Faculdade de
Computação e Informática

Métodos de ordenação

Bubble, selection, insertion sort

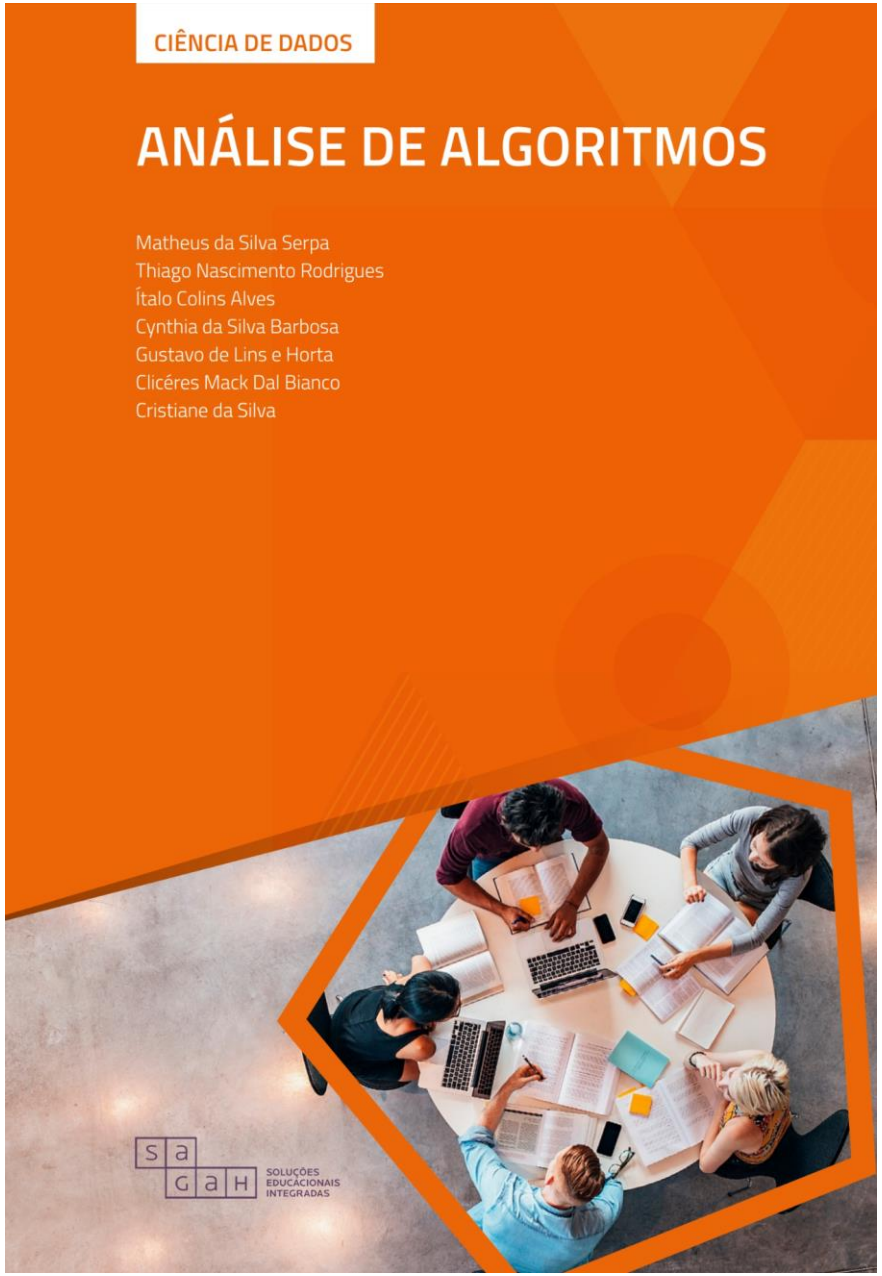
Algoritmos e Programação II – Turma 02D

2º semestre de 2023

Prof. André Kishimoto
Prof. Gustavo Scalabrini Sampaio
Prof. Leandro Carlos Fernandes

(Conteúdo adaptado do material elaborado e gentilmente cedido pela profa. Ana Grasielle Dionisio Correa e prof. Tomaz Mikio Sasaki)

Sugestão de leitura



Ordenação de dados
Métodos simples

Disponível na

Minha Biblioteca

Ordenação de elementos

Exemplo:

Array inicial:

Ordenação de elementos

Exemplo:

Array inicial:

Array ordenado:

Ordenação de elementos

Exemplo:

Array inicial:

Bubble sort

Fonte: https://en.m.wikipedia.org/wiki/File:Bubbles_3D.jpg

Bubble sort (analogia)

As cartas já estão na sua mão.

Você arruma a ordem de cada par de cartas vizinhas.

Fonte: https://pt.wikipedia.org/wiki/Ficheiro:Bridge_declarer.jpg

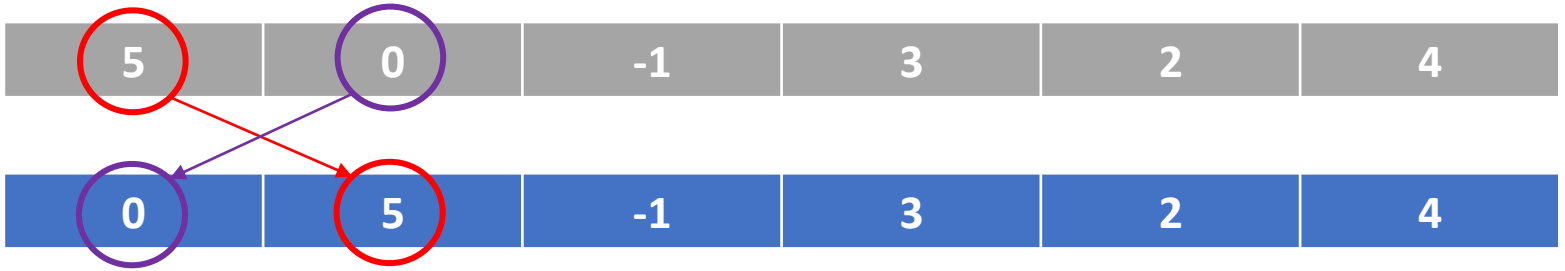
Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.

5	0	-1	3	2	4

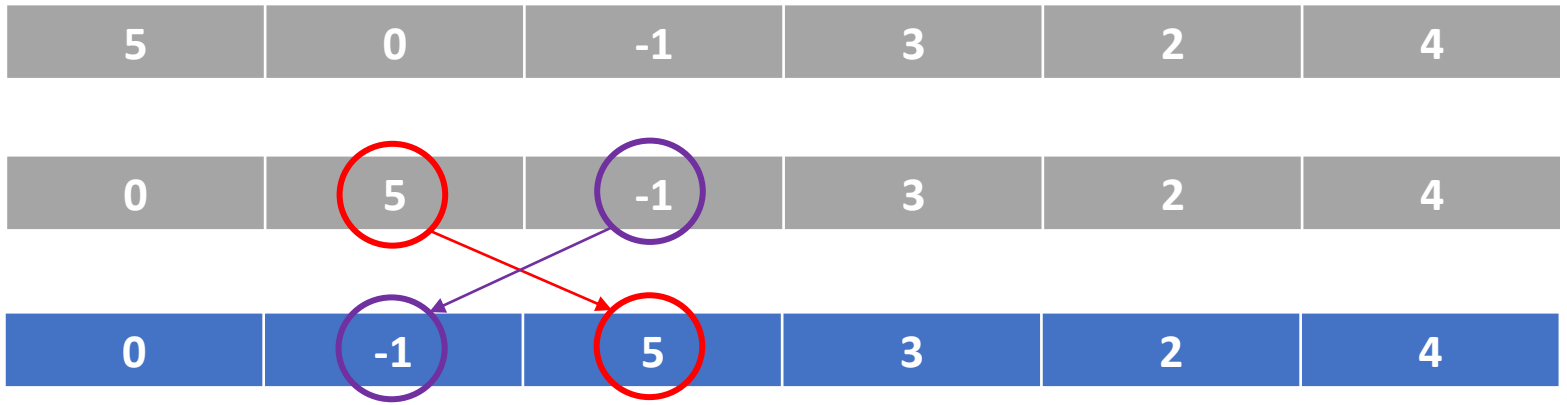
Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.



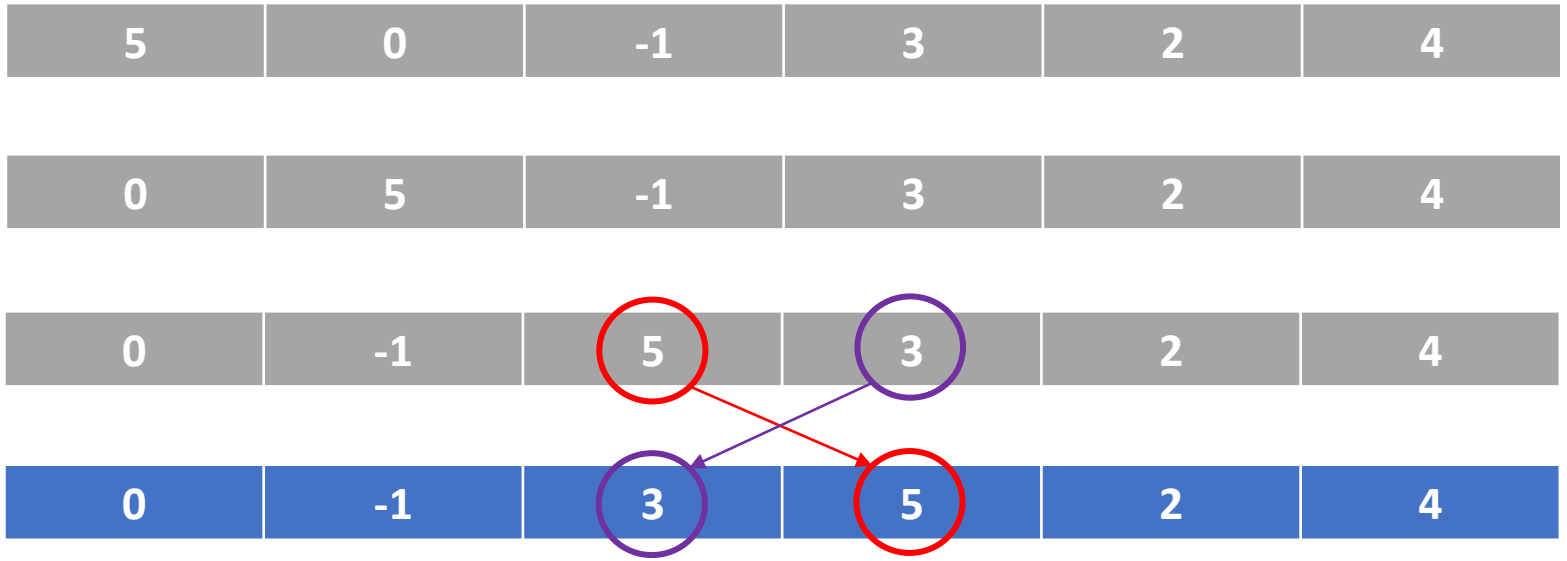
Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.



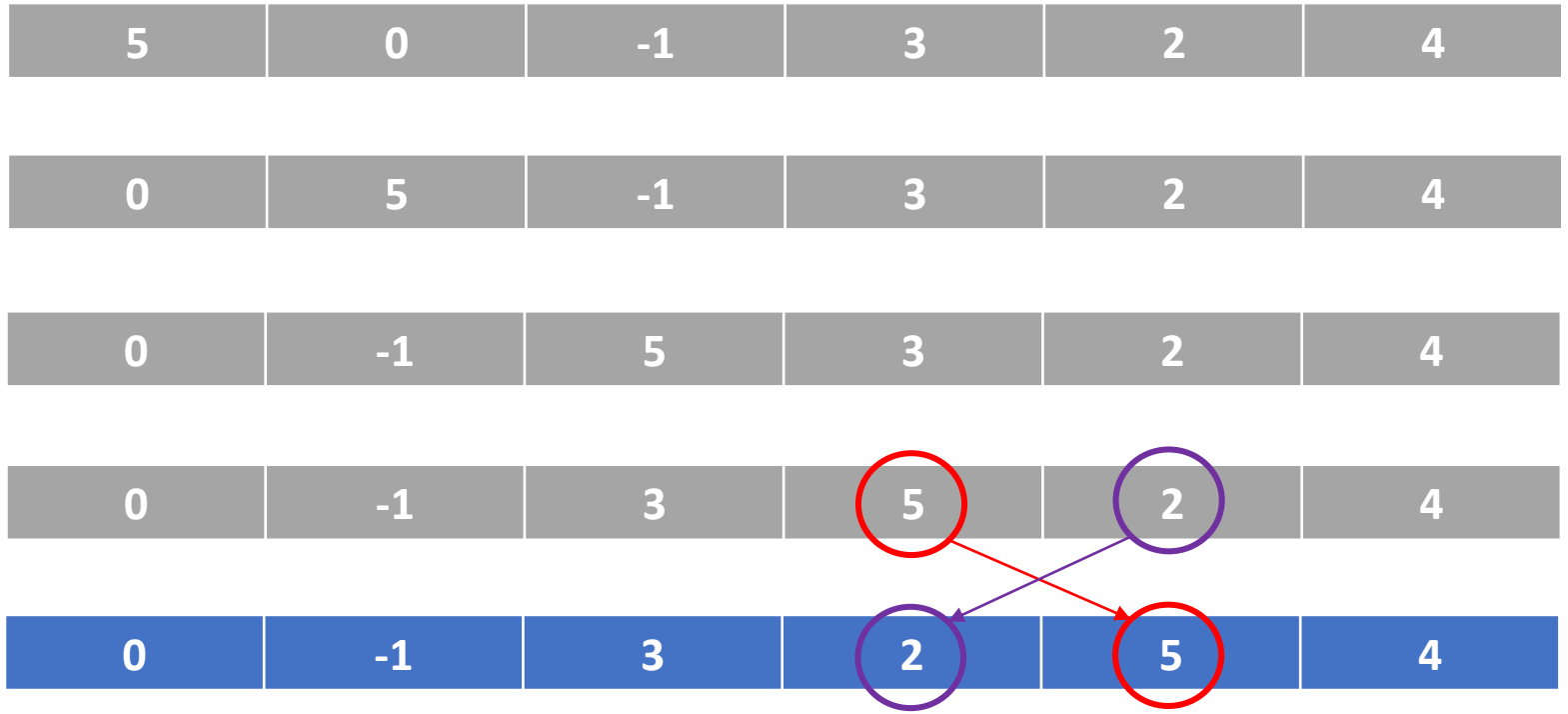
Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.



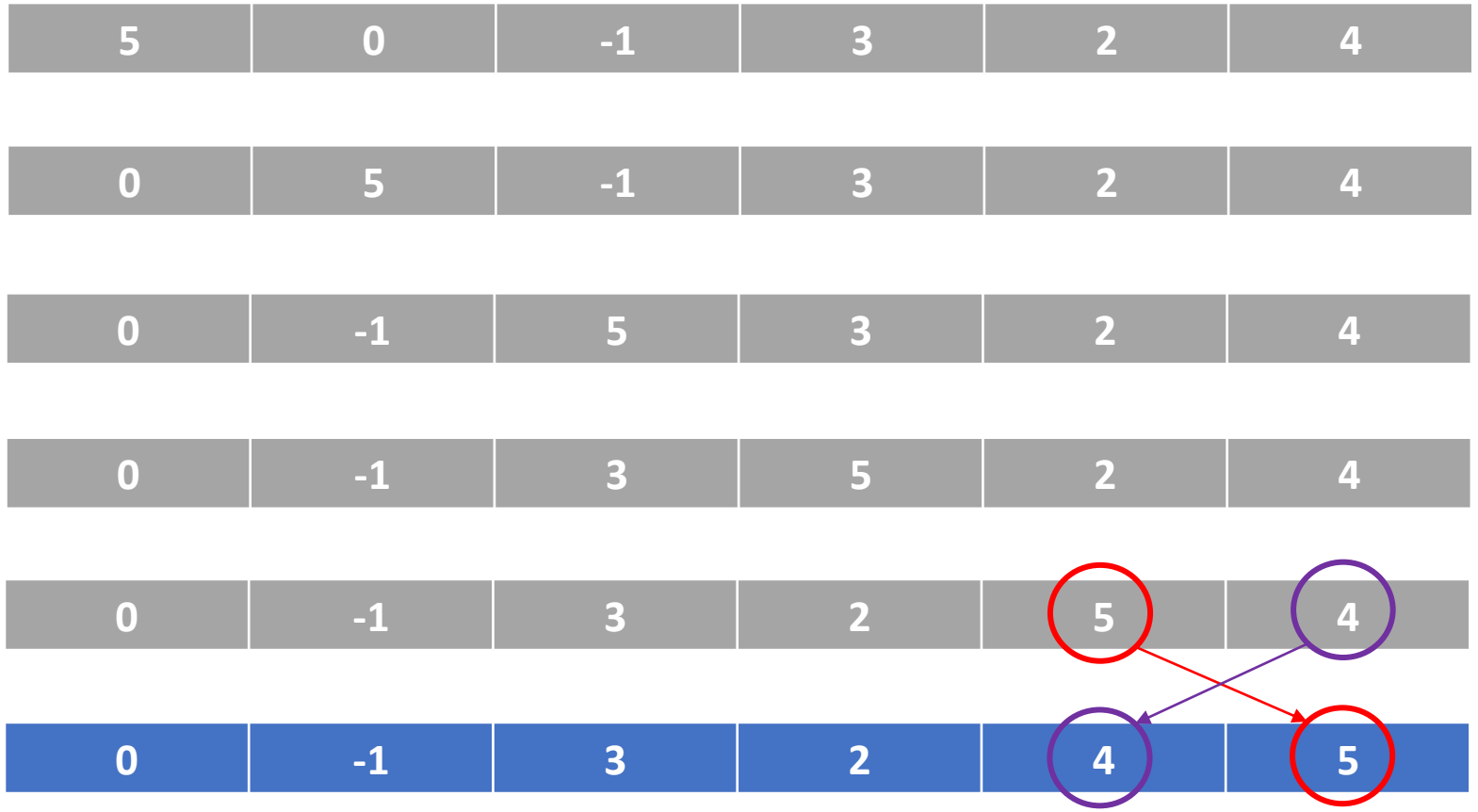
Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.



Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.



Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.

5	0	-1	3	2	4

0	5	-1	3	2	4

0	-1	5	3	2	4

0	-1	3	5	2	4

0	-1	3	2	5	4

0	-1	3	2	4	5

o maior valor ficou na última posição

Trocas

Exemplo: Vamos percorrer o array comparando os pares consecutivos e, se necessário, trocando os valores de posição.

5	0	-1	3	2	4

0	5	-1	3	2	4

0	-1	5	3	2	4

0	-1	3	5	2	4

0	-1	3	2	5	4

0	-1	3	2	4	5

Os elementos ainda não estão ordenados.
É necessário repetir o procedimento mais algumas vezes.

Bubble sort

- Consiste em percorrer o vetor, comparando cada par de elementos consecutivos.
- Se o par estiver fora de ordem, suas posições são trocadas.
- Este procedimento é realizado ***n-1*** vezes (onde ***n*** é o número de elementos no vetor).
  - Porém: ver desafio.

Bubble sort (comentários)

```
void bubble_sort(int v[], int n) {

  // repetir (n – 1) vezes

  // percorrer cada posição do vetor

    // se elemento atual for maior que o próximo

       // trocar os elementos entre as 2 posições



}
```

Bubble sort (código)

```
void bubble_sort(int v[], int n) {
  int i, contador;
  // repetir (n – 1) vezes
  for (contador = 1; contador <= n - 1; contador++) {
  // percorrer cada posição do vetor
    for (i = 0; i < n - 1; i++) {
    // se elemento atual for maior que o próximo
      if (v[i] > v[i + 1]) {
        // trocar os elementos entre as 2 posições
        trocar(v, i, i + 1);
      }
    }
  }
}
```

Bubble sort (código)

```
void bubble_sort(int v[], int n) {
  int i, contador;
  // repetir (n – 1) vezes
  for (contador = 1; contador <= n - 1; contador++) {
  // percorrer cada posição do vetor
    for (i = 0; i < n - 1; i++) {
    // se elemento atual for maior que o próximo
      if (v[i] > v[i + 1]) {
        // trocar os elementos entre as 2 posições
        trocar(v, i, i + 1);
      }
    }
  }
}
```

i não assume o valor do último índice, pois a última posição não tem um próximo elemento.

Bubble sort com sentinela

É o algoritmo já apresentado com as seguintes particularidades:

- Cada vez que o vetor é percorrido para fazer as comparações e trocas, uma variável (a “sentinela”) é utilizada para saber se alguma troca foi realizada.
- Se nenhuma troca foi realizada, o vetor já está ordenado e não é mais necessário continuar a execução do algoritmo.

Bubble sort com sentinela

```
void bubble_sort_sentinela(int v[], int n) {
  int i, contador;
  for (contador = 1; contador <= n - 1; contador++) {
    bool sentinela = false;
    for (i = 0; i < n - 1; i++) {
      if (v[i] > v[i + 1]) {
        trocar(v, i, i + 1);
        sentinela = true;
      }
    }
    if (!sentinela)
      return;
  }
}
```

Bubble sort com sentinela

```
void bubble_sort_sentinela(int v[], int n) {
  int i, contador;
  for (contador = 1; contador <= n - 1; contador++) {
    bool sentinela = false;
    for (i = 0; i < n - 1; i++) {
      if (v[i] > v[i + 1]) {
        trocar(v, i, i + 1);
        sentinela = true;
      }
    }
    if (!sentinela)
      return;
  }
}
```

Bubble sort

Desafio:

Pense na pergunta a seguir e elabore a sua própria resposta sem consultar referências ou ChatGPT e similares!

Como implementar o método de ordenação bolha (bubble sort) de forma que um array seja ordenado com menos comparações do que os algoritmos apresentados nos slides anteriores?

Selection sort (analogia)

As cartas estão espalhadas com a face para cima.

Você pega uma carta por vez; sempre a de menor valor.

Fonte: https://en.wikipedia.org/wiki/Patience_%28game%29#/media/File:Carpet_patience_1.jpg

Selection sort

- Percorremos o vetor, garantindo que na posição atual colocamos o menor valor que está no vetor a partir da posição atual.
- Para encontrar o menor valor, comparamos o valor da posição atual com cada um dos próximos valores.

Selection sort (comentários)

```
void selection_sort(int v[], int n) {
  // para cada posição do vetor (1º percurso)

    // salvar o índice com menor valor ("pegar menor carta"), que começa pelo
    // índice atual de cada iteração do 1º percurso

    // percorrer cada uma das próximas posições do vetor (2º percurso)

      // se o elemento deste segundo percurso for menor que o mínimo, atualizar o
      // índice que contém o menor valor.


    // trocar os elementos entre a posição do 1º percurso e o índice mínimo

}
```

Selection sort (código)

```
void selection_sort(int v[], int n) {
  // para cada posição do vetor (1º percurso)
  for (int i = 0; i < n - 1; i++) {
    // salvar o índice com menor valor ("pegar menor carta"), que começa pelo
    // índice atual de cada iteração do 1º percurso
    int min = i;
    // percorrer cada uma das próximas posições do vetor (2º percurso)
    for (int j = i + 1; j < n; j++) {
      // se o elemento deste segundo percurso for menor que o mínimo, atualizar o
      // índice que contém o menor valor.
      if (v[j] < v[min]) min = j;
    }
    // trocar os elementos entre a posição do 1º percurso e o índice mínimo
    if (min != i) trocar(v, i, min);
  }
}
```

Selection sort (código)

```
void selection_sort(int v[], int n) {
  // para cada posição do vetor (1º percurso)
  for (int i = 0; i < n - 1; i++) {
    // salvar o índice com menor valor ("pegar menor carta"), que começa pelo
    // índice atual de cada iteração do 1º percurso
    int min = i;
    // percorrer cada uma das próximas posições do vetor (2º percurso)
    for (int j = i + 1; j < n; j++) {
      // se o elemento deste segundo percurso for menor que o mínimo, atualizar o
      // índice que contém o menor valor.
      if (v[j] < v[min]) min = j;
    }
    // trocar os elementos entre a posição do 1º percurso e o índice mínimo
    if (min != i) trocar(v, i, min);
  }
}
```

i não assume o valor do último índice, pois a última posição não tem próximos elementos.

Selection sort (código)

```
void selection_sort(int v[], int n) {
  // para cada posição do vetor (1º percurso)
  for (int i = 0; i < n - 1; i++) {
    // salvar o índice com menor valor ("pegar menor carta"), que começa pelo
    // índice atual de cada iteração do 1º percurso
    int min = i;
    // percorrer cada uma das próximas posiç
    for (int j = i + 1; j < n; j++) {
      // se o elemento deste segundo percurso for menor que o mínimo, atualizar o
      // índice que contém o menor valor.
      if (v[j] < v[min]) min = j;
    }
    // trocar os elementos entre a posição do 1º percurso e o índice mínimo
    if (min != i) trocar(v, i, min);
  }
}
```

j recebe valor inicial de (i+1) porque queremos percorrer os próximos elementos.

Insertion sort (analogia)

As cartas estão em uma pilha viradas com a face para baixo.

Você retira uma carta por vez e já organiza as cartas na sua mão em ordem crescente.

Fonte: https://pt.wikipedia.org/wiki/Ficheiro:Playing_cards_modified.jpg

Insertion sort

- No início há apenas o primeiro elemento no grupo ordenado.
- Percorremos o vetor a partir da segunda posição e cada novo elemento é inserido no grupo que já está ordenado, na posição correta.
- Para encontrar esta posição correta, percorremos o grupo ordenado no sentido inverso (do último para o primeiro) até encontrar um elemento menor. Os elementos maiores devem ser deslocados para a direita.

Insertion sort (comentários)

```
void insertion_sort(int v[], int n) {
  // para cada posição do vetor (1º percurso)
  // observe que começamos pelo 2º elemento, pois consideramos que o 1º elemento já está ordenado

    // guardar elemento



    // percorrer no sentido inverso enquanto não chegar ao início do vetor e não encontrar um menor

      // se o elemento do 2º percurso for maior que o elemento do 1º percurso

        // deslocar elemento do 2º percurso para a direita

        // continuar procurando


      // se o elemento do 2º percurso for menor, encontramos a posição onde o elemento do 1º percurso
      // deve ser inserido


    // inserir elemento na posição encontrada


}
```

Insertion sort (código)

```
void insertion_sort(int v[], int n) {
  // para cada posição do vetor (1º percurso)
  // observe que começamos pelo 2º elemento, pois consideramos que o 1º elemento já está ordenado
  for (int i = 1; i < n; i++) {
    // guardar elemento
    int elemento = v[i];
    int j = i - 1;
    bool encontrou = false;
    // percorrer no sentido inverso enquanto não chegar ao início do vetor e não encontrar um menor
    while (j >= 0 && !encontrou) {
      // se o elemento do 2º percurso for maior que o elemento do 1º percurso
      if (v[j] > elemento) {
        // deslocar elemento do 2º percurso para a direita
        v[j + 1] = v[j];
        // continuar procurando
        j--;
      }
      // se o elemento do 2º percurso for menor, encontramos a posição onde o elemento do 1º percurso
      // deve ser inserido
      else encontrou = true;
    }
    // inserir elemento na posição encontrada
    v[j + 1] = elemento;
  }

}
```

Universidade Presbiteriana
Mackenzie

Faculdade de
Computação e Informática