Busca e Ordenação

Algoritmos de Ordenação Quadráticos

Prof. Edson Alves - UnB/FGA

2020

Sumário

Fundamentos de Ordenação

Ordenação parcial e ordenação total

- Seja $a = \{a_1, a_2, \ldots, a_N\}$ uma sequência de N elementos
- Seja $R \subset a \times a$ uma relação
- Dados dois elementos $a_i, a_j \in a$, a_i se relaciona com a_j se $(a_i, a_j) \in R$
- $(a_i, a_j) \in R$ não implica, necessariamente, $(a_j, a_i) \in R$
- Seja $S = \{a_k \in a \mid \exists b \in a : (a_k, b) \in R \vee (b, a_k) \in R\}$
- Dizemos que R é uma relação de ordem parcial se, para todos $x, y, z \in S$, temos que
  1. $(x, x) \in R$
  2. se $(x, y) \in R$ e $(y, x) \in R$ então x e y são iguais
  3. se $(x, y) \in R$ e $(y, z) \in R$ então $(x, z) \in R$
- Se para todos $x, y \in a$ vale $(x, y) \in R$ ou $(y, x) \in R$, então R é uma relação de ordem total

Definição de ordenação

- Dizemos que uma sequência a está ordenada de acordo com a relação de ordem R se, para todos $i = 2, 3, \ldots, N$, temos que $(a_{i-1}, a_i) \in R$
- Uma algoritmo de ordenação $A(a, R)$ recebe, como entrada, uma sequência a e uma relação de ordem R e, ao final do algoritmo, a sequência a está ordenada de acordo com a relação R
- Na prática, a relação R é implementada como uma função binária f tal que $f(x, y)$ retorna verdadeiro se $(x, y) \in R$
- Como a definição de ordenação depende da relação R, uma mesma sequência pode estar ordenada de acordo com R_1 e não ordenada de acordo com R_2

Características dos algoritmos de ordenação

- Se a sequência a ser ordenada pode ser armazenada inteiramente em memória, o algoritmo é dito interno; caso contrário, é chamado externo
- Se o algoritmo usa apenas a memória da própria sequência (e talvez uma pequena quantidade adicional para variáveis temporárias), o algoritmo é denominado *in-place*
- Se o algoritmo demanda uma cópia extra da sequência, é chamado *not-in-place* ou *out-of-place*
- Um algoritmo de ordenação é estável se ele preserva a ordem relativa de elementos iguais

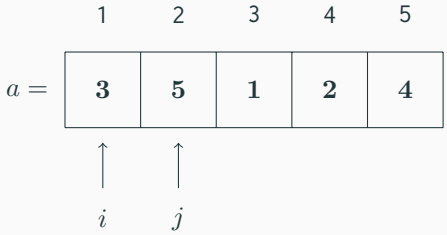
Selection Sort

Selection Sort

- *Selection sort* é um algoritmo de ordenação de simples entendimento e codificação
- Primeiramente ele identifica o menor dentre todos os elementos da sequência e o armazena na primeira posição
- Em seguida, procura o menor elemento dentre os que restaram, e o move para segunda posição
- Em faz o mesmo para a terceira, quarta, até a última posição
- A complexidade assintótica é $O(N^2)$, onde N é o número de elementos da sequência

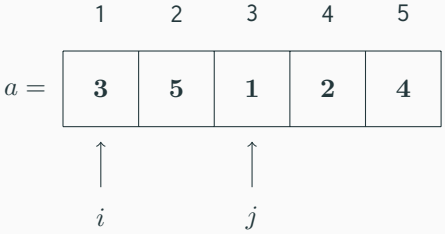
Visualização do *selection sort*

$k = 1$



Visualização do *selection sort*

$k = 3$

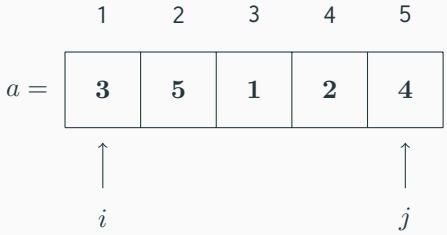


Visualização do *selection sort*

$k = 3$

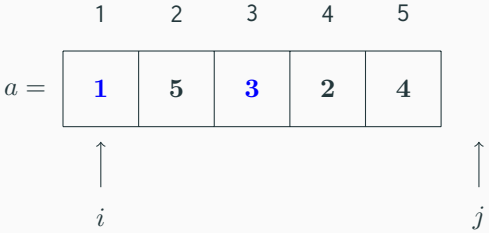
Visualização do *selection sort*

$k = 3$



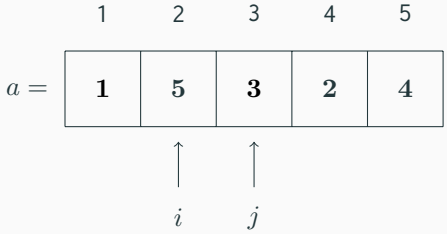
Visualização do *selection sort*

$k = 3$



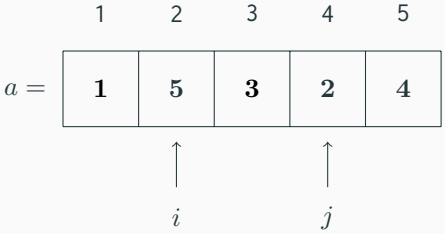
Visualização do *selection sort*

$k = 3$



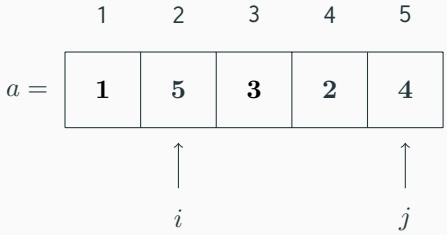
Visualização do *selection sort*

$k = 4$



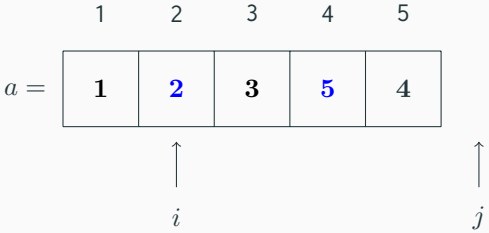
Visualização do *selection sort*

$k = 4$



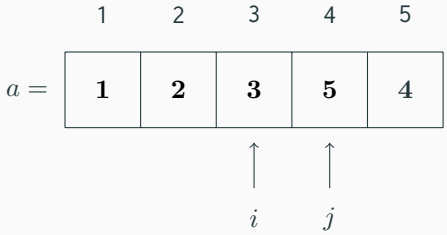
Visualização do *selection sort*

$k = 4$



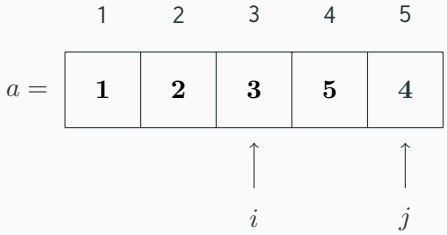
Visualização do *selection sort*

$k = 3$



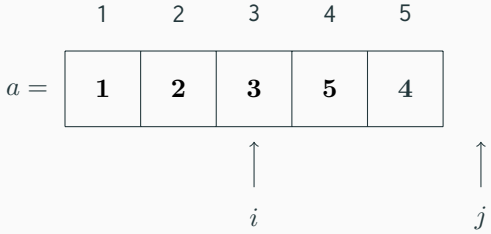
Visualização do *selection sort*

$k = 3$



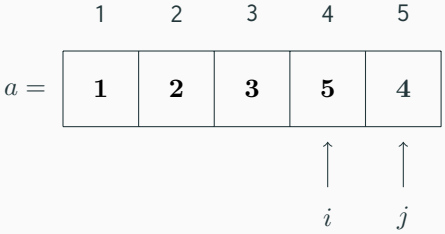
Visualização do *selection sort*

$k = 3$



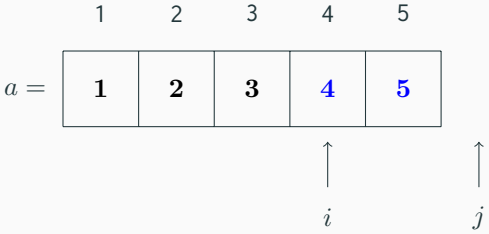
Visualização do *selection sort*

$k = 5$



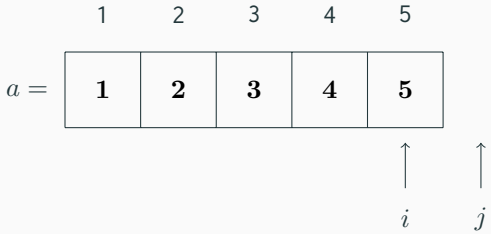
Visualização do *selection sort*

$k = 5$



Visualização do *selection sort*

$k = 5$



Implementação do *selection sort* em C++

```cpp
#include <bits/stdc++.h>

using namespace std;

template<typename T>
void selection_sort(vector<T>& as)
{
    int N = as.size();

    for (int i = 0; i < N; ++i)
    {
        int k = i;                      // k = índice do menor elemento

        for (int j = i + 1; j < N; ++j)
            if (as[j] < as[k])
                k = j;

        swap(as[i], as[k]);
    }
}
```

Implementação do *selection sort* em C++

```
22 int main()
23 {
24     vector<int> as { 3, 5, 1, 2, 4 };
25
26     selection_sort(as);
27
28     for (size_t i = 0; i < as.size(); ++i)
29         cout << as[i] << (i + 1 == as.size() ? '\n' : ' ');
30
31     return 0;
32 }
```

Observações sobre o *selection sort*

- O pior caso do algoritmo acontece quando a sequência está ordenada em sentido contrário, isto é, $\forall i = 2, 3, \ldots, N, (a_i, a_{i-1}) \in R$
- No pior caso, são feitas 2 atribuições no início, $6N$ atribuições no laço externo (considerando 3 atribuições por `swap()`) e

$$\sum_{i=0}^{N-1} 2(N - i - 1) = 2 \sum_{k=0}^{N-1} k = N(N-1)$$

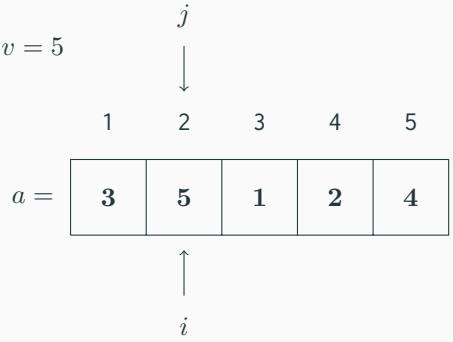
  atribuições no laço interno
- Assim, $f(N) = 2 + 6N + N(N-1)$ é $O(N^2)$
- O *selection sort* é um algoritmo instável *in-place*

Insertion Sort

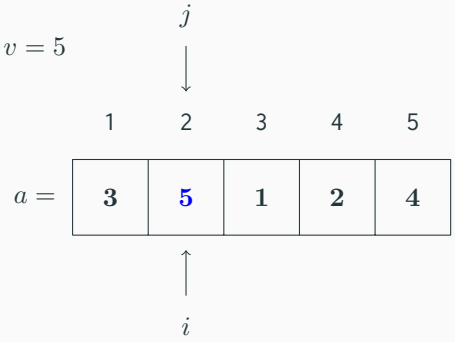
Insertion Sort

- *Insertion sort* é um algoritmo de ordenação similar ao *selection sort*
- Ele é estável, *in-place* e tem complexidade $O(N^2)$
- Ele considera, inicialmente, que um sequência com um único elemento já está ordenada
- Em seguida, para cada elemento da sequência, ele procura a posição correta no vetor ordenado que está à esquerda do elemento, e o insere nesta posição
- É o tipo de ordenação que os jogadores de cartas costumam usar

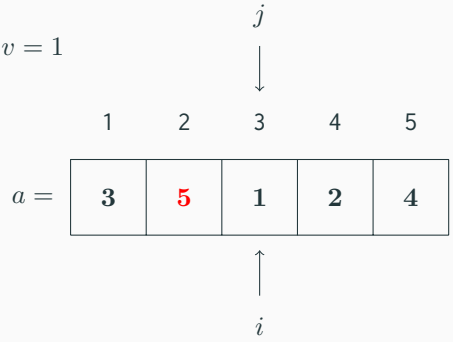
Visualização do *insert sort*



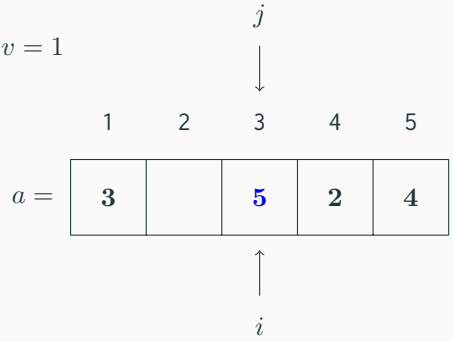
Visualização do *insert sort*



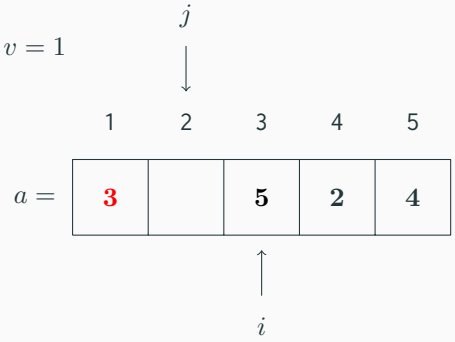
Visualização do *insert sort*



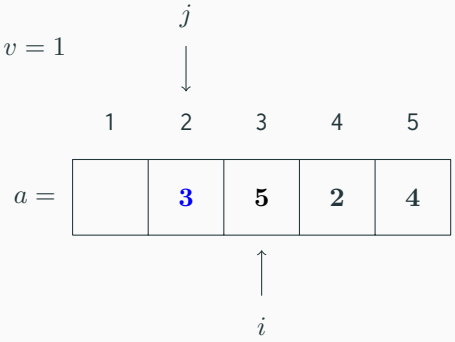
Visualização do *insert sort*



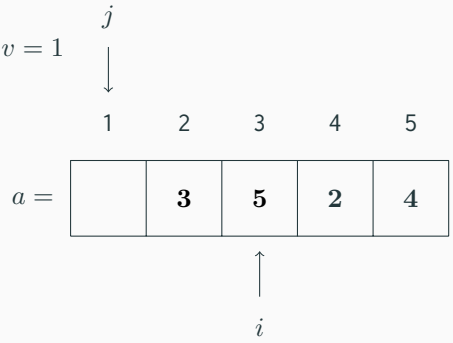
Visualização do *insert sort*



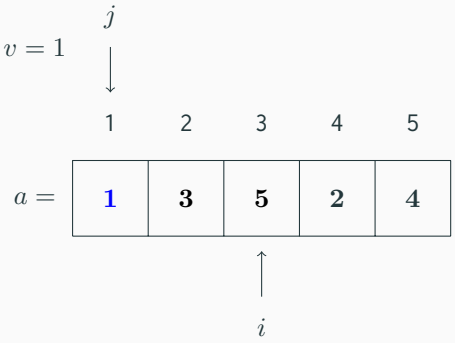
Visualização do *insert sort*



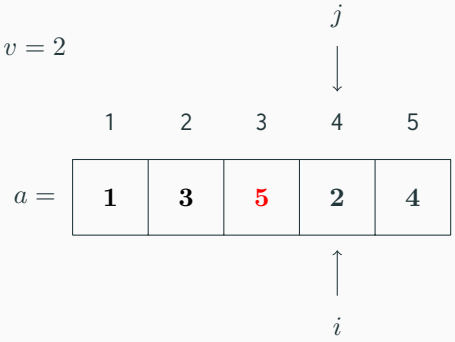
Visualização do *insert sort*



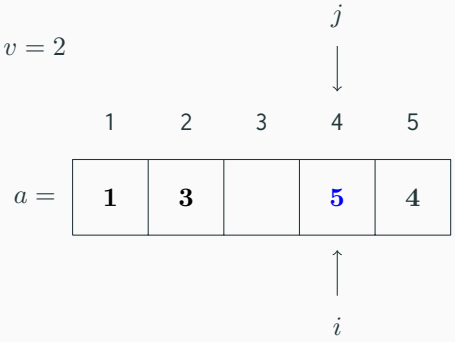
Visualização do *insert sort*



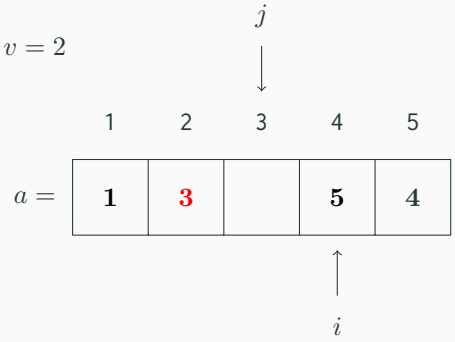
Visualização do *insert sort*



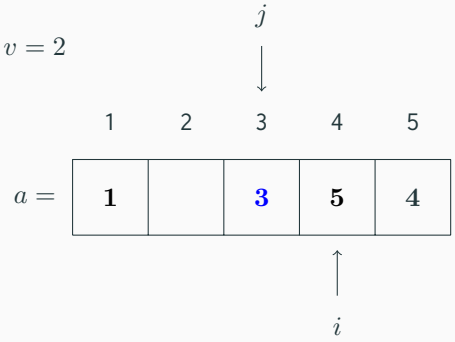
Visualização do *insert sort*



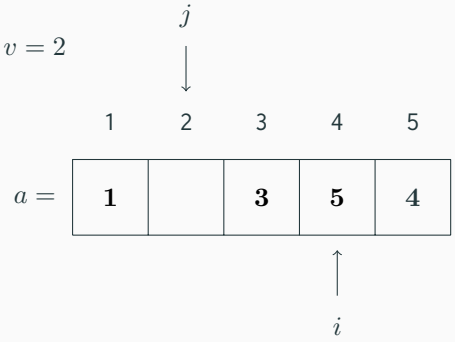
Visualização do *insert sort*



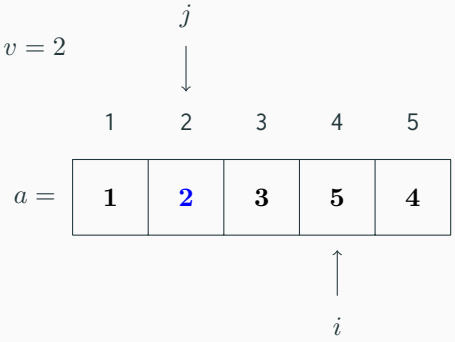
Visualização do *insert sort*



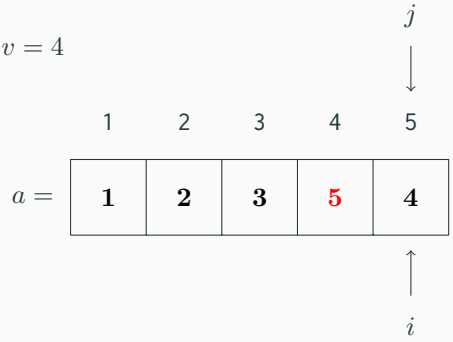
Visualização do *insert sort*



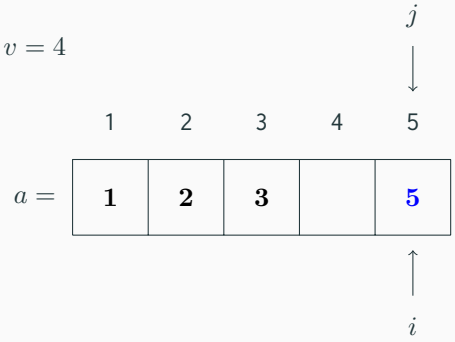
Visualização do *insert sort*



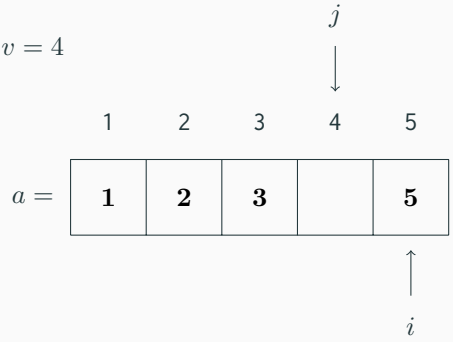
Visualização do *insert sort*



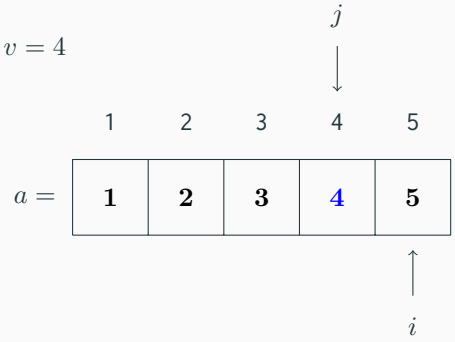
Visualização do *insert sort*



Visualização do *insert sort*



Visualização do *insert sort*



Implementação do *insert sort* em C++

```cpp
#include <bits/stdc++.h>

using namespace std;

template<typename T>
void insert_sort(vector<T>& as)
{
    int N = as.size();

    for (int i = 1, j; i < N; ++i)
    {
        auto v = as[i];

        for (j = i; j and as[j - 1] > v; --j)
            as[j] = as[j - 1];

        as[j] = v;
    }
}
```

Implementação do *insert sort* em C++

```
21 int main()
22 {
23     vector<int> as { 3, 5, 1, 2, 4 };
24
25     insert_sort(as);
26
27     for (size_t i = 0; i < as.size(); ++i)
28         cout << as[i] << (i + 1 == as.size() ? '\n' : ' ');
29
30     return 0;
31 }
```

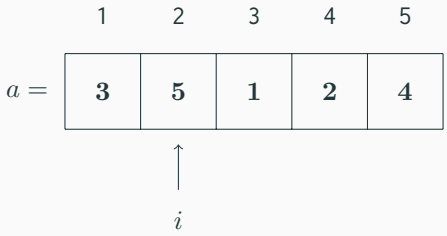
Bubble Sort

Bubble Sort

- *Bubble sort* é um algoritmo de ordenação estável, *in-place* e quadrático
- É um algoritmo popular em cursos introdutórios de algoritmos
- Ele itera até N vezes sobre os elementos
- Em cada iteração, se ele encontrar um par de elementos adjacentes que estão fora de ordem, ele inverte a posição de ambos
- Se uma dada iteração não fizer nenhuma troca, o algoritmo é finalizado
- Como, a cada iteração, ao menos o maior elemento fora de posição será posicionado corretamente, o algoritmo sempre termina com o vetor ordenado

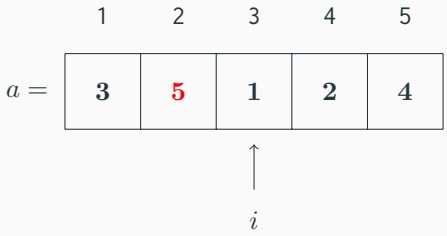
Visualização do *bubble sort*

updated $=$ `false`



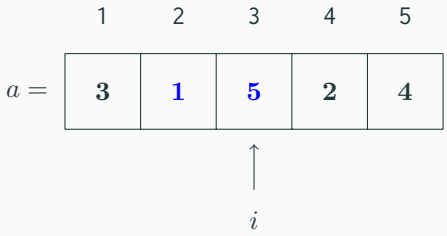
Visualização do *bubble sort*

updated $=$ `false`



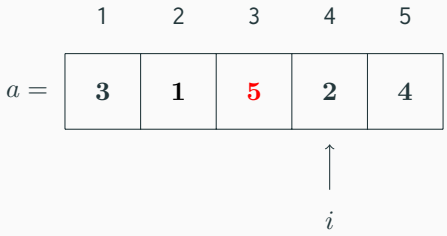
Visualização do *bubble sort*

updated $=$ **true**



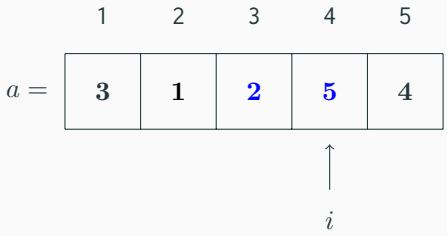
Visualização do *bubble sort*

updated $=$ **true**



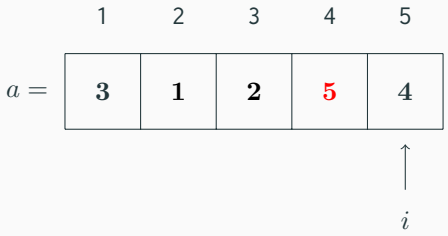
Visualização do *bubble sort*

updated $=$ **`true`**



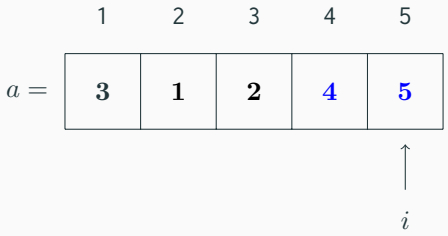
Visualização do *bubble sort*

updated $=$ **true**



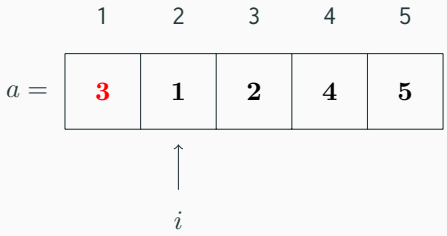
Visualização do *bubble sort*

updated $=$ **`true`**



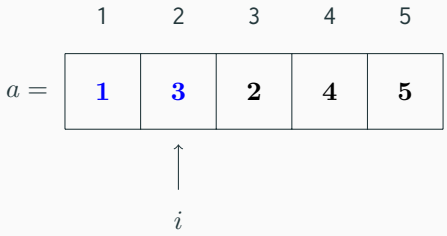
Visualização do *bubble sort*

updated $=$ **`false`**



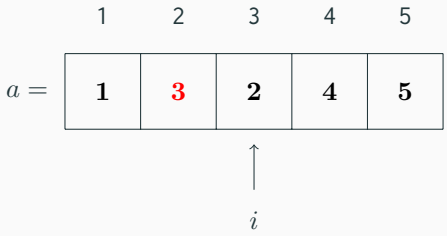
Visualização do *bubble sort*

updated $=$ **true**



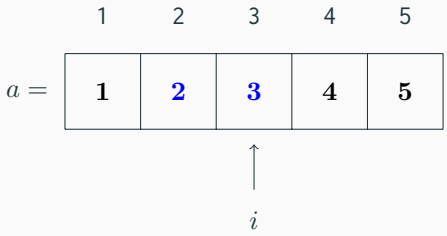
Visualização do *bubble sort*

updated $=$ **true**



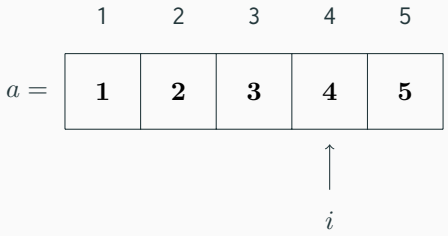
Visualização do *bubble sort*

updated $=$ **true**



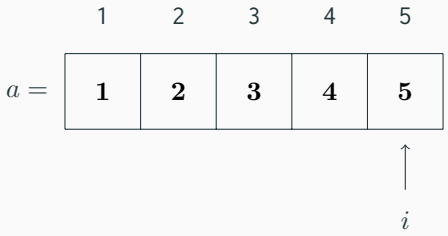
Visualização do *bubble sort*

updated $=$ **true**



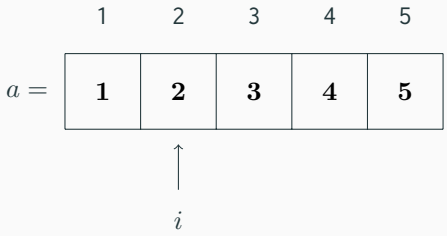
Visualização do *bubble sort*

updated $=$ **true**



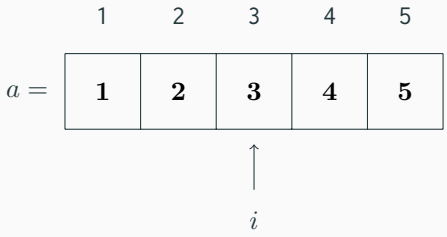
Visualização do *bubble sort*

updated $=$ **`false`**



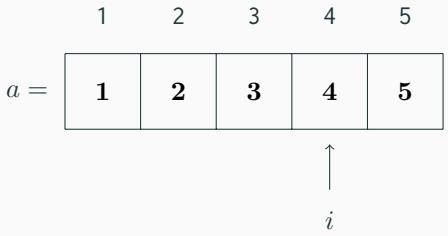
Visualização do *bubble sort*

updated $=$ **`false`**



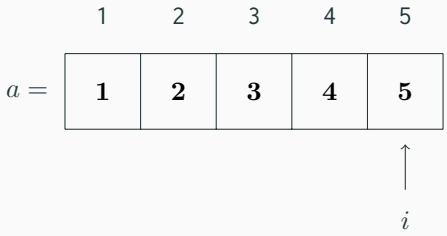
Visualização do *bubble sort*

updated $=$ **`false`**



Visualização do *bubble sort*

updated $=$ **`false`**



Implementação do *bubble sort* em C++

```cpp
#include <bits/stdc++.h>

using namespace std;

template<typename T>
void bubble_sort(vector<T>& as) {
    int N = as.size();
    bool updated;

    do {
        updated = false;

        for (int i = 1; i < N; ++i) {
            if (as[i - 1] > as[i]) {
                updated = true;
                swap(as[i - 1], as[i]);
            }
        }
    } while (updated);
}
```

Implementação do *bubble sort* em C++

```
22 int main()
23 {
24     vector<int> as { 3, 5, 1, 2, 4 };
25
26     bubble_sort(as);
27
28     for (size_t i = 0; i < as.size(); ++i)
29         cout << as[i] << (i + 1 == as.size() ? '\n' : ' ');
30
31     return 0;
32 }
```

Referências

1. **DROZDEK**, Adam. *Algoritmos e Estruturas de Dados em C++*, 2002.
2. **KERNIGHAN**, Bryan; **RITCHIE**, Dennis. *The C Programming Language*, 1978.
3. **STROUSTROUP**, Bjarne. *The C++ Programming Language*, 2013.
4. **SEDGEWICK**, Robert. *Algorithms*, 4th edition, 2011.
5. Wikipédia. In-place algorithm, acesso em 01/10/2018.
6. Wikipédia. Partially Ordered Set, acesso em 01/10/2018.