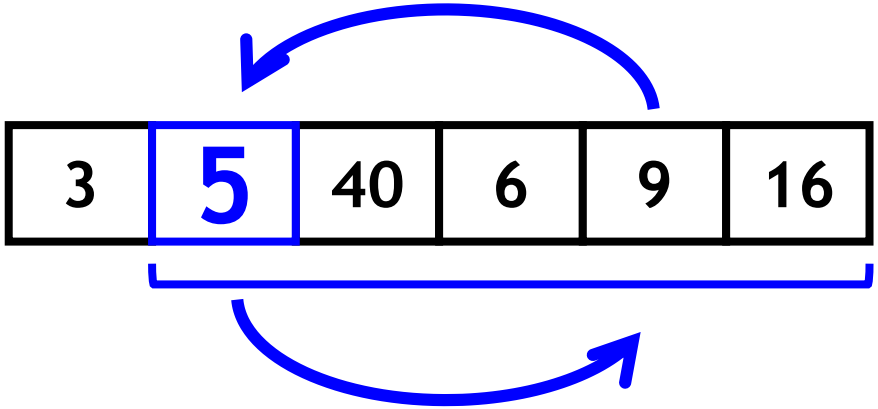
Agora o primeiro elemento já está em ordem. Então repetimos o processo para o restante do vetor.

Selection sort

- Ideia geral:
  - Encontra menor elemento no restante do vetor:

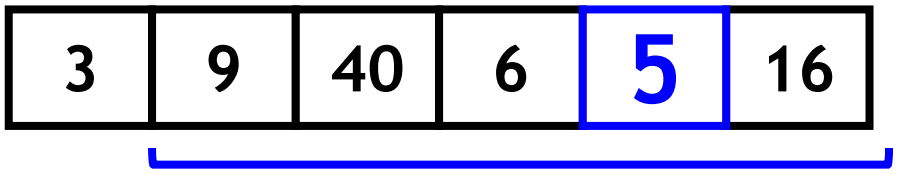


  - Troca segundo menor elemento com o segundo elemento do vetor:

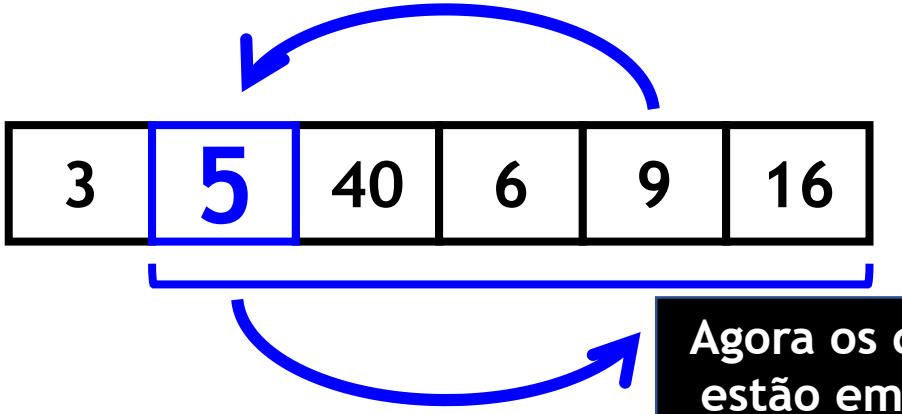


Selection sort

- Ideia geral:
  - Encontra menor elemento no restante do vetor:



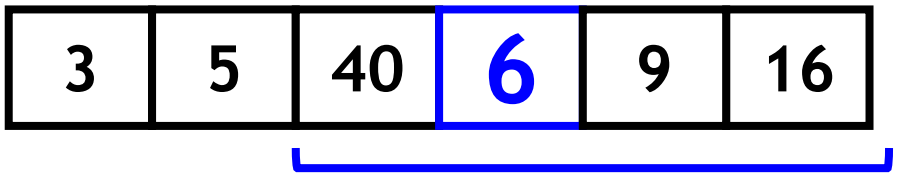
  - Troca segundo menor elemento com o segundo elemento do vetor:



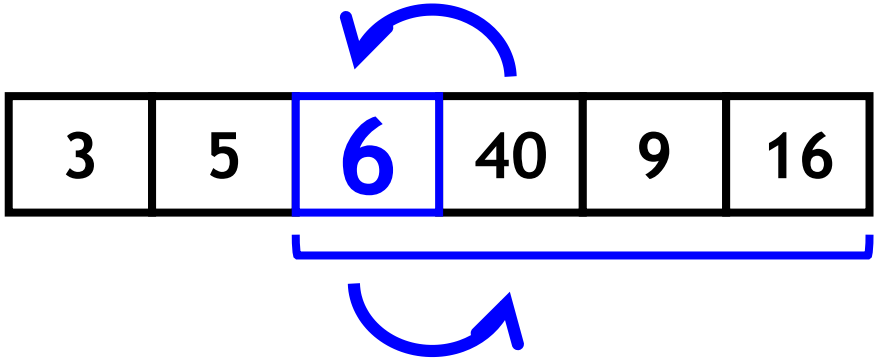
Agora os dois primeiros elementos já estão em ordem. Então repetimos o processo para o restante do vetor.

Selection sort

- Ideia geral:
  - Encontra menor elemento no restante do vetor:

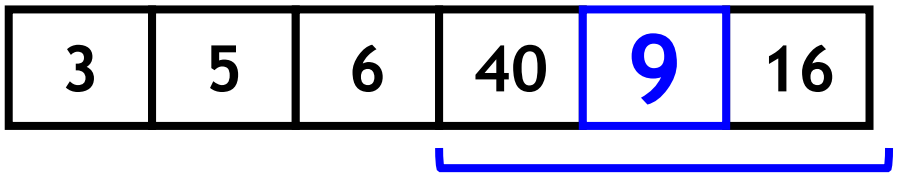


  - Troca menor elemento com o terceiro elemento do vetor:

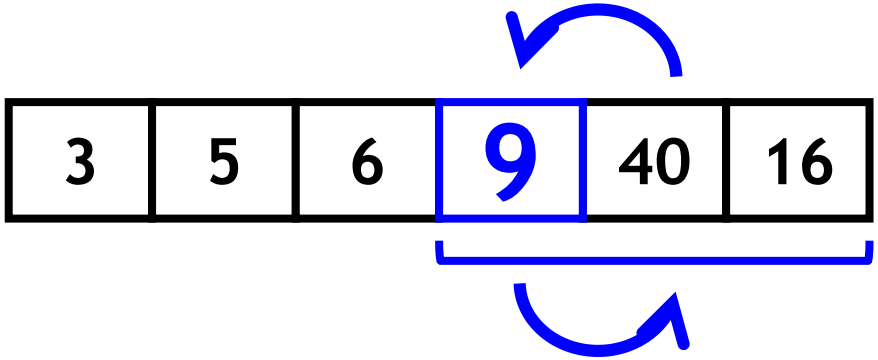


Selection sort

- Ideia geral:
  - Encontra menor elemento no restante do vetor:

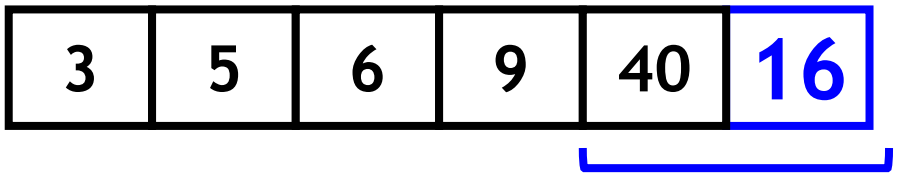


  - Troca menor elemento com o quarto elemento do vetor:

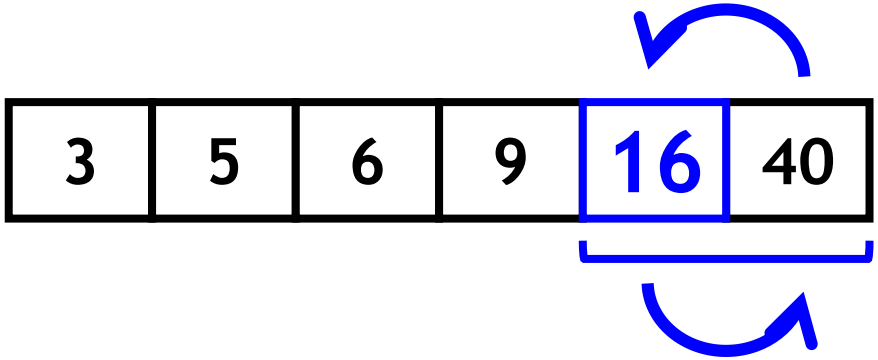


Selection sort

- Ideia geral:
  - Encontra menor elemento no restante do vetor:



  - Troca menor elemento com o quinto elemento do vetor:



Após isso, o algoritmo é finalizado!

Selection sort

```
void selectionSort(int *v, int n) {
    int count=0;
    int i;
    for (i = 0; i < n-1; i++) {
        int indice_menor = i;
        int j;
        for (j = i+1; j < n; j++)
            if (v[j] < v[indice_menor])
             indice_menor = j;

        int tmp = v[i];
        v[i] = v[indice_menor];
        v[indice_menor] = tmp;
    }
}
```

Encontra o menor elemento no restante do vetor.

Troca o menor com o elemento atual.

Quantas comparações de elementos do vetor realizamos nesse algoritmo?

```
void selectionSort(int *v, int n) {
    int count=0;
    int i;
    for (i = 0; i < n-1; i++) {
        int indice_menor = i;
        int j;
        for (j = i+1; j < n; j++)
            if (v[j] < v[indice_menor])
             indice_menor = j;

        int tmp = v[i];
        v[i] = v[indice_menor];
        v[indice_menor] = tmp;
    }
}
```

Encontra o menor elemento no restante do vetor.

Troca o menor com o elemento atual.

Análise do selection sort

- Número de comparações no pior caso:

$$\frac{n(n-1)}{2}$$

- Número de comparações no melhor caso:

$$\frac{n(n-1)}{2}$$

Ok, mas e para ordenar em **ordem decrescente**?

Ok, mas e para ordenar em **ordem decrescente**?

Ao invés de trocar pelo menor elemento, trocamos pelo maior elemento!

No algoritmo isso implica apenas em mudar nossa comparação de:

```
if (v[j] < v[indice_menor])
```

Para:

```
if (v[j] > v[indice_menor])
```

Também é recomendável renomear a variável indice_menor para indice_maior

Ok, mas e para ordenar em **ordem decrescente?**

Ao invés de trocar pelo menor elemento, trocamos pelo maior elemento!

No algoritmo isso implica apenas em mudar nossa comparação de:

```
if (v[j] < v[indice_menor])
```

Para:

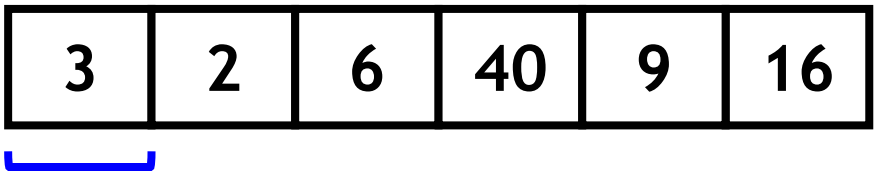
```
if (v[j] > v[indice_maior])
```

Insertion sort

Insertion sort

- Ideia geral:
  - Inicia com subvetor de um elemento (primeiro elemento do vetor);
  - Depois percorre os demais elementos (do segundo em diante) e os insere na posição correta no subvetor ordenado.

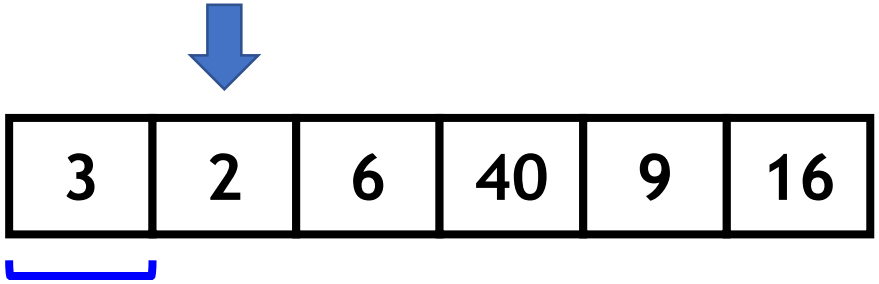
Insertion sort



Subvetor ordenado

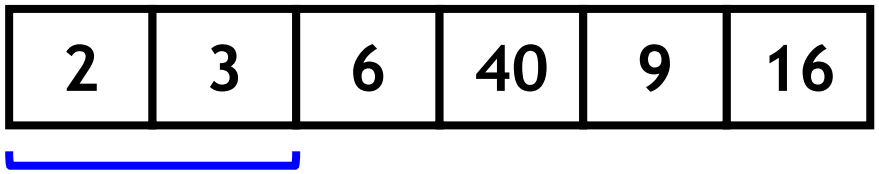
Insertion sort

Onde este elemento deveria ser inserido para manter o subvetor ordenado?



Subvetor ordenado

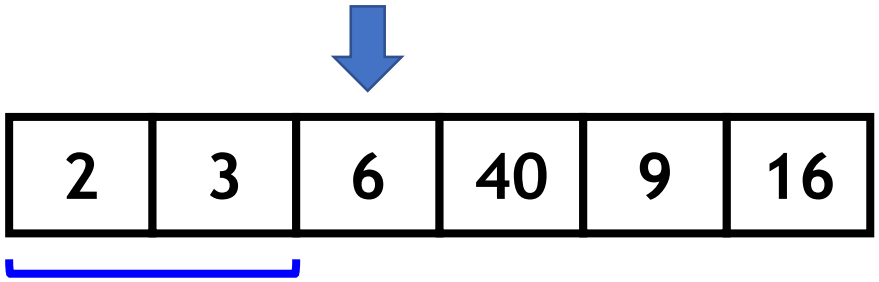
Insertion sort



Subvetor ordenado

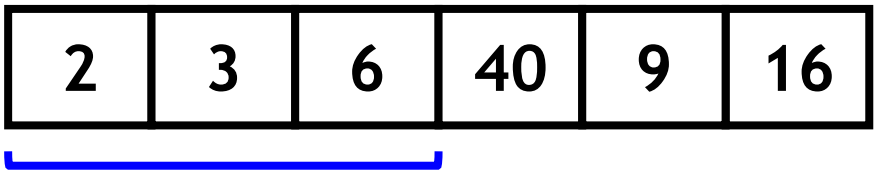
Insertion sort

Onde este elemento deveria ser inserido para manter o subvetor ordenado?



Subvetor ordenado

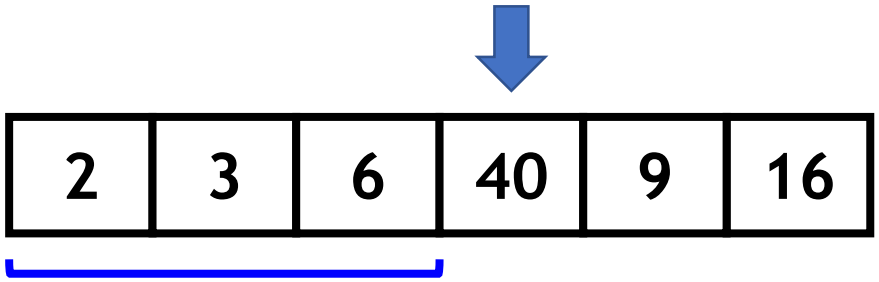
Insertion sort



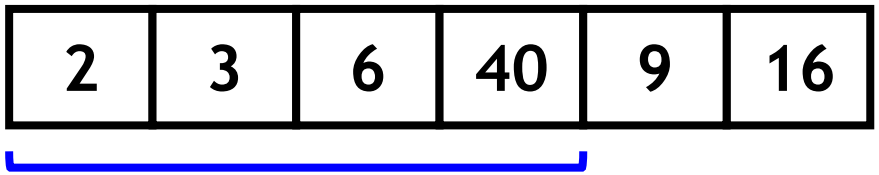
Subvetor ordenado

Insertion sort

Onde este elemento deveria ser inserido para manter o subvetor ordenado?

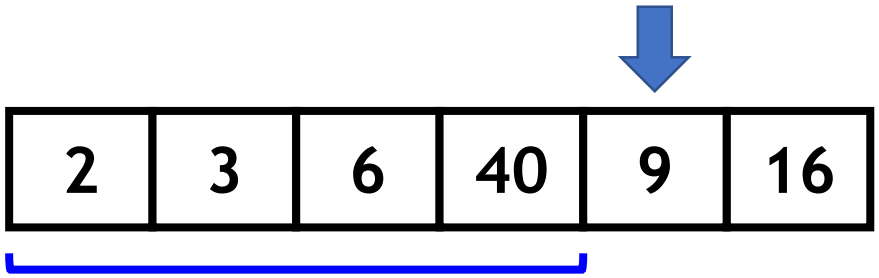


Insertion sort

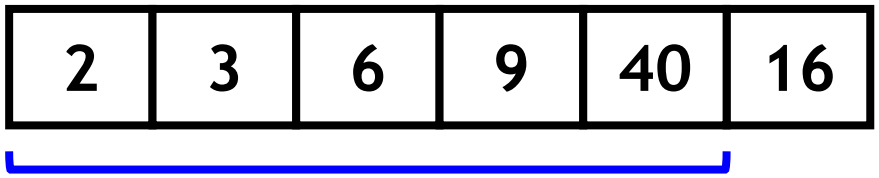


Insertion sort

Onde este elemento deveria ser inserido para manter o subvetor ordenado?



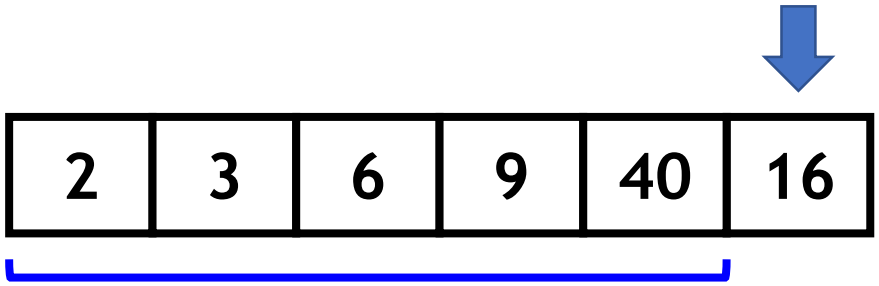
Insertion sort



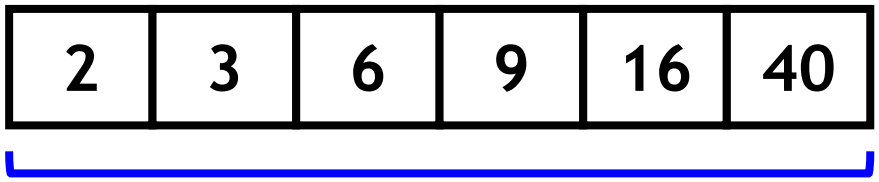
Subvetor ordenado

Insertion sort

Onde este elemento deveria ser inserido para manter o subvetor ordenado?



Insertion sort



Insertion sort

```
void insertion_sort(int *v, int n) {
    int i, k;
    for (i = 1; i < n; i++) {
        int item_atual = v[i];

        int indice_para_inserir = i;
        for (k = i - 1; k >= 0
          && item_atual < v[k]; k--) {
            v[k+1] = v[k];
            indice_para_inserir--;
        }
        v[indice_para_inserir] = item_atual;
    }
}
```

Guarda elemento atual

Encontra índice para inserção e desloca elementos para a direita.

Insere elemento atual.

Insertion sort

```
void insertion_sort(int *v, int n) {
    int i, k;
    for (i = 1; i < n; i++) {
        int item_atual = v[i];

        int indice_para_inserir = i;
        for (k = i - 1; k >= 0
          && item_atual < v[k]; k--) {
            v[k+1] = v[k];
            indice_para_inserir--;
        }
        v[indice_para_inserir] = item_atual;
    }
}
```

Guarda elemento atual

Encontra índice para inserção e desloca elementos para a direita.

Insere elemento atual.

Veja que apenas fazemos deslocamentos até encontrar a posição correta para inserção.

Quantas comparações de elementos do vetor realizamos nesse algoritmo?

```
void insertion_sort(int *v, int n) {
    int i, k;
    for (i = 1; i < n; i++) {
        int item_atual = v[i];

        int indice_para_inserir = i;
        for (k = i - 1; k >= 0
          && item_atual < v[k]; k--) {
            v[k+1] = v[k];
            indice_para_inserir--;
        }
        v[indice_para_inserir] = item_atual;
    }
}
```

Guarda elemento atual

Encontra índice para inserção e desloca elementos para a direita.

Insere elemento atual.

Ok, mas e para ordenar em **ordem decrescente**?

Ok, mas e para ordenar em **ordem decrescente**?

Assim como no selection sort, apenas temos que mudar a condição de comparação de:

```
item_atual < v[k]
```

Para:

```
item_atual > v[k]
```

Análise do insertion sort

- Número de comparações no pior caso:

$$\frac{n(n-1)}{2}$$

- Número de comparações no melhor caso:

$$n-1$$

Exemplos dos algoritmos

- Selection sort:
  - https://visualgo.net/en/sorting?slide=7
- Insertion sort:
  - https://visualgo.net/en/sorting?slide=8

Bibliografia básica

- PINHEIRO, F. A. C. Elementos de programação em C. Porto Alegre, RS: Bookman, 2012.
- FORBELLONE, A. L. V.; EBERSPACHER, H. F. Lógica de programação: a construção de algoritmos e estruturas de dados. 3ª edição. São Paulo, SP: Prentice Hall, 2005.
- CORMEN, T. H.; LEISERSON, C. E.; RIVEST, R. L.; STEIN, C. Algoritmos: teoria e prática. 2ª edição. Rio de Janeiro, RJ: Campus, 2002.

Bibliografia complementar

- AGUILAR, L. J. Programação em C++: algoritmos, estruturas de dados e objetos. São Paulo, SP: McGraw-Hill, 2008.
- DROZDEK, A. Estrutura de dados e algoritmos em C++. São Paulo, SP: Cengage Learning, 2009.
- KNUTH D. E. The art of computer programming. Upper Saddle River, USA: Addison- Wesley, 2005.
- SEDGEWICK, R. Algorithms in C++: parts 1-4: fundamentals, data structures, sorting, searching. Reading, USA: Addison-Wesley, 1998.
- SZWARCFITER, J. L.; MARKENZON, L. Estruturas de dados e seus algoritmos. 3a edição. Rio de Janeiro, RJ: LTC, 1994.
- TEWNENBAUM, A. M.; LANGSAM, Y.; AUGENSTEIN, M. J. Estruturas de dados usando C. São Paulo, SP: Pearson Makron Books, 1995.