ALGORITMOS E ESTRUTURAS DE DADOS

Busca Linear e Binária

Busca

- A hipótese básica assumida no processo de busca é que o conjunto de dados, dentre o qual um determinado elemento deve ser procurado, possui tamanho fixo, ou seja, um vetor:

```
item a[N];
```

- onde **item** representa uma estrutura de dados contendo um campo que atua como chave para a pesquisa e N é uma constante indicando o número de elementos

```
typedef struct
{ int key;    // chave de busca
  ...         // demais campos da estrutura
} item;
```

- Objetivo da busca: dado **x** encontrar **a[i].key == x**
- O índice **i** resultante permite acesso aos demais campos

Busca

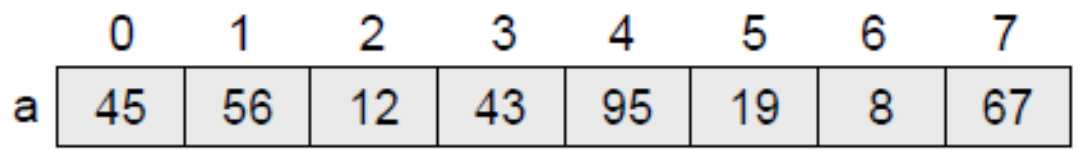
- Para estudo, vamos admitir que o tipo **item** seja composto apenas do campo chave, ou seja, o dado é a própria chave.
- Além disso, para facilitar o estudo ainda mais, a chave de busca será um inteiro, ou seja, o vetor **a** será declarado como:

```
int a[N];
```

- Lembrando que N é uma constante que indica o número de elementos do vetor
- Assim, objetivo da busca se resume a dado **x** encontrar **a[i] == x**

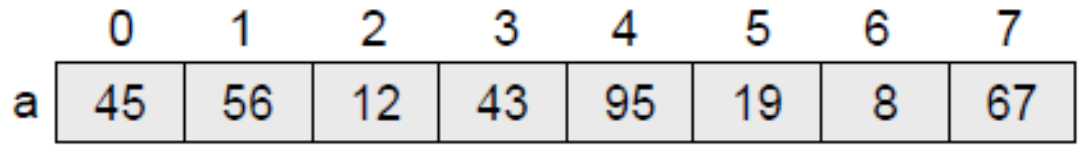
Exemplo

- Busca de x = 19, retorna i = 5
- Busca de x = 45, retorna i = 0
- Busca de x = 8, retorna i = 6
- E a busca de x = 81?



Exemplo

- Busca de x = 19, retorna i = 5
- Busca de x = 45, retorna i = 0
- Busca de x = 8, retorna i = 6
- E a busca de x = 81?



- Depende da implementação!
- Pode retornar i = -1 (ou outro valor) indicativo que a busca não teve êxito

Busca Linear (ou Sequencial)

- Utilizada quando não há informações adicionais sobre os dados a serem pesquisados
- A busca linear termina quando for satisfeita uma das duas condições seguintes:
  1. O elemento é encontrado, isto é, **a[i] == x**
  2. Todo o vetor foi analisado, mas o elemento x não foi encontrado
- Algoritmo:

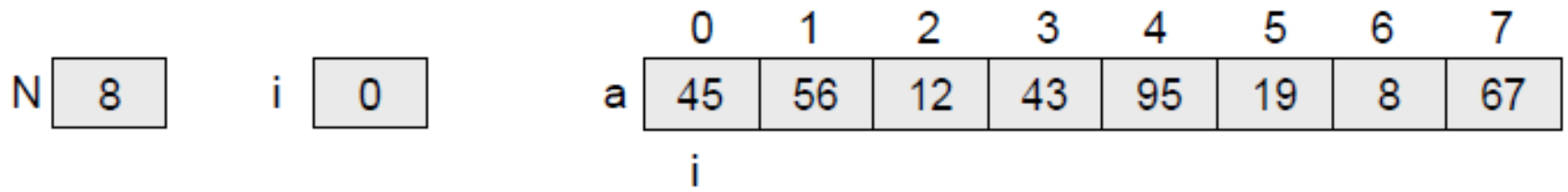
```
i = 0;
while (i < N && a[i] != x)
   i++;
```

- Ao término do laço:
  - Se **i == N** então x não foi encontrado
  - senão **a[i] == x**, **i** é a posição onde **x** foi encontrado

Busca Linear (ou Sequencial)

- Busca de x = 19

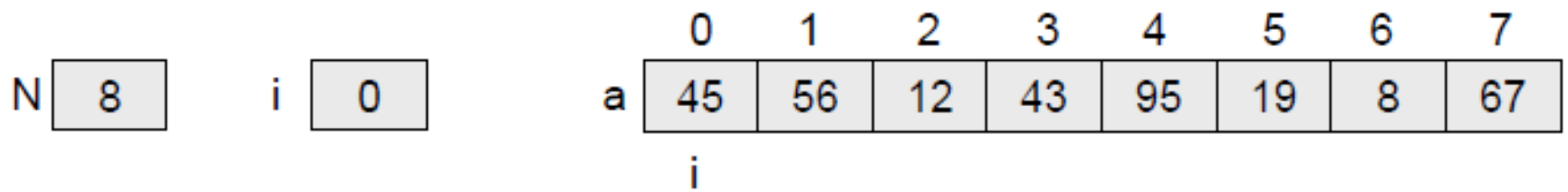
```
i = 0;


while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

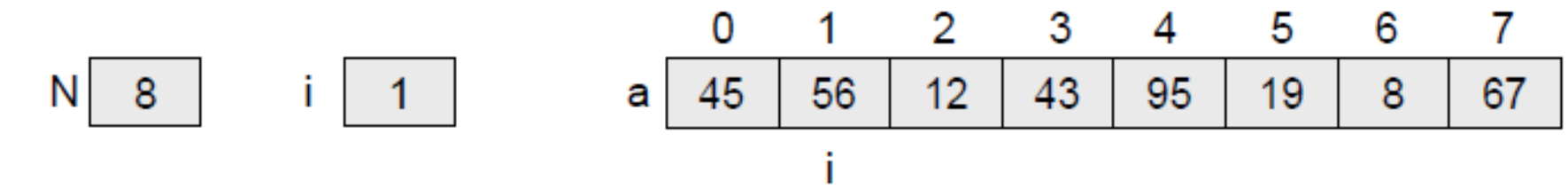
```
i = 0;
                    V                    V
while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

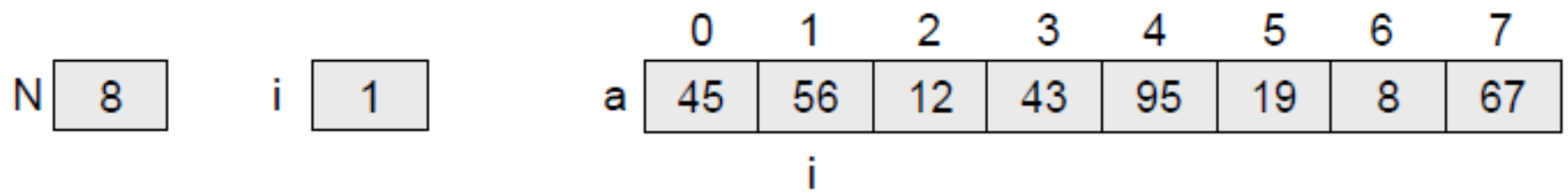
```
i = 0;


while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

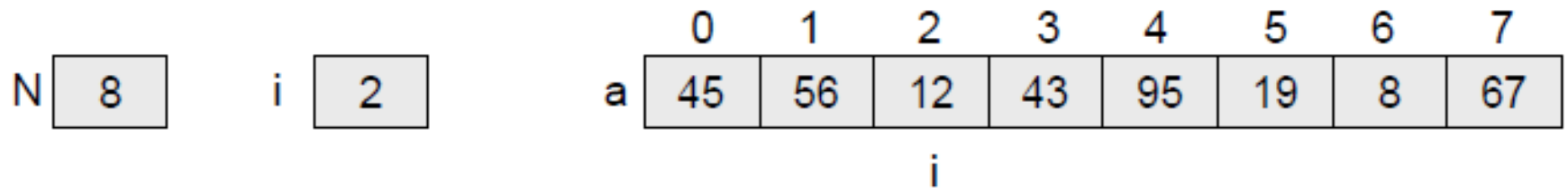
```
i = 0;
                    V                    V
while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

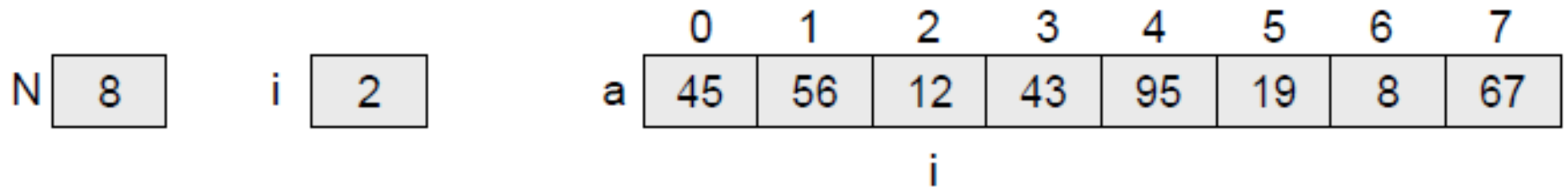
```
i = 0;

while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

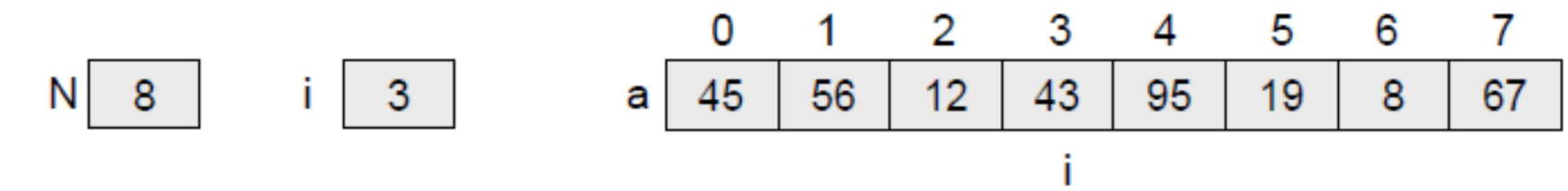
```
i = 0;
                    V                    V
while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

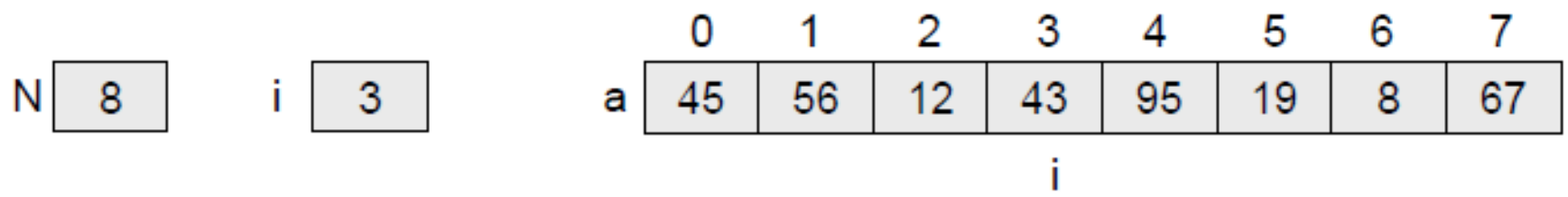
```
i = 0;


while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

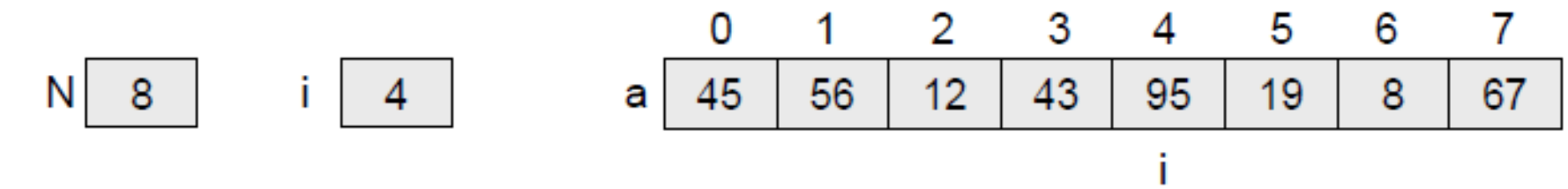
```
i = 0;
                    v                   v
while (i < N && a[i] != x)
    i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

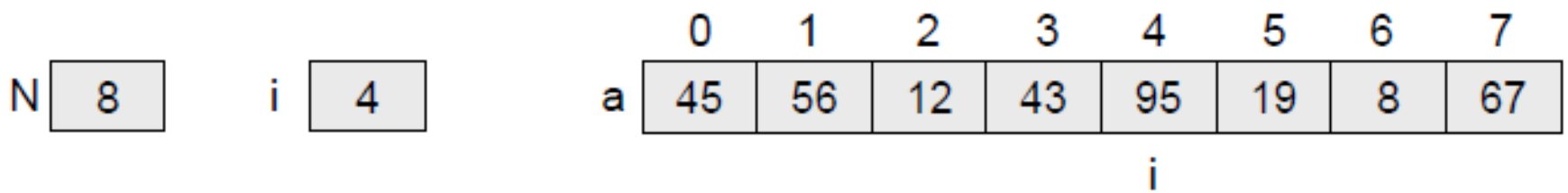
```
i = 0;


while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

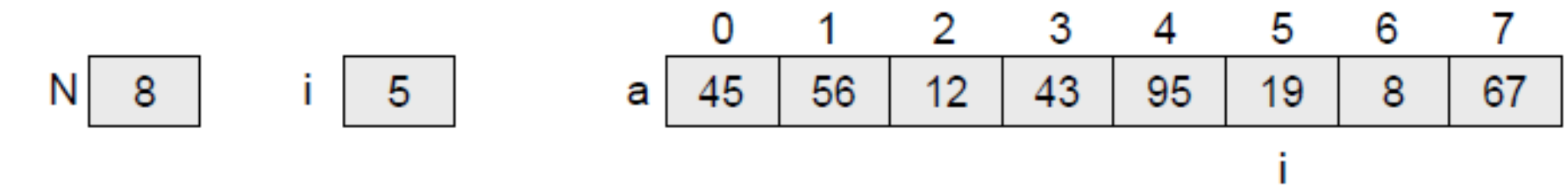
```
i = 0;
                    V                   V
while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

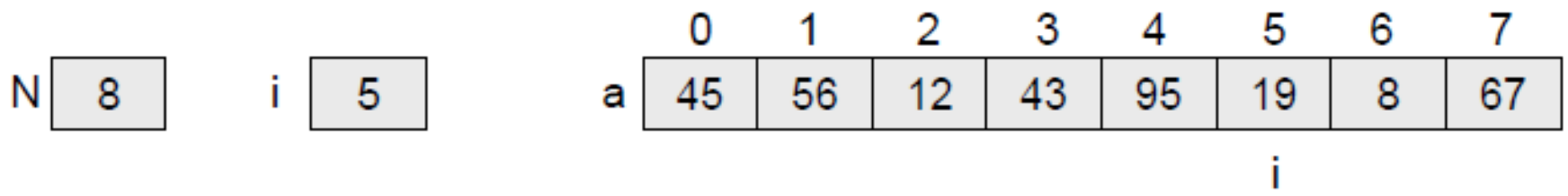
```
i = 0;

while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

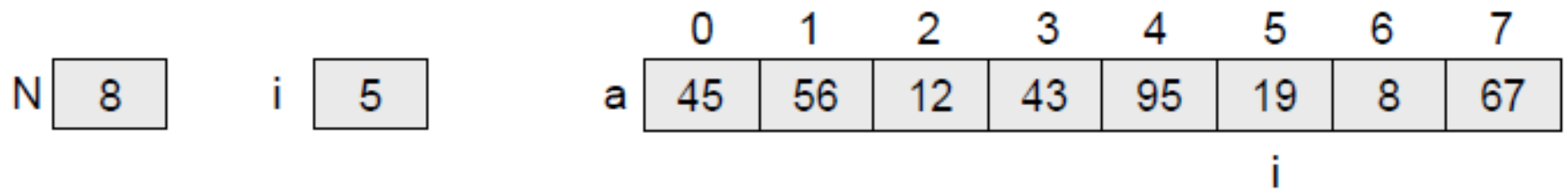
```
i = 0;
                    V                   F
while (i < N && a[i] != x)
   i++;
```



Busca Linear (ou Sequencial)

- Busca de x = 19

```
i = 0;

while (i < N && a[i] != x)
   i++;
```



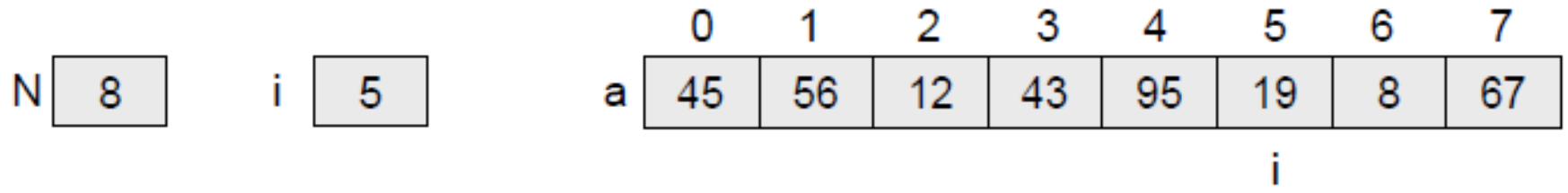
- Término do laço: Se **i != N** então **x** foi encontrado na posição **i** do vetor

Busca Linear (ou Sequencial)

- Busca de x = 19

```
i = 0;

while (i < N && a[i] != x)
   i++;
```

Agora façam isso em C++



- Término do laço: Se **i != N** então **x** foi encontrado na posição **i** do vetor

Exemplo em C++

```
#include <iostream>
using namespace std;

int busca_sequencial(int x, int N, int a[])
{ int i = 0;

  while (i < N && a[i] != x)
    i++;
  return (i == N) ? -1 : i;
}

int main(void)
{ const int m = 8;
  int v[m] = {45,56,12,43,95,19,8,67};

  cout << "Busca de 19 = " << busca_sequencial(19,m,v) << endl;
  cout << "Busca de 45 = " << busca_sequencial(45,m,v) << endl;
  cout << "Busca de 8  = " << busca_sequencial(8,m,v) << endl;
  cout << "Busca de 81 = " << busca_sequencial(81,m,v) << endl;
  return 0;
}
```

```
Busca de 19 = 5
Busca de 45 = 0
Busca de 8  = 6
Busca de 81 = -1
```

Análise da Busca Linear

- Em média são efetuadas **N/2** comparações de chaves para encontrar um elemento particular **x** no vetor **a** de **N** elementos
- O pior caso requerer **N** comparações de chaves
- Isso pode consumir muito tempo quando o número de elementos do vetor é grande

Busca Linear com Sentinela

- O uso da sentinela tem como objetivo acelerar a busca, através da simplificação da expressão booleana
- A idéia básica é fazer com que o elemento **x** sempre seja encontrado
- Para isso, introduz-se um elemento adicional no final do vetor

Busca Linear com Sentinela

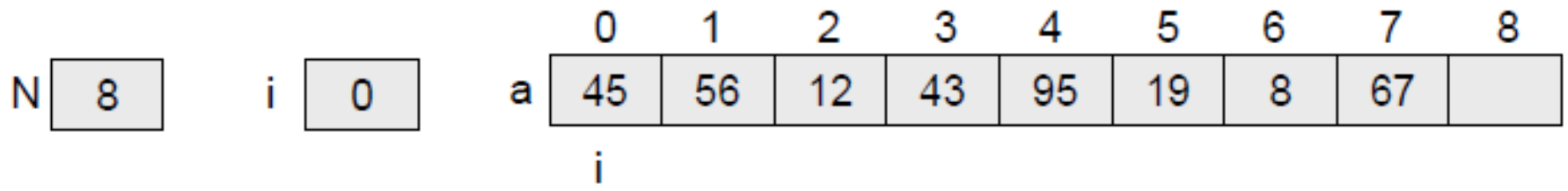
- Algoritmo:

```
item a[N+1];
i = 0;
a[N] = x;    // sentinela
while (a[i] != x)
   i++;
```

- Ao final do laço, **i == N** implica que **x** não foi encontrado (exceto o correspondente à sentinela).

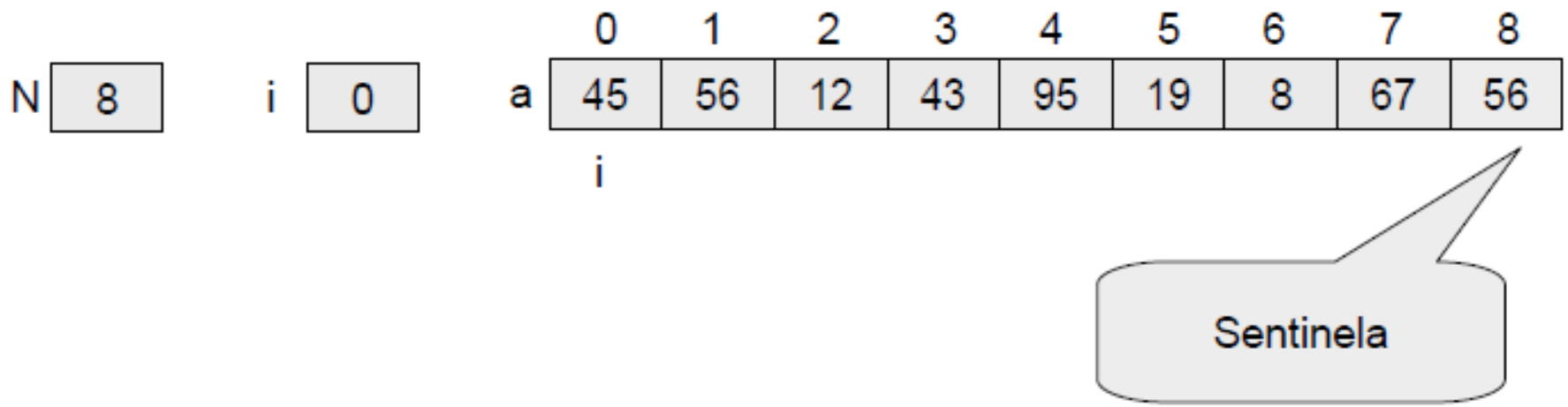
Busca de x = 56

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```



Busca de x = 56

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

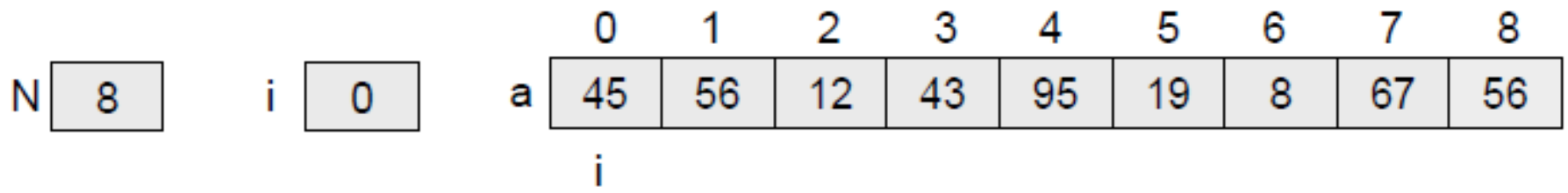


Busca de x = 56

```
i = 0;
a[N] = x;
```

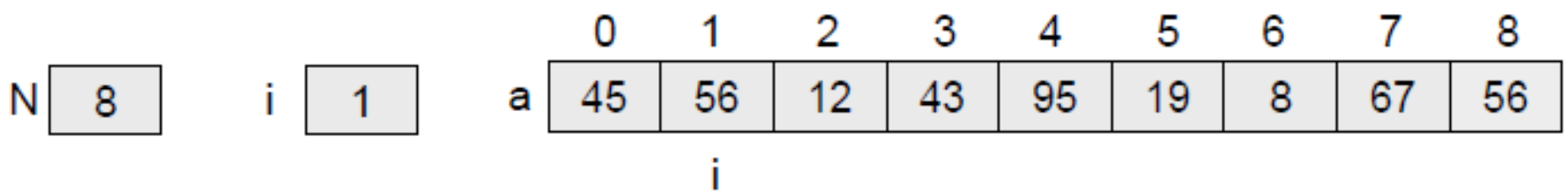
v

```
while (a[i] != x)
   i++;
```



Busca de x = 56

```
i = 0;
a[N] = x;

while (a[i] != x)
    i++;
```

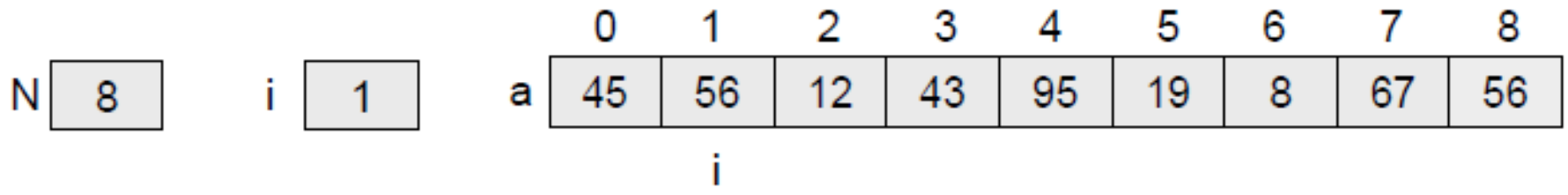


Busca de x = 56

```
i = 0;
a[N] = x;
```

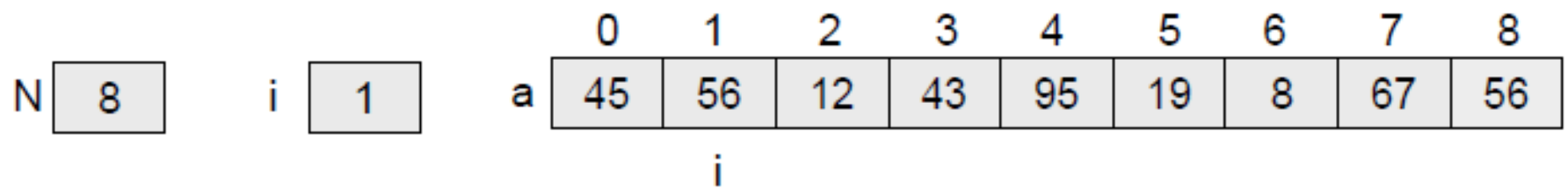
F

```
while (a[i] != x)
   i++;
```



Busca de x = 56

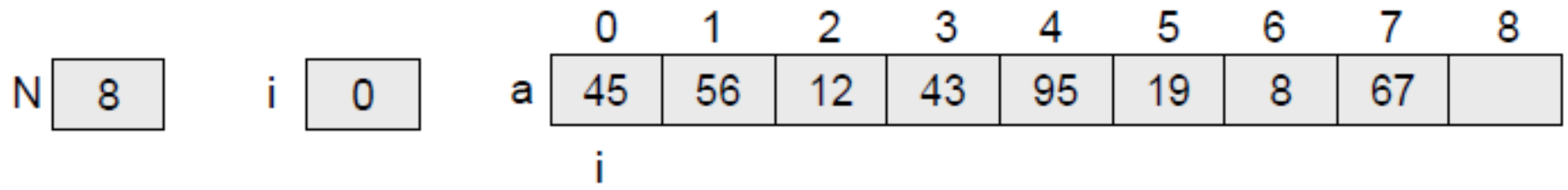
```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```



- Término do laço: Se **i != N** então **x** foi encontrado na posição **i** do vetor

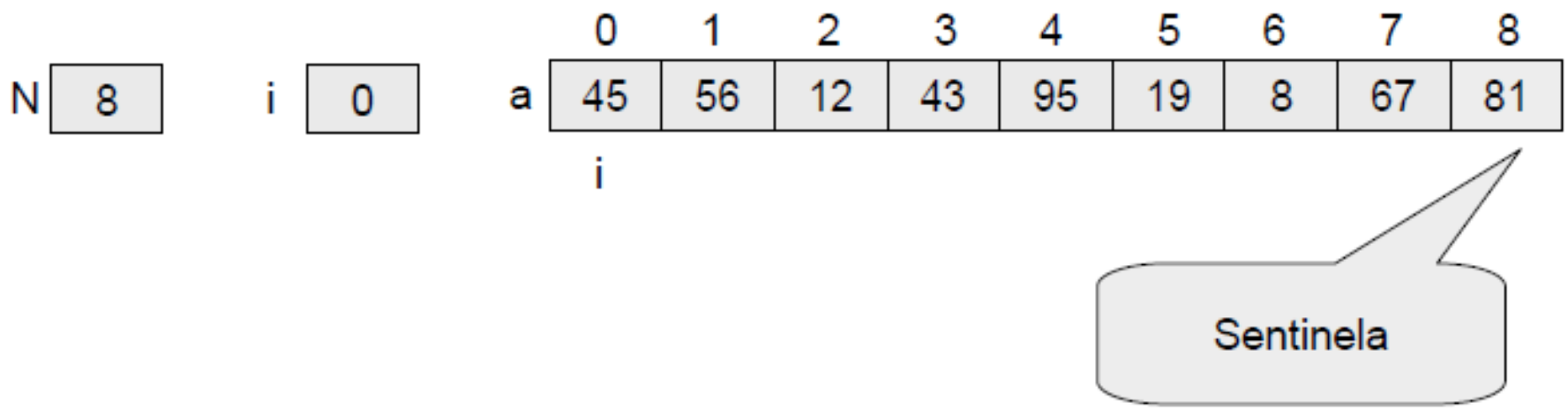
Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

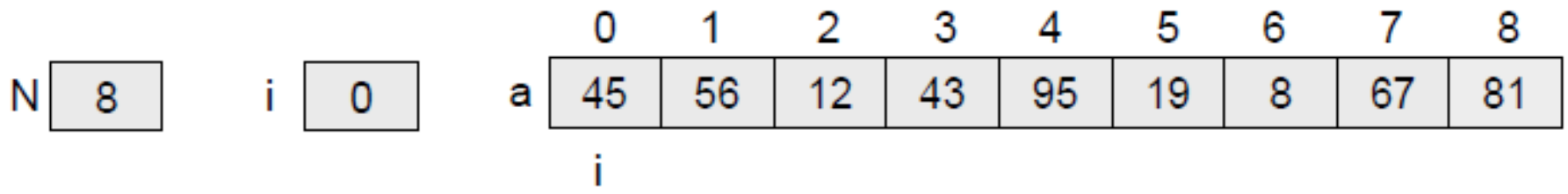


Busca de x = 81

```
i = 0;
a[N] = x;
```

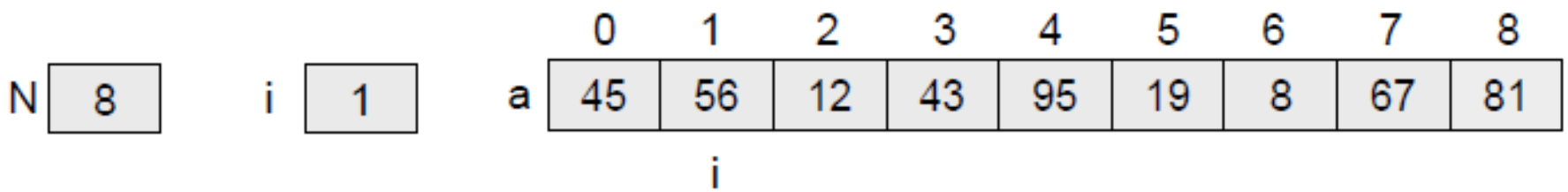
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

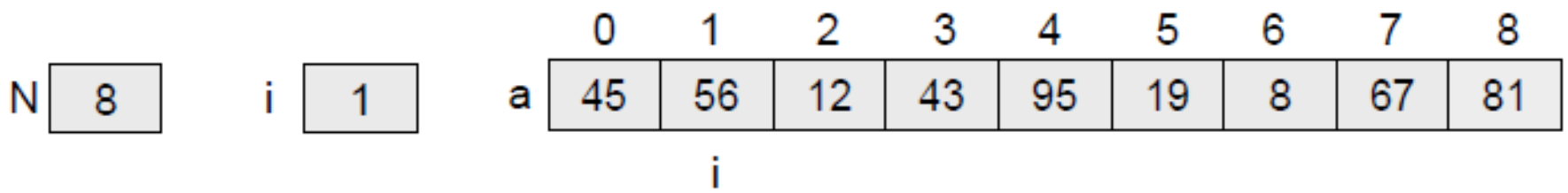


Busca de x = 81

```
i = 0;
a[N] = x;
```

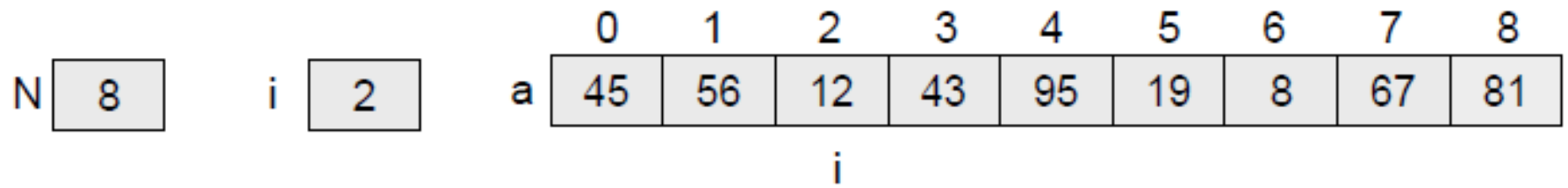
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

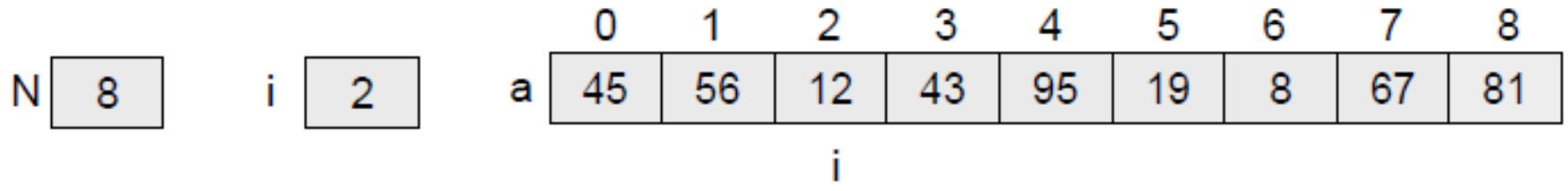


Busca de x = 81

```
i = 0;
a[N] = x;
```

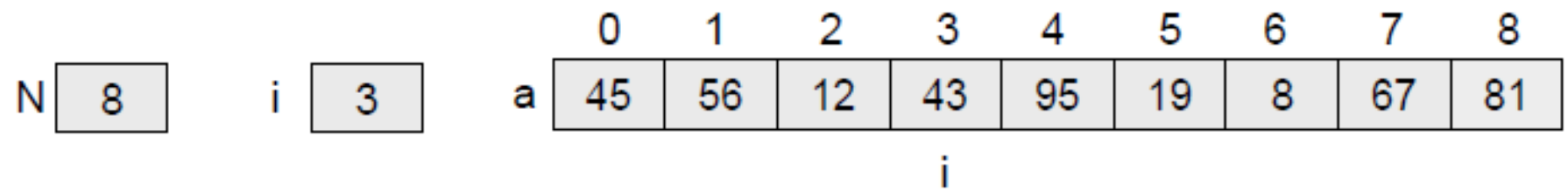
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

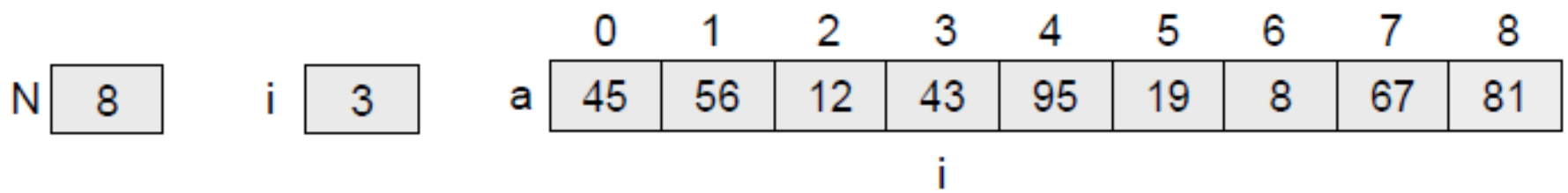


Busca de x = 81

```
i = 0;
a[N] = x;
```

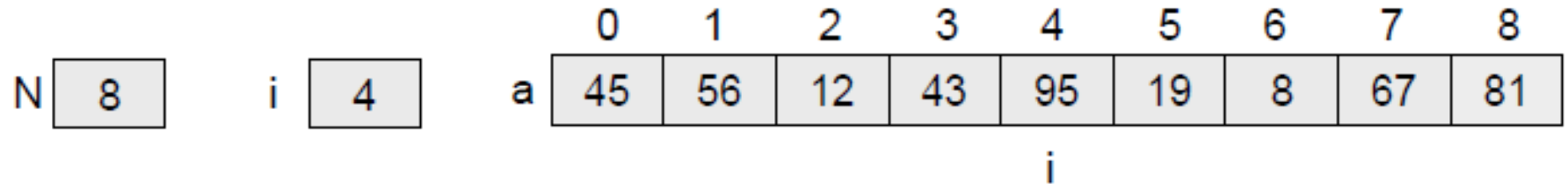
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

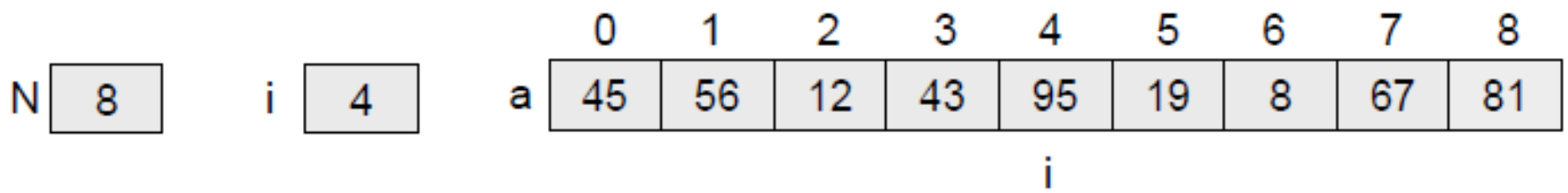


Busca de x = 81

```
i = 0;
a[N] = x;
```

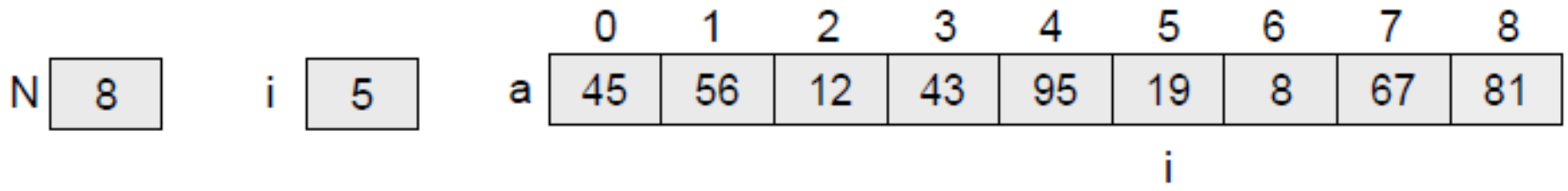
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

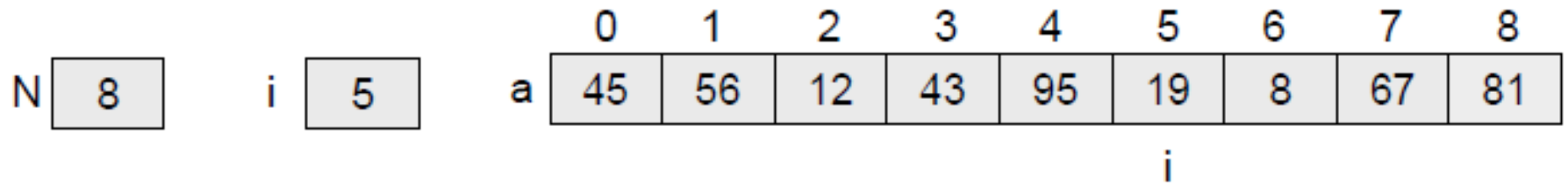


Busca de x = 81

```
i = 0;
a[N] = x;
```

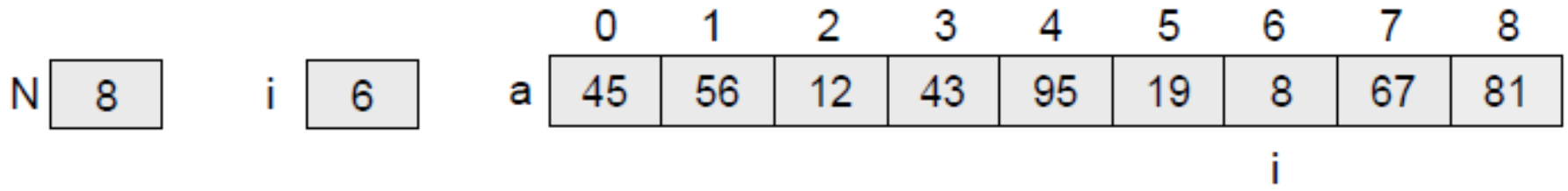
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

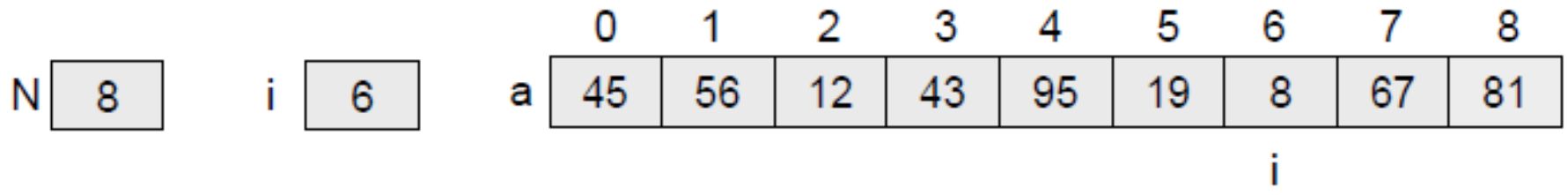


Busca de x = 81

```
i = 0;
a[N] = x;
```

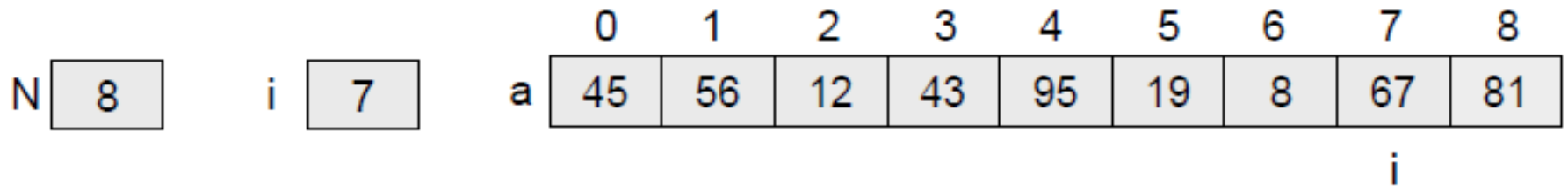
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

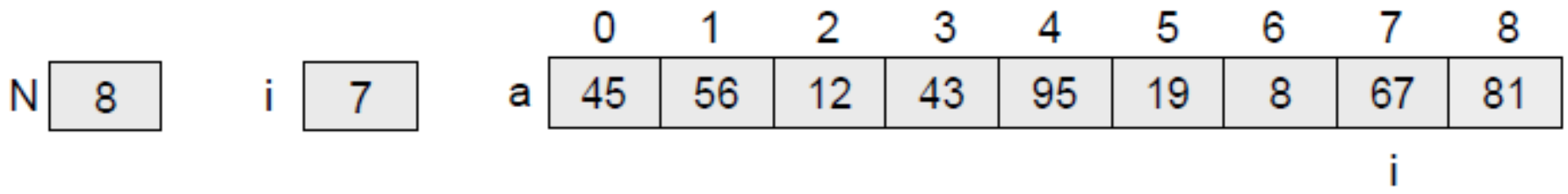


Busca de x = 81

```
i = 0;
a[N] = x;
```

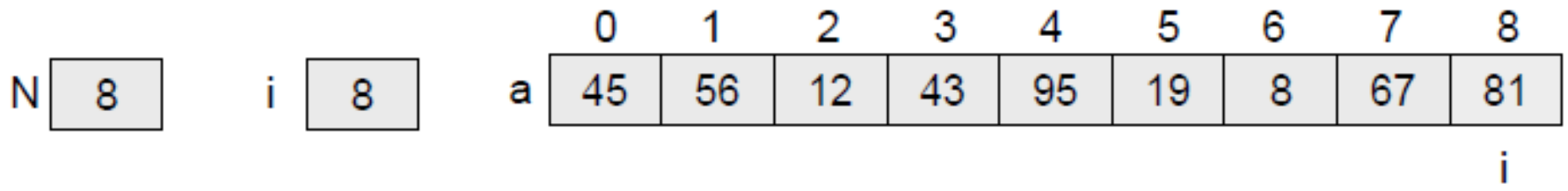
v

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```

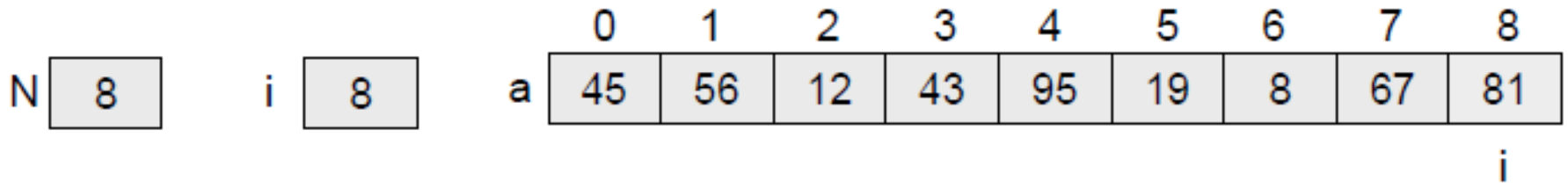


Busca de x = 81

```
i = 0;
a[N] = x;
```

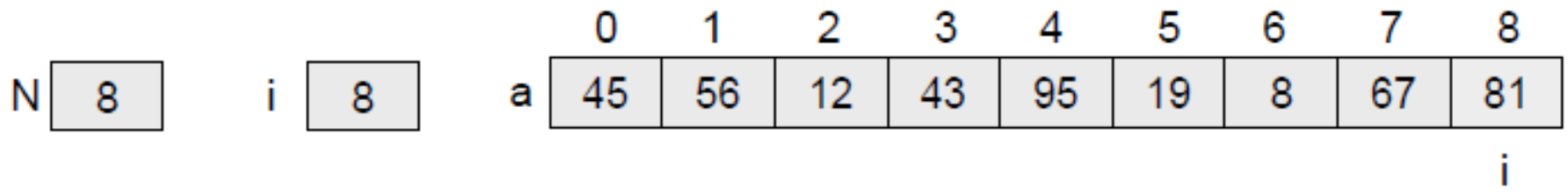
F

```
while (a[i] != x)
   i++;
```



Busca de x = 81

```
i = 0;
a[N] = x;

while (a[i] != x)
   i++;
```



- Término do laço: Se **i != N** então **x** foi encontrado na posição **i** do vetor. Como **i == N**, então **x** não foi encontrado (exceto sentinela)

Exemplo em C++

```
#include <iostream>
using namespace std;

int busca_sentinela(int x, int N, int a[])
{ int i = 0;

  a[N] = x;  // sentinela
  while (a[i] != x)
    i++;
  return (i == N) ? -1 : i;
}

int main(void)
{ const int m = 8;
  int v[m+1] ={45,56,12,43,95,19,8,67};

  cout << "Busca de 19 = " << busca_sentinela(19,m,v) << endl;
  cout << "Busca de 45 = " << busca_sentinela(45,m,v) << endl;
  cout << "Busca de 8  = " << busca_sentinela(8,m,v) << endl;
  cout << "Busca de 81 = " << busca_sentinela(81,m,v) << endl;
  return 0;
}
```

```
Busca de 19 = 5
Busca de 45 = 0
Busca de 8  = 6
Busca de 81 = -1
```

Exemplo em C++

Seja uma sequência de números inteiros (distintos) armazenadas em um array A[i], $1 \leq i \leq n$. Apresentar um programa para realizar uma busca sequencial (com sentinela) de um elemento x no array A. Considere n = 10. Esse programa deve apresentar, no mínimo, um menu com as seguintes opções:

1) Inserir elemento no array;
2) Apresentar array;
3) Informar elemento a ser buscado;
4) Sair do programa.

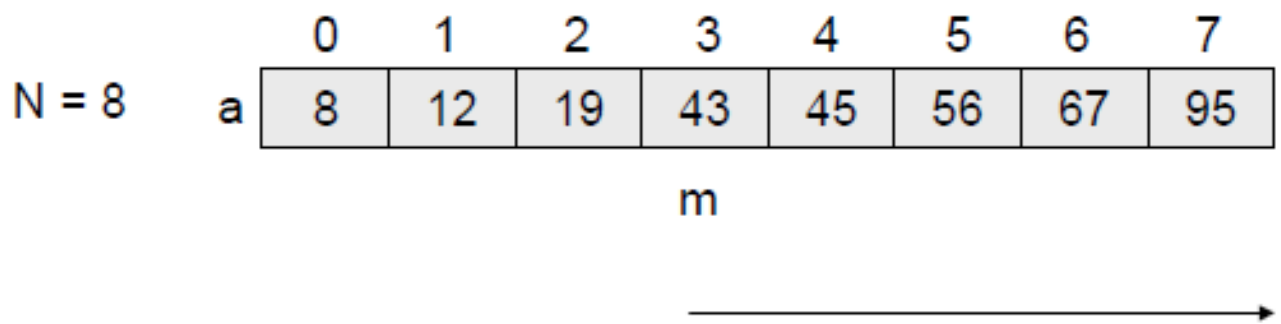
A resposta da busca deve informar se o elemento está ou não no array e, em caso positivo, em qual a sua posição no array. O programa deve estar orientado a objetos.

Busca Binária

- Não é possível acelerar a busca sem que se disponha de maiores informações acerca do elemento a ser localizado
- Sabe-se que uma busca pode ser mais eficiente se os dados estiverem ordenados, ou seja:
  a[0] <= a[1] <= … <= a[N-1]
- A idéia principal é a de teste um elemento sorteado aleatoriamente, por exemplo, **a[m],** comparando-o com o elemento de busca **x**.
  - Se tal elemento for igual a **x**, a busca termina.
  - Se for menor que **x**, conclui-se que todos os elementos com índices menores ou iguais a **m** podem ser eliminados dos próximos testes.
  - Se for maior que **x**, todos aqueles elementos com índices maiores ou iguais a **m** podem ser também eliminados da busca.

Busca Binária

- Busca de x = 19
- Suponha m = 3



Como a[m] > x,
Elementos com índices maiores que
m podem ser eliminados da busca

Busca Binária

- Algoritmo:

	0	1	2	3	4	5	6	7
a	8	12	19	43	45	56	67	95

N = 8

```
L = 0;
R = N - 1;
achou = false;
while (L <= R && ! achou)
{ m = qualquer valor entre L e R;
  if (a[m] == x)
    achou = true;
  else
    if (a[m] < x)
      L = m + 1;
    else
      R = m - 1;
}
```

Busca Binária

- Embora a escolha de m seja aparentemente arbitrária (no sentido que o algoritmo funciona independentemente dele) o valor desta variável influencia na eficiência do algoritmo
- É claro que, a cada passo, deve-se eliminar o maior número possível de elementos em futuras buscas
- A solução ótima é escolher a mediana dos elementos, porque ela elimina, em qualquer caso, metade dos elementos do vetor

Busca Binária

- A eficiência pode ser ligeiramente melhorada através da permutação entre as duas cláusulas de comparação.
- A condição de igualdade deve ser testada em segundo lugar, porque o sucesso ocorre apenas uma vez em todo o processo
- Porém, a questão mais relevante se refere ao fato de, como na busca linear, se poder ou não encontrar uma solução que proporcione uma condição mais simples para a finalização do processo
- É possível obter tal algoritmo rápido se for abandonada a meta de terminar a busca no instante exato em que for encontrado o elemento pesquisado
- Isso parece pouco inteligente à primeira vista, mas observando-se mais a fundo, pode-se perceber facilmente que o ganho em eficiência em cada passo será maior do que a perda ocasionada pela comparação de alguns poucos elementos adicionais

Busca Binária Rápida

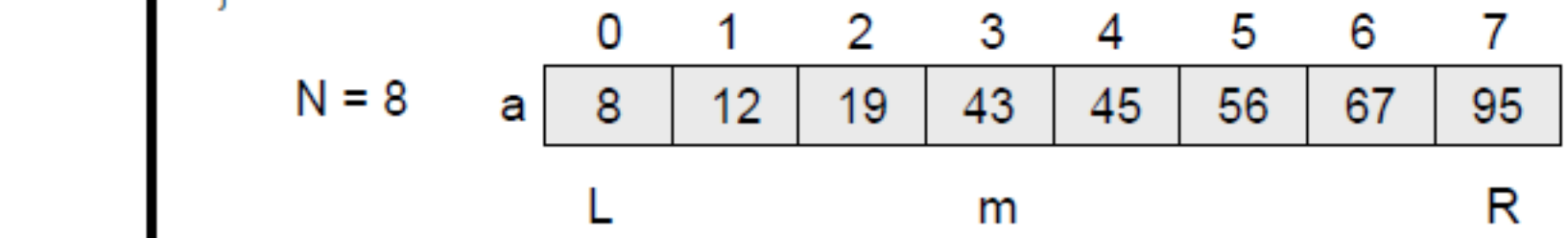
- Algoritmo:

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```

- Se ao término do algoritmo a condição **a[R] == x** for verdadeira, então **x** foi encontrado na posição **R** de **a**; caso contrário **x** não foi encontrado.

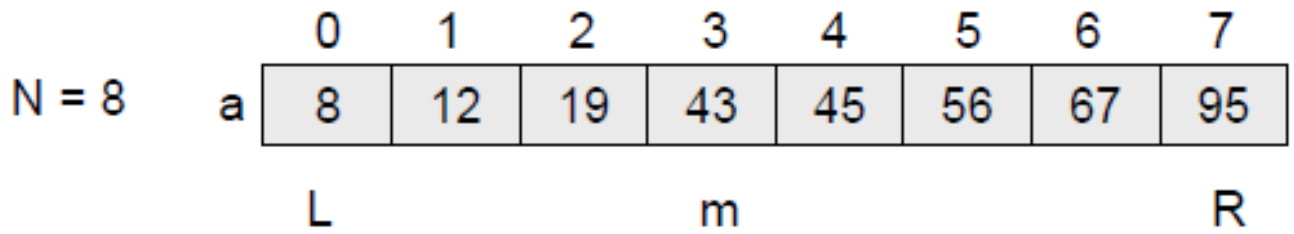
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



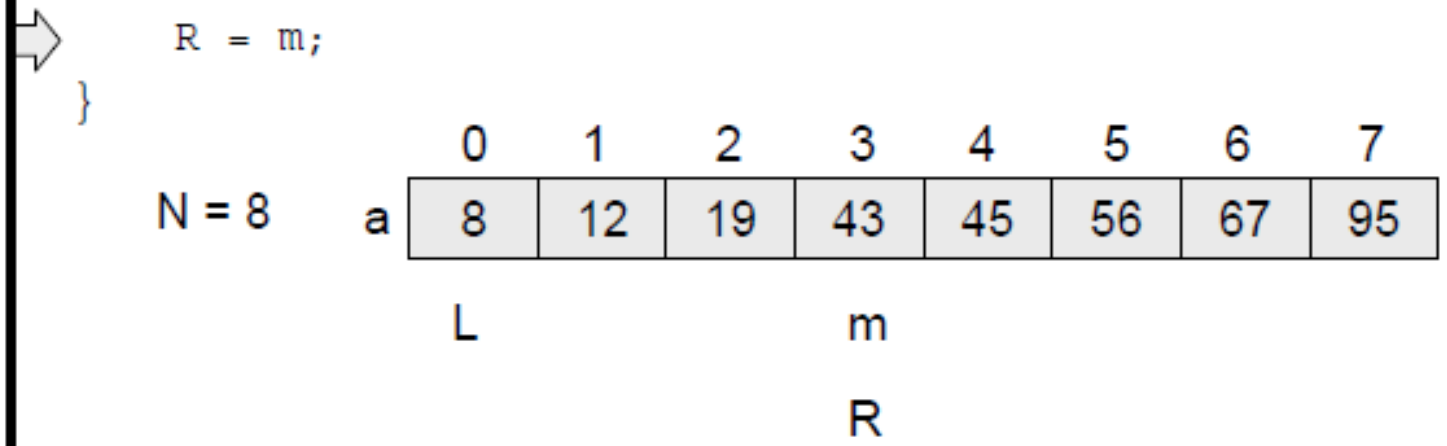
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



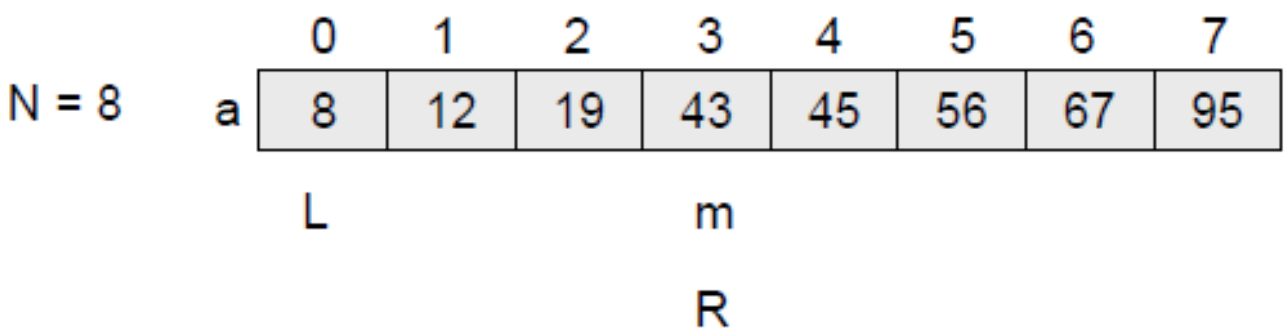
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



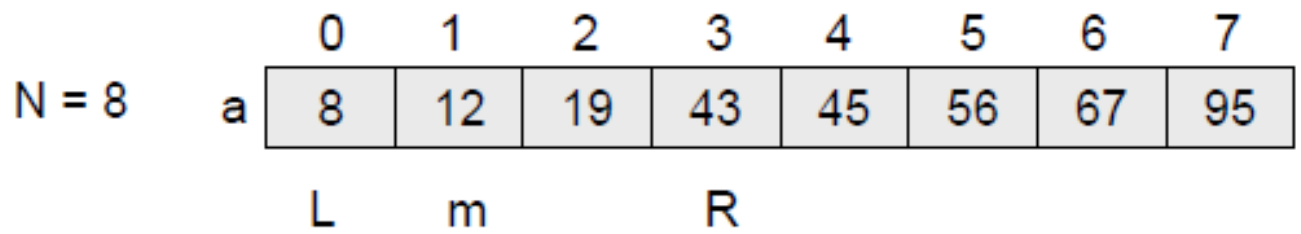
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



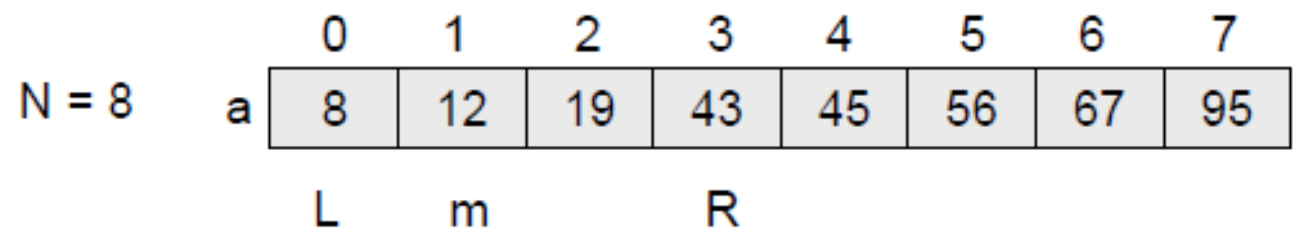
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



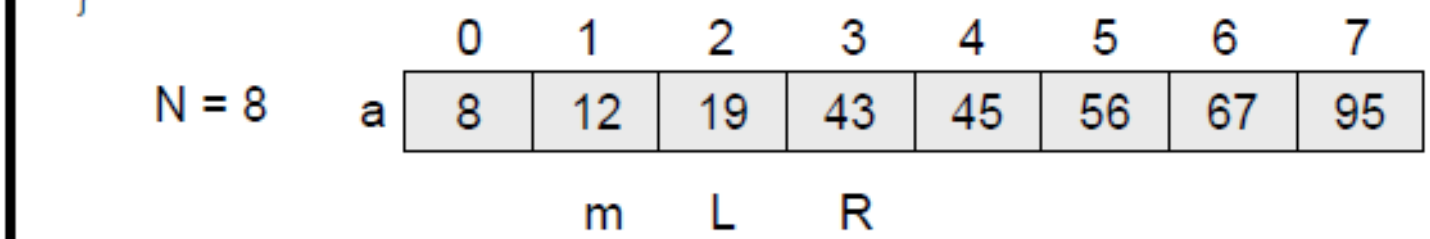
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



Busca de x = 19

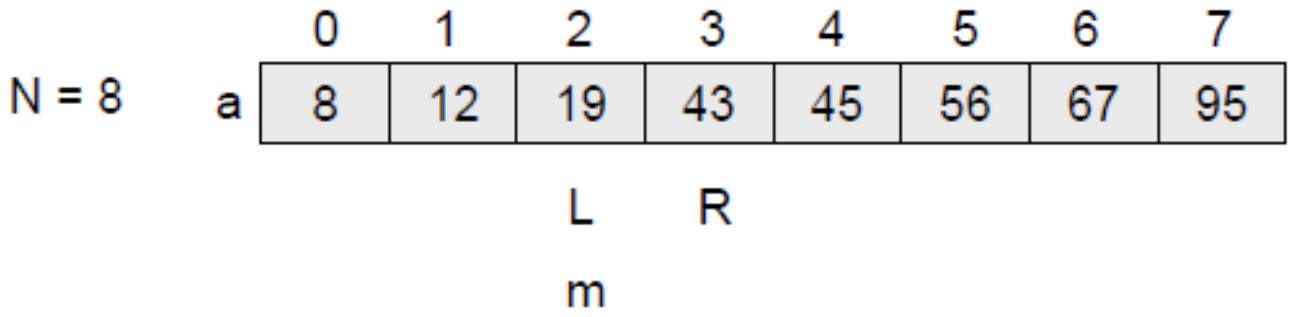
```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```

N = 8

	0	1	2	3	4	5	6	7
a	8	12	19	43	45	56	67	95
		m	L	R				

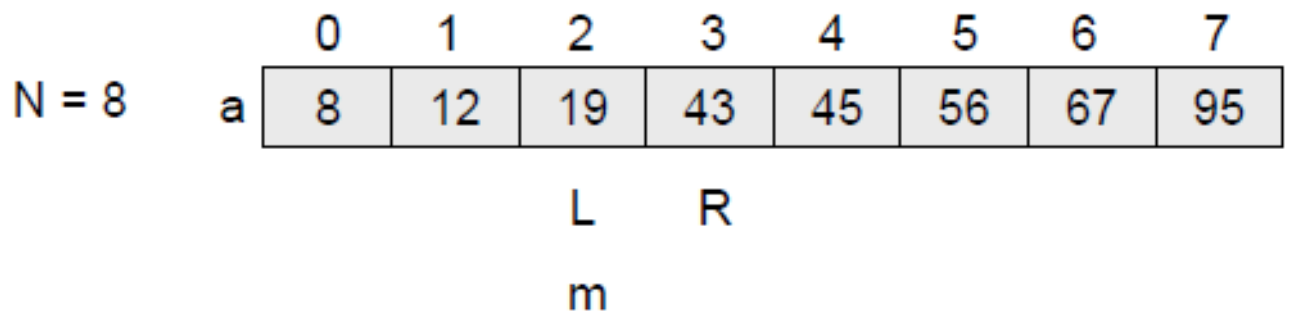
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



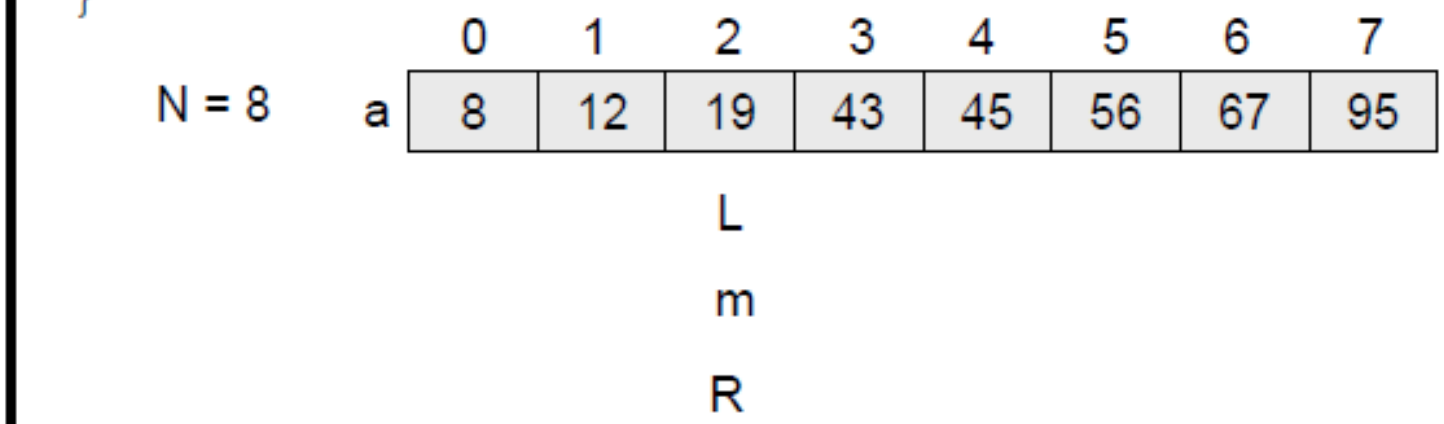
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



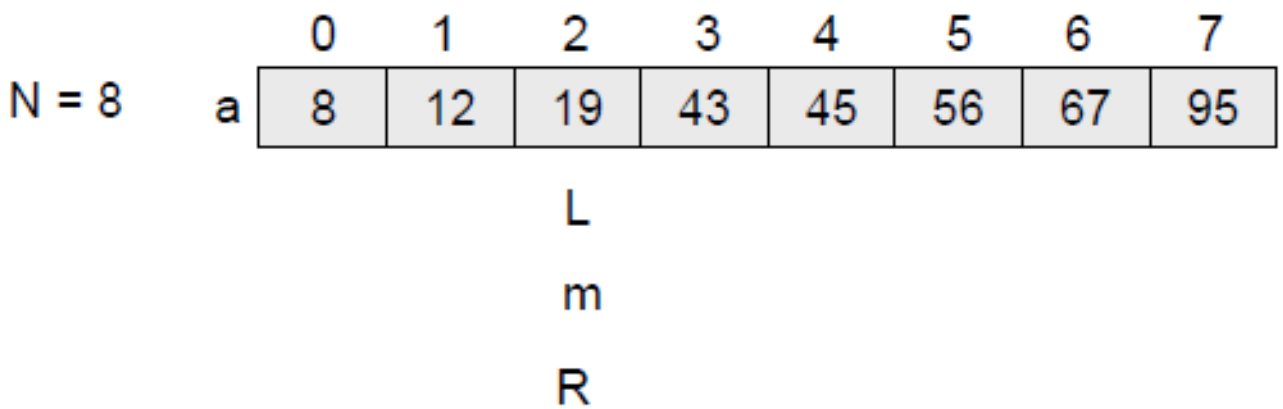
Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```



Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```

N = 8

	0	1	2	3	4	5	6	7
a	8	12	19	43	45	56	67	95

L

m

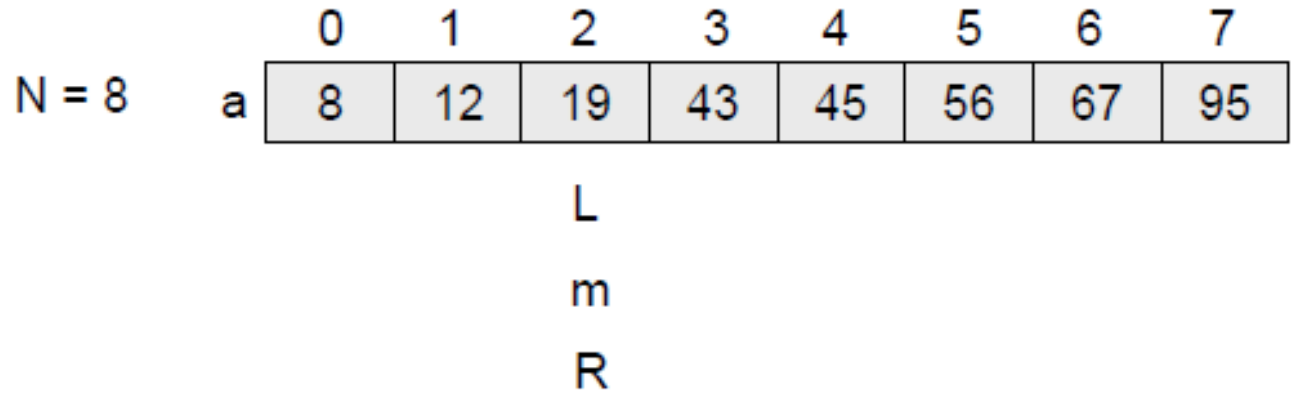
R

- Término do laço: Como **a[R] == x**, x foi encontrado na posição R de **a**

Busca de x = 19

```
L = 0;
R = N - 1;
while (L < R)
{ m = (L + R) / 2;
  if (a[m] < x)
    L = m + 1;
  else
    R = m;
}
```

Agora façam isso em C++



- Término do laço: Como **a[R] == x**, x foi encontrado na posição R de **a**

Exemplo em C++

```
#include <iostream>
using namespace std;
int busca_binaria_rapida(int x, int N, int a[])
{ int L,R,m;

  L = 0;
  R = N - 1;
  while (L < R)
  { m = (L + R) / 2;
    if (a[m] < x)
      L = m + 1;
    else
      R = m;
  }
  return (x == a[R]) ? R : -1;
}
int main(void)
{ const int m = 8;
  int v[m+1] ={8,12,19,43,45,56,67,95};

  cout << "Busca de 19 = " << busca_binaria_rapida(19,m,v) << endl;
  cout << "Busca de 45 = " << busca_binaria_rapida(45,m,v) << endl;
  cout << "Busca de 8  = " << busca_binaria_rapida(8,m,v) << endl;
  cout << "Busca de 81 = " << busca_binaria_rapida(81,m,v) << endl;
  return 0;
}
```

```
Busca de 19 = 5
Busca de 45 = 0
Busca de 8  = 6
Busca de 81 = -1
```

Análise da Busca Binária Rápida

- Em média são efetuadas $^{+}$ **$\log_2(N)-1^{+}$** comparações de chaves para encontrar um elemento particular **x** no vetor **a** de **N** elementos
- O pior caso requerer $^{+}$ **$\log_2 N$** $^{+}$comparações

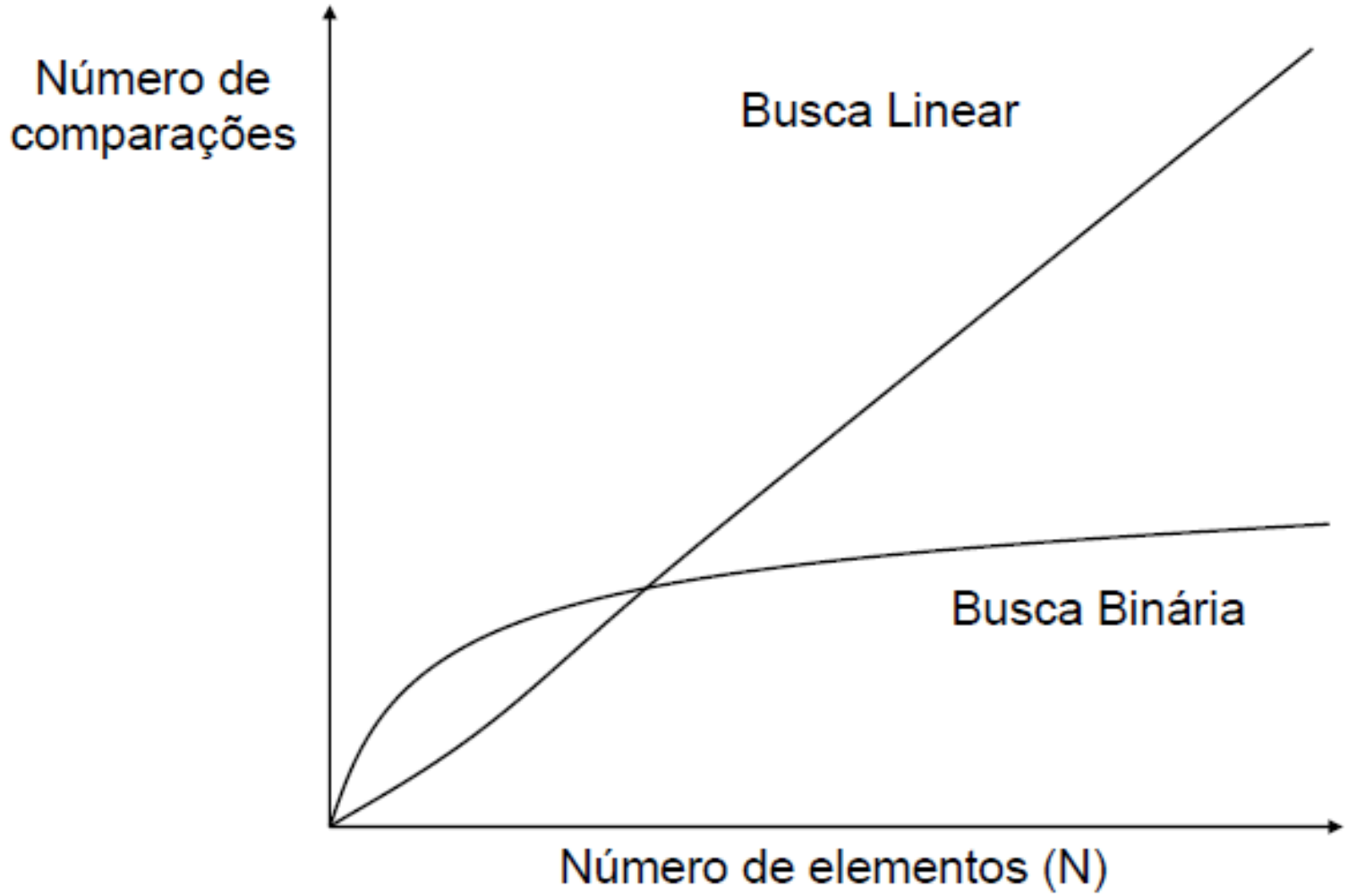
N	Nº Comparações (pior caso)
8	3
128	7
1.024	10
32.768	15
1.048.576	20
1.073.741.824	30
1.099.511.627.776	40
10^{80}	266

Comparação

- Considerando os algoritmos de busca vistos, a tabela seguinte mostra a ordem de grandeza dos números mínimo ($C_{mín}$), médio ($C_{méd}$) e máximo ($C_{máx}$) de comparações de chaves.

Algoritmo	$C_{mín}$	$C_{méd}$	$C_{máx}$
Busca Linear	O(1)	O(N)	O(N)
Busca Linear com Sentinela	O(1)	O(N)	O(N)
Busca Binária	O(1)	$O(\log_2 N)$	$O(\log_2 N)$
Busca Binária Rápida	$O(\log_2 N)$	$O(\log_2 N)$	$O(\log_2 N)$

Comparação


Número de comparações
Busca Linear
Busca Binária
Número de elementos (N)

Resumo

- Das análises dos algoritmos de busca, está claro que o método de busca binária tem um desempenho tão bom ou melhor do que o método de busca linear
- Entretanto, a atualização dos índices esquerdo, direito e médio (**L**, **R** e **m** no algoritmo, respectivamente) requer tempo adicional
- Assim, para vetores com poucos elementos, a busca linear é adequada
- Para vetores com muitos elementos, a busca binária é mais eficiente, mas isso requer que o vetor esteja ordenado

Exemplo em C++

DESAFIO ORDENAÇÃO PARES E IMPARES

Seja uma sequência de números inteiros (distintos) armazenadas em um array A[i], $1 \leq i \leq n$. Apresentar um programa para realizar a separação dos elementos pares e ímpares e armazená-los em dois novos arrays (par/impar). Esse programa deve apresentar, no mínimo, um menu com as seguintes opções:

1) Inserir elementos no array;
2) Separar elementos pares e ímpares
3) Apresentar arrays;
3) Ordenar elementos e apresentar arrays;
4) Sair do programa.

O programa deve estar orientado a objetos, utilizando métodos para realização de cada um dos itens do menu.

ALGORITMOS E ESTRUTURAS DE DADOS

Busca Linear e Binária