Algoritmos de pesquisa

- Define-se pesquisa como a operação que permite encontrar ou concluir que não existe, um dado elemento num dado conjunto.
- A pesquisa de um elemento pode ser feita num conjunto ordenado ou não.
- Quando o conjunto não está ordenado, o método usado é o exaustivo, que consiste em percorrer sequencialmente todo o conjunto (desde o primeiro) até se encontrar o elemento desejado ou, não o encontrando, se concluir que não existe.
- Quando o conjunto está ordenado, existem vários métodos, como sejam os de pesquisa sequencial e binária.
- No que se segue, considera-se que a ordenação é crescente.

Algoritmos de pesquisa

Pesquisa exaustiva (algoritmo)

***Pesquisar o elemento* Elem *no vector* V *de tamanho* tam**

```
k ← -1      //  significa que Elem não foi encontrado em V
i ←  0
Enquanto (i < tam) e (k = -1) Fazer
    Se (V[i] = Elem) então
        k ← i
    senão
        i ← i + 1
Se (k = -1) então
    Elem não se encontra em V
senão
    Elem encontra-se na posição k
```

Algoritmos de pesquisa

Pesquisa exaustiva (versão 1)

```
int PesquisaExaustiva (int Elem, int V[], int tam)
{
  int i = 0, k = -1;     //  k = posição onde se encontra Elem em V
  while ( (i < tam)  &&  (k == -1) )
     if (Elem == V[i])
        k = i;
     else
        i = i + 1;
   return (k);
}
```

Algoritmos de pesquisa

Pesquisa exaustiva (versão 2)

```
int PesquisaExaustiva (int Elem, int V[], int tam)
{
   int  i = 0;
   while ( (i < tam)  &&  (Elem != V[i]) )
       i = i + 1;
   if (i == tam)
       return (-1);
   else
       return (i);
}
```

Algoritmos de pesquisa

Pesquisa sequencial (algoritmo)

***Pesquisar o elemento* Elem *no vector* V *de tamanho* tam**

```
k ← -1    //  significa que Elem ainda não foi encontrado em V
i ← 0     //  índice dos elementos do vector V
Enquanto (i < tam) e (k = -1) Fazer
    Se (V[i] = Elem) então
        k ← i
    senão
        Se (V[i] < Elem) então
            i ← i + 1
        senão
            k ← -2;    //  significa que Elem não está em V
Se (k ≥ 0) então
    Elem encontra-se na posição k
senão
    Elem não se encontra em V
```

Algoritmos de pesquisa

Pesquisa sequencial (versão 1)

```
int PesquisaSequencial (int Elem, int V[], int tam)
{
  int i = 0, k = -1;     //  k = posição onde se encontra Elem em V
  while ( (i < tam)  &&  (k == -1) )
     if (Elem == V[i])
        k = i;
     else
        if (V[i] < Elem)
           i = i + 1;
        else
           k = -2;
  return (k);
}
```

Algoritmos de pesquisa

Pesquisa sequencial (versão 2)

```
int PesquisaSequencial (int Elem, int V[], int tam)
{
   int  i = 0;
   while ( (i < tam)  &&  (V[i] < Elem) )
      i = i + 1;
   if ( (i < tam)  &&  (Elem == V[i]) )
      return (i);
   else
      return (-1);
}
```

Algoritmos de pesquisa

Pesquisa binária (algoritmo)

- A ideia é comparar o elemento a pesquisar com o elemento que está ao meio do vector e analisar 3 situações diferentes:
  - 1ª) se aquele elemento é igual ao que está ao meio,
  - 2ª) se aquele elemento está antes do meio,
  - 3ª) se aquele elemento está depois do meio.
- Se aconteceu a 1ª situação, então foi encontrado o elemento e está no vector naquela posição.
- Se aconteceu a 2ª situação, então basta pesquisar aquele elemento no subvector até ao meio.
- Se aconteceu a 3ª situação, então basta pesquisar aquele elemento no subvector do meio para a frente

Algoritmos de pesquisa

Pesquisa binária (algoritmo iterativo)

***Pesquisar o elemento* Elem *no vector* V *de tamanho* tam**

```
inicio  ← 0
fim     ← tam - 1
k ← -1  // k recebe a posição de Elem (no início presume-se que não está)
Enquanto  ( (inicio ≤ fim) e (k = -1) )  Fazer
    meio ← (inicio + fim) / 2
    Se (Elem = V[meio]) então
        k ← meio
    senão
        Se (Elem < V[meio]) então
            fim ← meio – 1
        senão
            inicio ← meio + 1
Se (k ≥ 0) então Elem encontra-se em V na posição k
                 senão Elem não se encontra em V
```

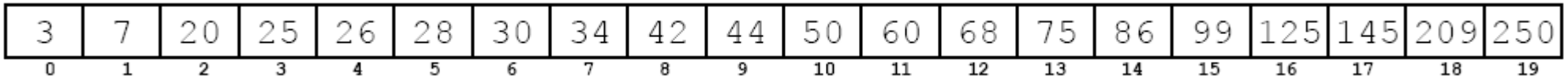
Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

```
int PesquisaBinaria (int Elem, int V[], int tam)
{
   int  inicio = 0, fim = tam – 1, meio, k = -1;
   while ( (inicio <= fim)  &&  (k == -1) )
   {
      meio = (inicio + fim) / 2;
      if (Elem == V[meio])
         k = meio;
      else
         if (Elem < V[meio])
            fim = meio - 1;
         else
            inicio = meio + 1;
   }
   return (k);
}
```

Algoritmos de pesquisa

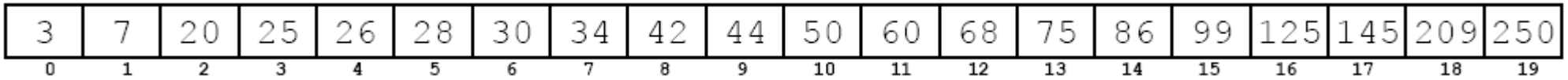
Pesquisa binária (versão iterativa)



Valor a pesquisar: 34

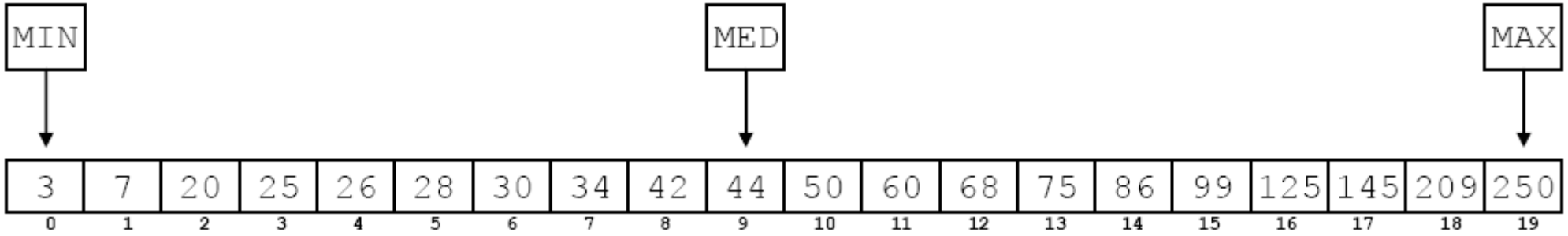
Algoritmos de pesquisa

Pesquisa binária (versão iterativa)



Valor a pesquisar: 34

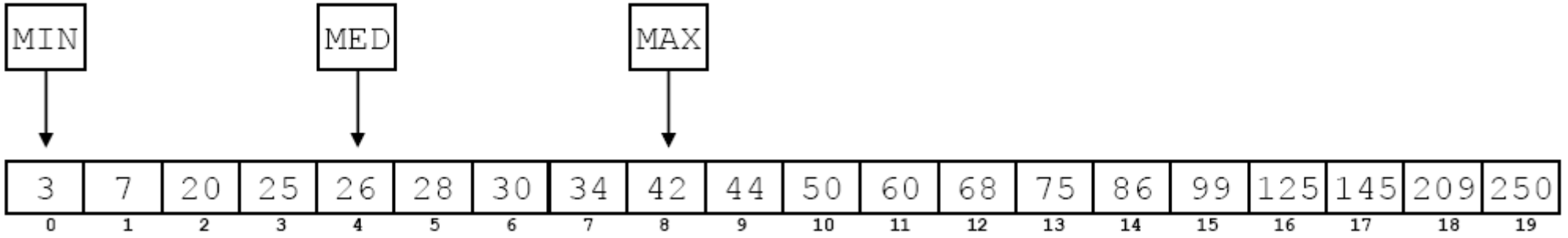
1ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

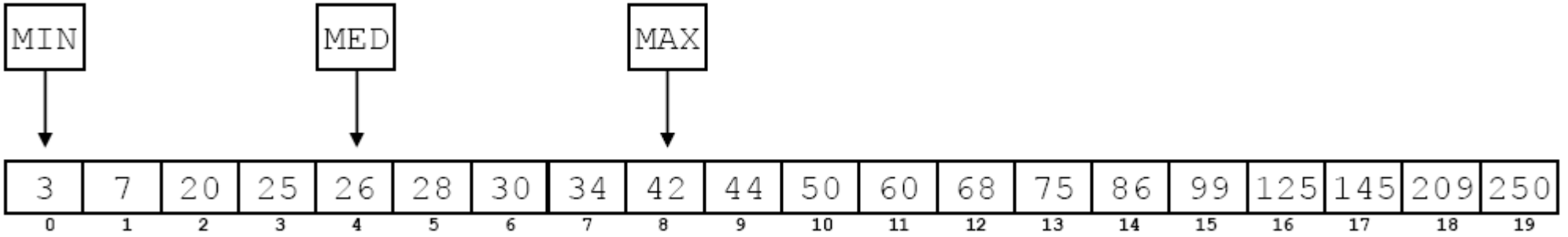
2ª tentativa:



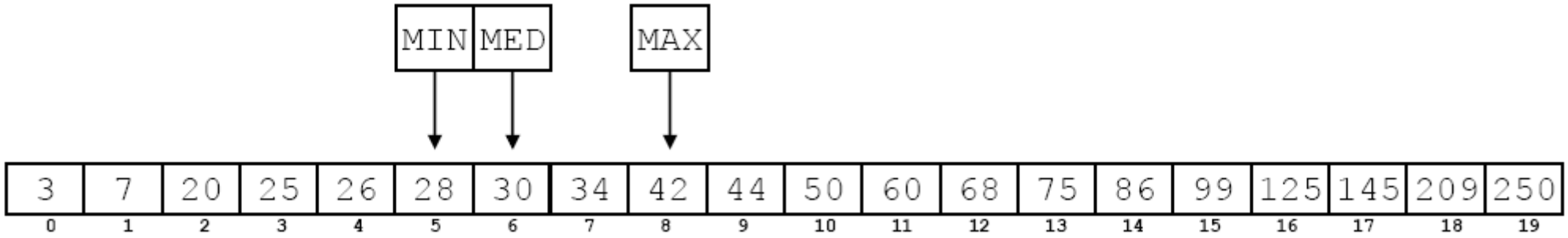
Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

2ª tentativa:



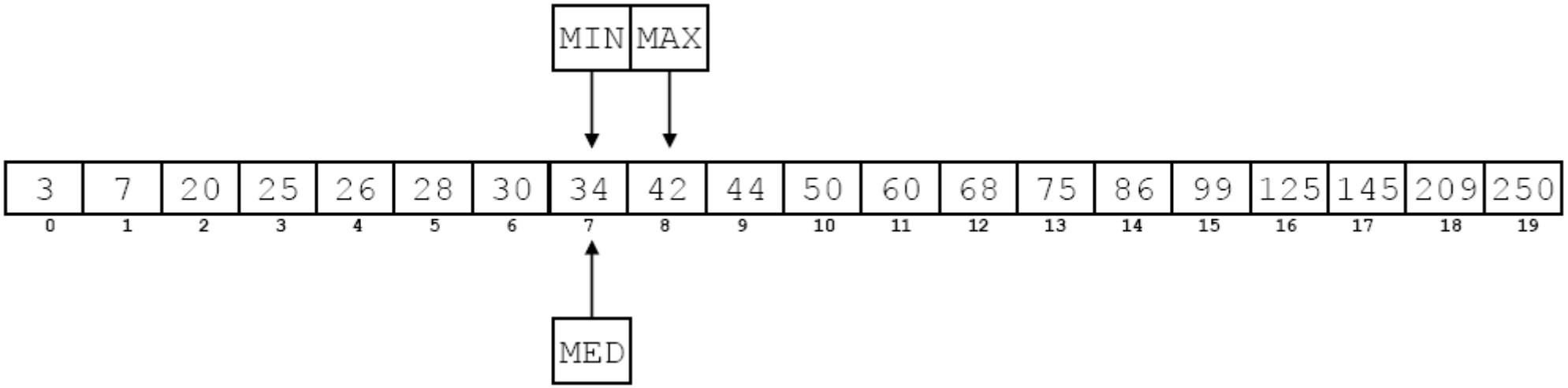
3ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

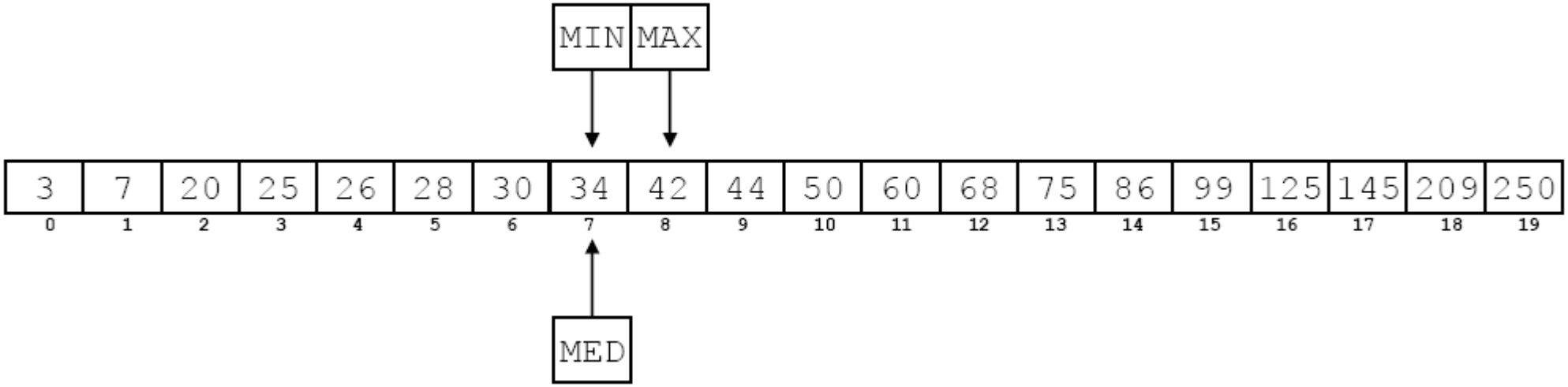
4ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

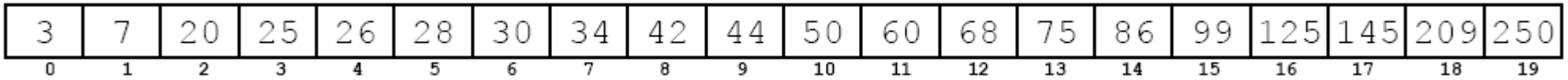
4ª tentativa:



O valor 34 foi encontrada após 4 tentativas

Algoritmos de pesquisa

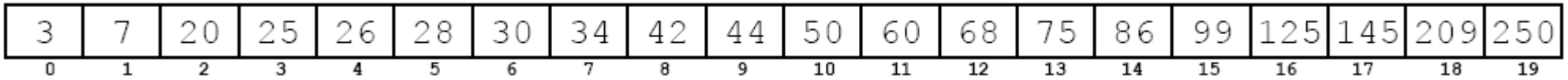
Pesquisa binária (versão iterativa)



Valor a pesquisar: 40

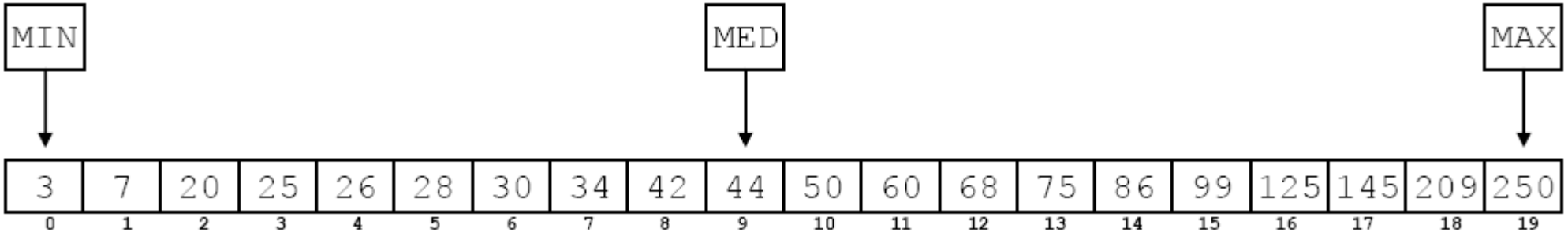
Algoritmos de pesquisa

Pesquisa binária (versão iterativa)



Valor a pesquisar: 40

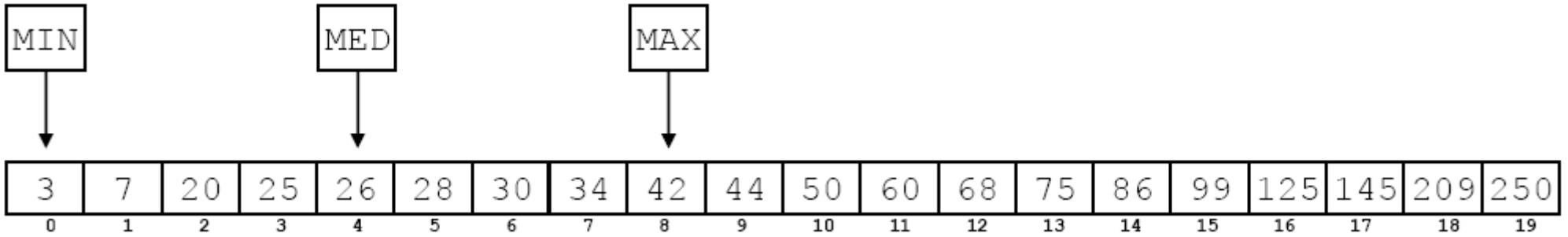
1ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

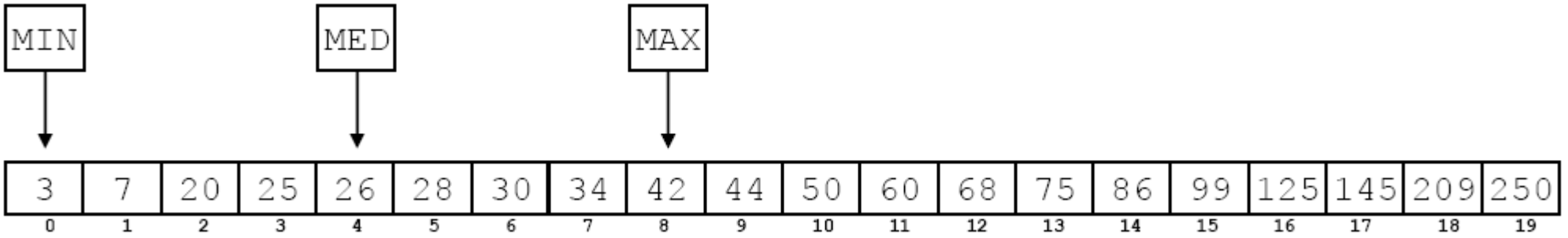
2ª tentativa:



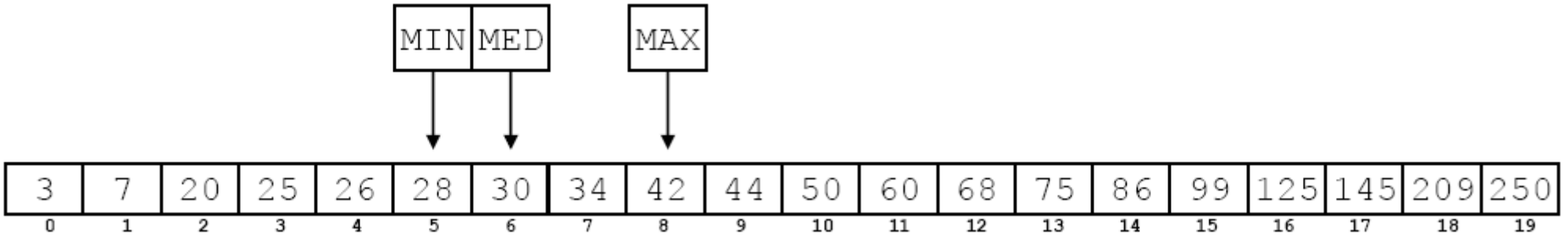
Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

2ª tentativa:



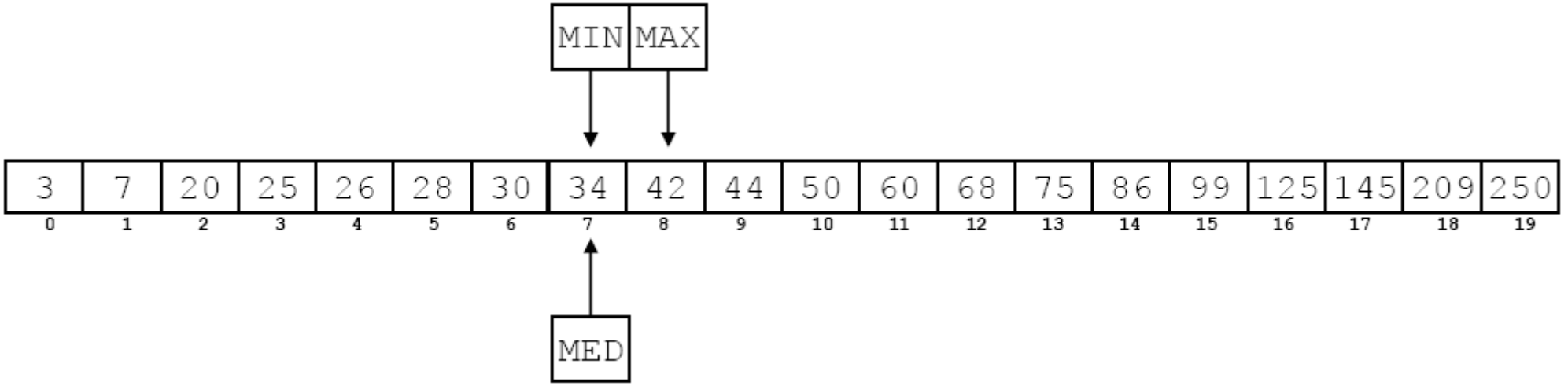
3ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

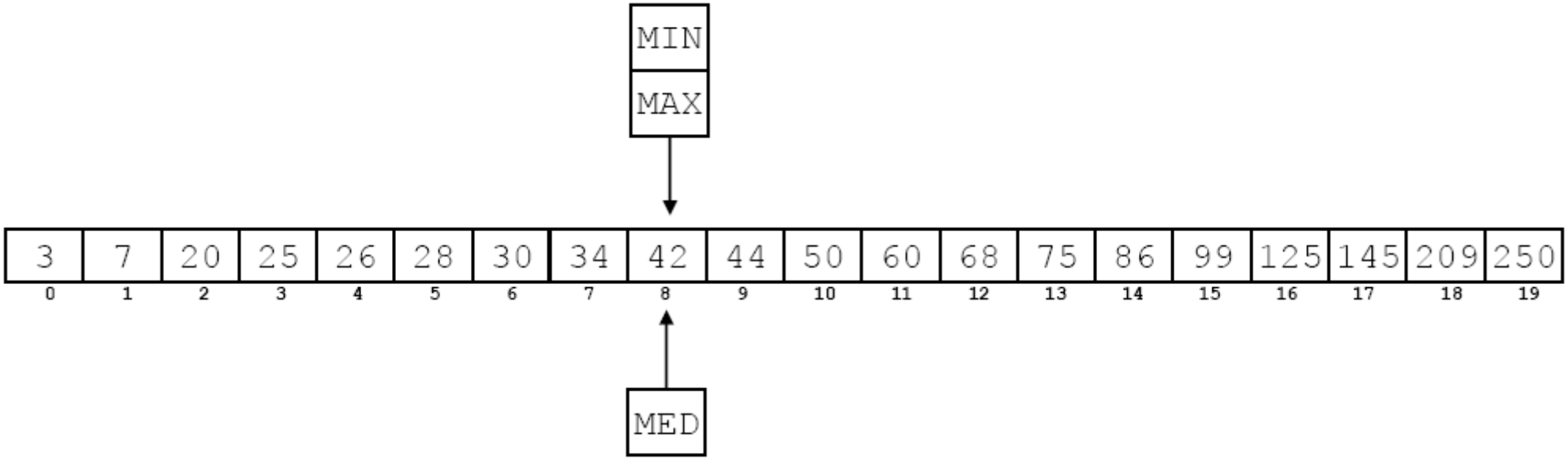
4ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

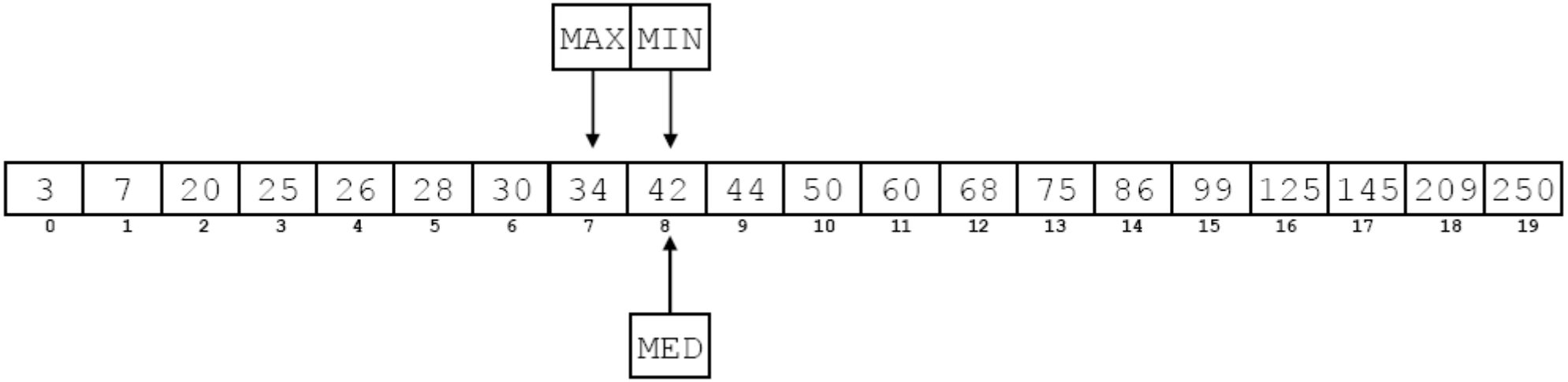
5ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

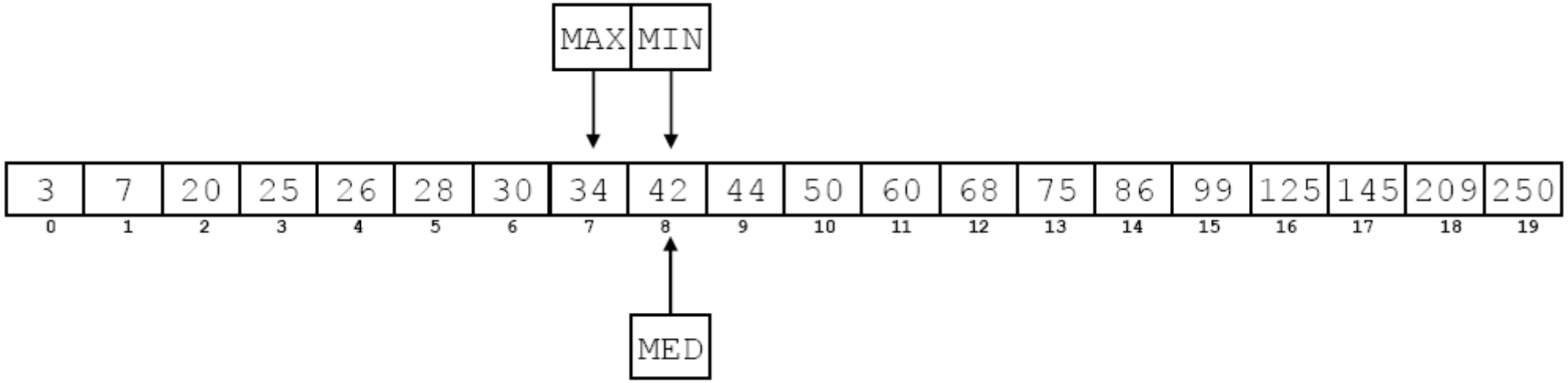
6ª tentativa:



Algoritmos de pesquisa

Pesquisa binária (versão iterativa)

6ª tentativa:



O valor 40 não está na lista,

o que se concluiu após 6 tentativas