Técnicas e Análise de Algoritmos

Busca e Ordenação - Parte 02

Professor: **Jeremias Moreira Gomes**
E-mail: jeremias.gomes@idp.edu.br

Introdução

Ordenação Parcial e Ordenação Total

- Seja uma sequência $a = \{a_1, a_2, \ldots, a_N\}$ de N elementos
- Seja $R \subset a \times a$
- 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 que $(a_j, a_i) \in R$

Ordenação Parcial e Ordenação Total

- Seja $S = \{a_k \in a | \exists b \in a : (a_k, b) \in R \vee (b, a_k) \in R\}$
- Dizemos que R é uma relação de ordem parcial se, para todos $x, y, z \in S$, temos que:
  - $(x, x) \in R$
  - se $(x, y) \in R$ e $(y, x) \in R$ então x e y são iguais
  - se $(x, y) \in R$ e $(y, z) \in R$ então x e $z \in R$

Ordenação Parcial e Ordenação Total

- Um exemplo simples de ordem parcial, é considerar a relação R, onde R é o tamanho de strings:
  - É possível dizer que “d” < “idp”
  - Ou que “id” ≤ “idp”
  - Porém, não se pode assumir algo sobre “id” e “ip”
    - Dada a relação, estes não são comparáveis

Ordenação Parcial e Ordenação Total

- 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
- 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 $(a_{i-1}, a_i) \in R$

Ordenação Parcial e Ordenação Total

- Um 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 por R1 mas não por R2

Características de um Algoritmo 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, o algoritmo é denominado **in-place**
  - Vai utilizar também algumas variáveis temporárias adicionais

Características de um Algoritmo de Ordenação

- Se o algoritmo usa uma cópia extra da sequência, ele é chamado de not-in-place ou **out-of-place**
- Um algoritmo de ordenação é **estável** se ele preserva a ordem relativa de elementos iguais

Características de um Algoritmo de Ordenação

- Se o algoritmo usa uma cópia extra da sequência, ele é chamado de not-in-place ou **out-of-place**
- Um algoritmo de ordenação é **estável** se ele preserva a ordem relativa de elementos iguais
  - [1, 4, 2, 8, 2, 3, 7, 9]
  - [1, 2, 2, 3, 4, 7, 8, 9] (estável)
  - [1, 2, 2, 3, 4, 7, 8, 9] (instável ou não estável)

Ordenações Quadráticas

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
  - Ótimo para um primeiro raciocínio de ordenação
- 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 destes

Bubble Sort

- 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

Bubble Sort

6 5 3 1 8 7 2 4

Bubble Sort

```
void bubble_sort(vector<int> &v)
{
    bool troca;
    do {
        troca = false;
        for (int i = 1; i < v.size(); i++) {
            if (v[i - 1] > v[i]) {
                swap(v[i - 1], v[i]);
                troca = true;
            }
        }
    } while (troca);
}
```

Selection Sort

Selection Sort

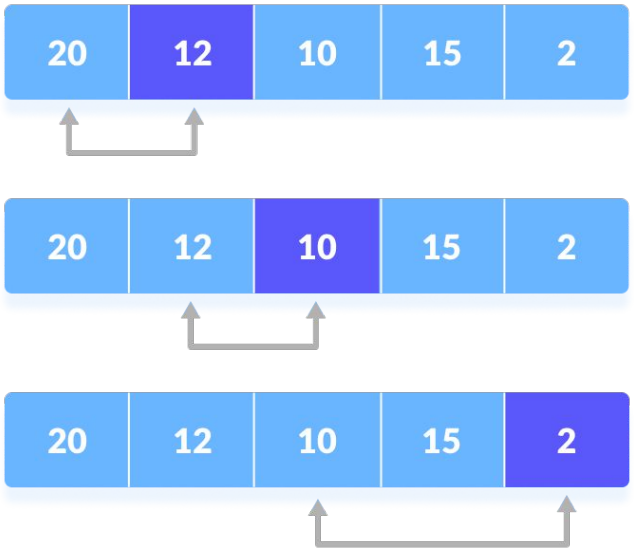
- Selection sort é um algoritmo de ordenação de simples entendimento e codificação
  - **Não é estável**
- Primeiramente ele identifica o menor dentre todos os elementos da sequência e o armazena na primeira posição

Selection Sort

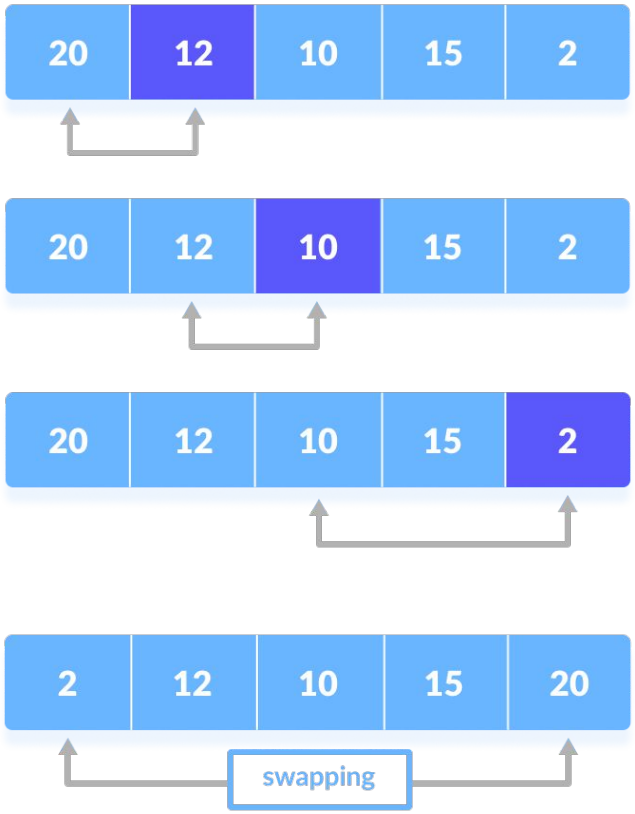
- Em seguida, procura o menor elemento dentre os que restaram, e o move para segunda posição
- Em seguida faz o mesmo para a terceira, quarta, até a última posição
- A complexidade assintótica é $O(N^2)$, onde N é o número de elementos da sequência

Selection Sort

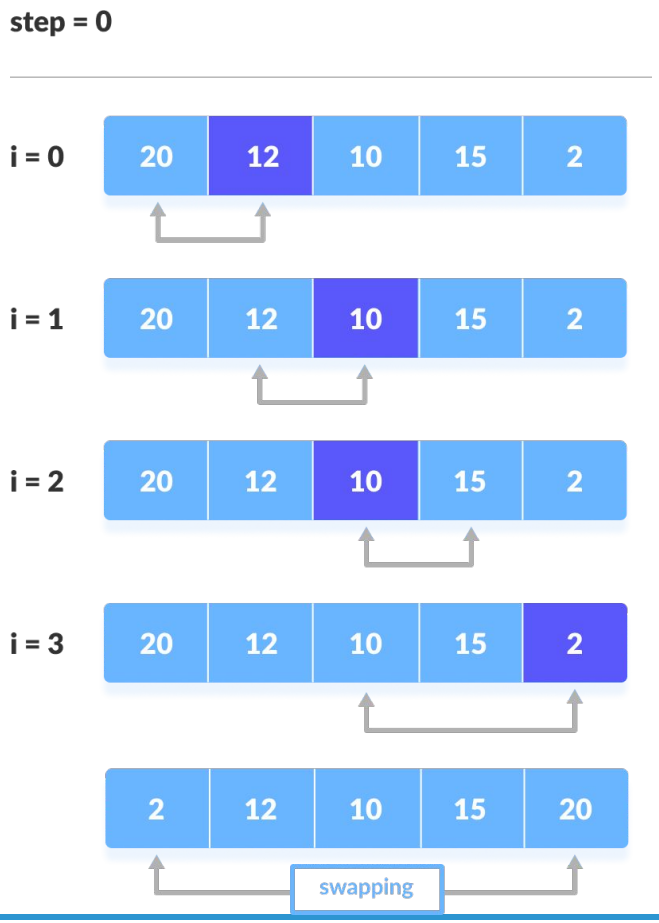
Selection Sort



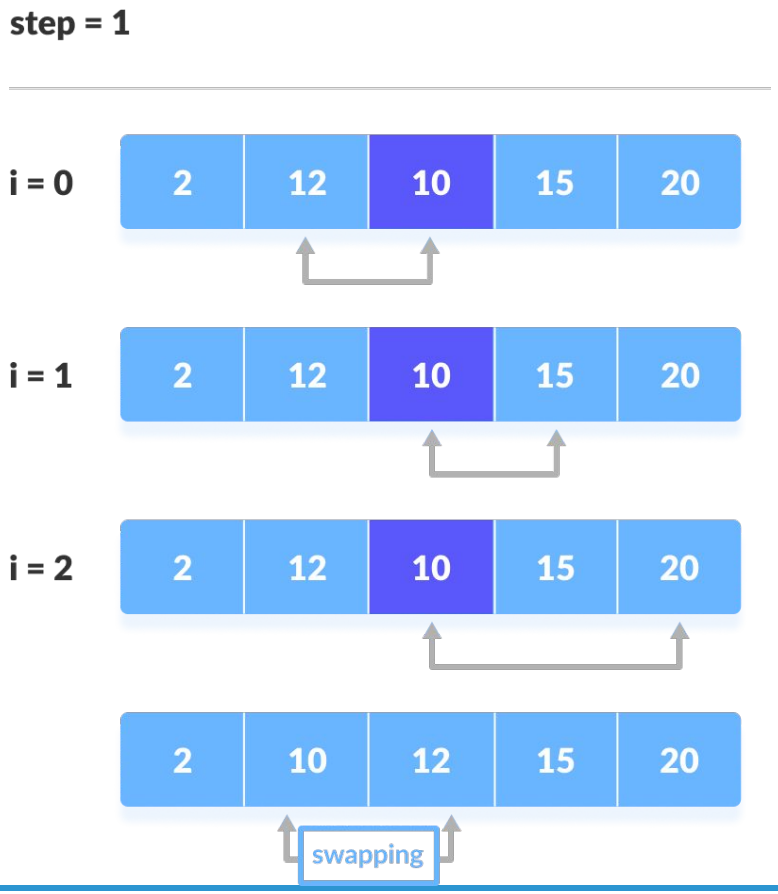
Selection Sort



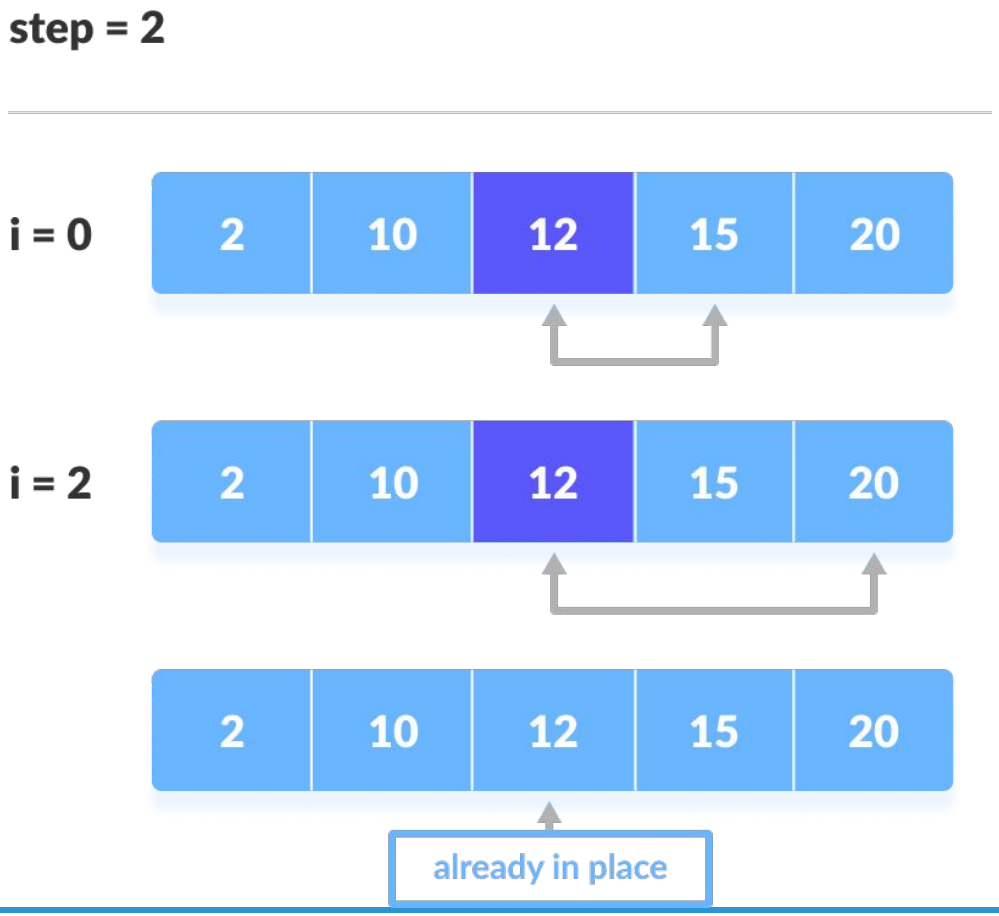
Selection Sort



Selection Sort



Selection Sort



Selection Sort

step = 3

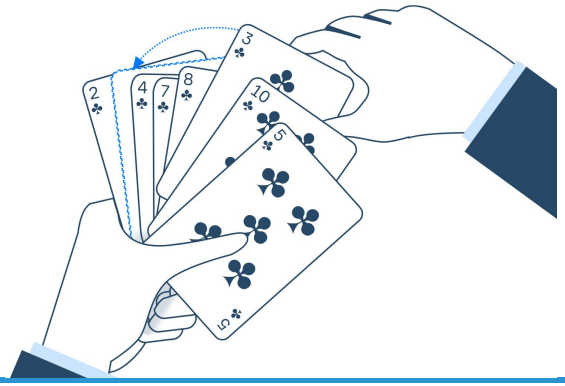
Selection Sort

```
void selection_sort(vector<int> &v)
{
    for (int i = 0; i < v.size() - 1; i++) {
        int menor = i;
        for (int j = i + 1; j < v.size(); j++) {
            if (v[j] < v[menor]) {
                menor = j;
            }
        }
        swap(v[i], v[menor]);
    }
}
```

Insertion Sort

Insertion Sort

- Insertion sort é um algoritmo de ordenação similar ao selection sort
- Ele é **estável**, **in-place** e tem complexidade $O(N^2)$
- É o tipo de ordenação que os jogadores de cartas costumam usar

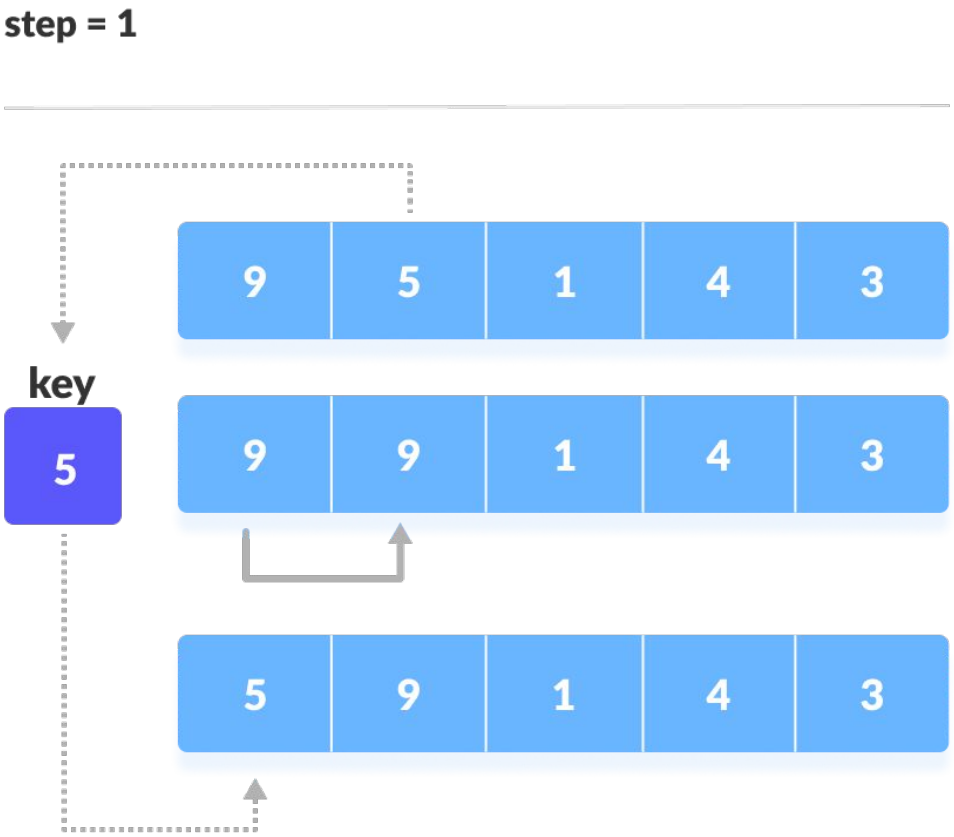


Insertion Sort

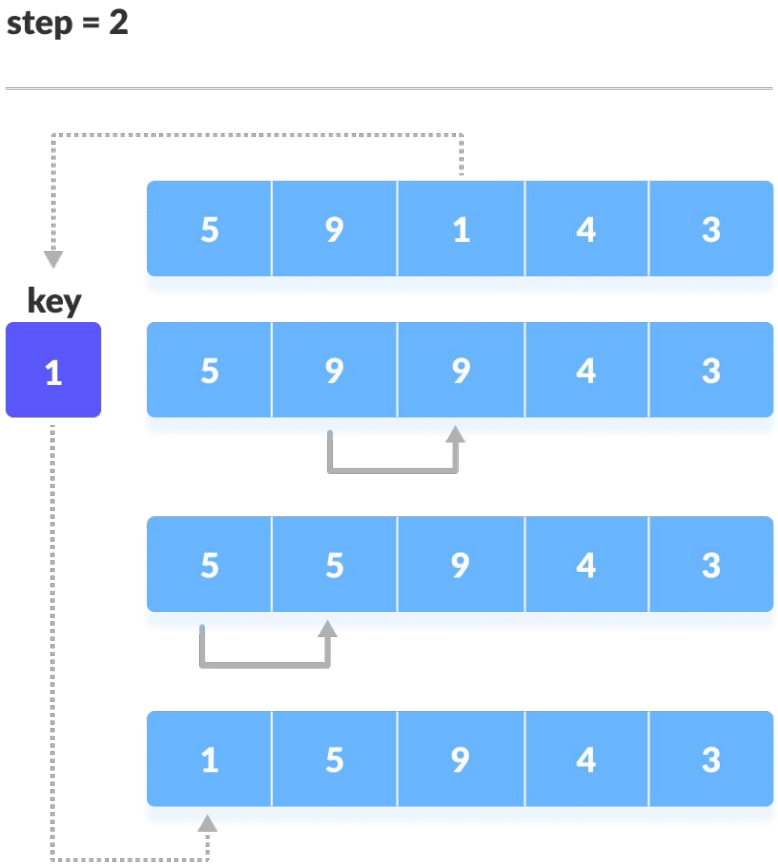
- Ele considera, inicialmente, que um sequência com um único elemento já está ordenada
- Em seguida, para cada elemento da sequência, ele procura a posição correta no vetor ordenado que está à esquerda do elemento, e o insere nesta posição

Insertion Sort

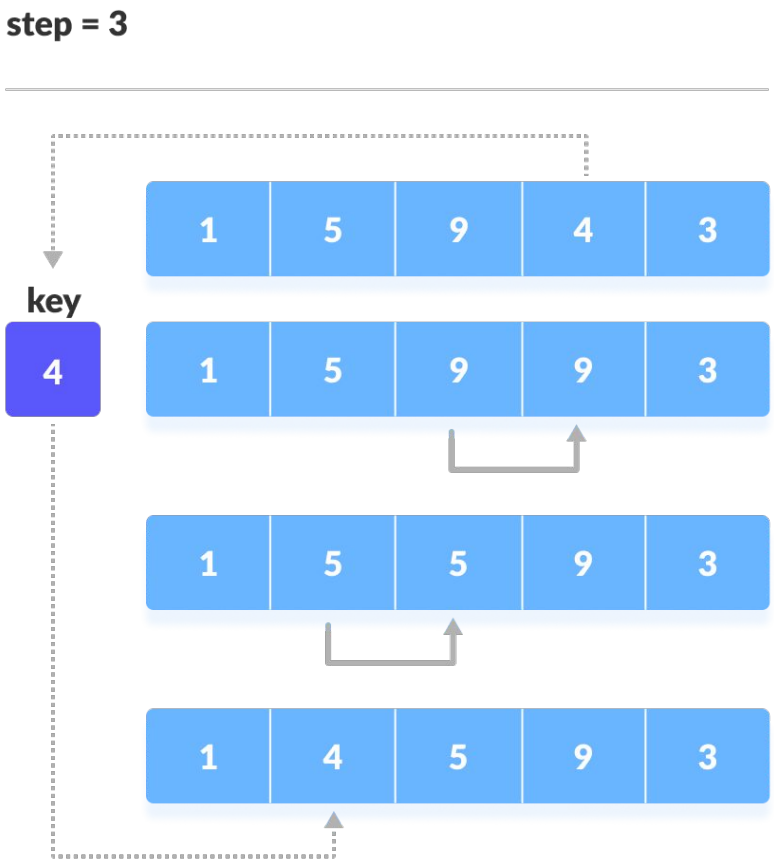
Insertion Sort



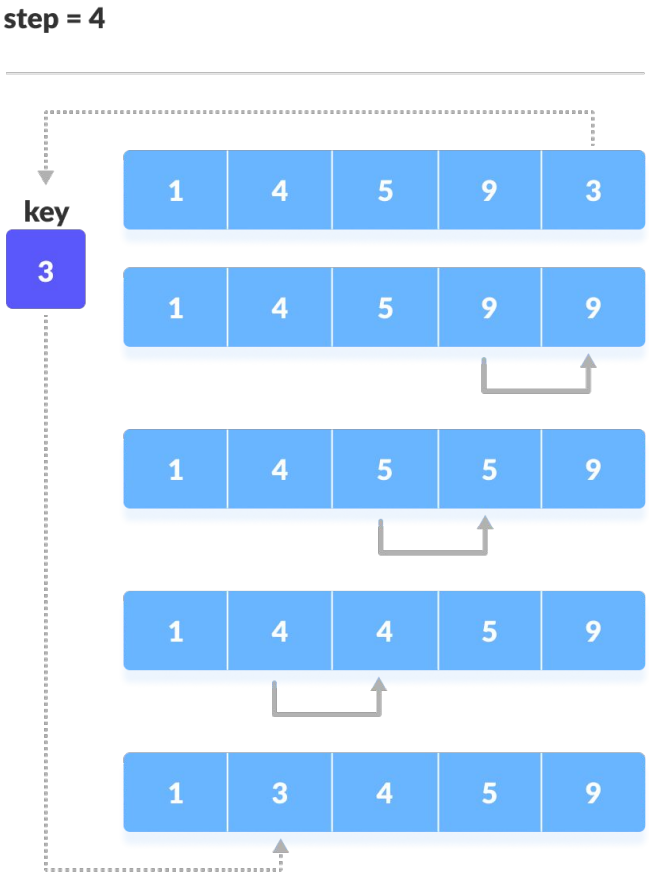
Insertion Sort



Insertion Sort



Insertion Sort



Insertion Sort

```
void insertion_sort(vector<int> &v)
{
    int i, j, tmp;
    for (i = 1; i < v.size(); i++) {
        tmp = v[i];
        for (j = i; j > 0 && v[j - 1] > tmp; j--) {
            v[j] = v[j - 1];
        }
        v[j] = tmp;
    }
}
```

Conclusão