Aula 28 – Busca Sequencial e Binária

Norton T. Roman & Luciano A. Digiampietri

Busca em Arranjo

- Suponha que temos um arranjo de inteiros
- Como fazemos para verificar se um determinado número está lá?

Busca em Arranjo

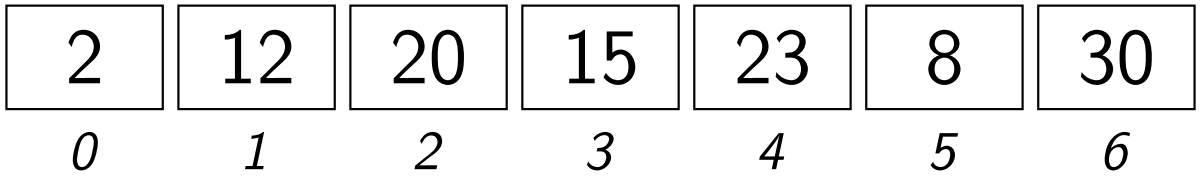
- Suponha que temos um arranjo de inteiros
- Como fazemos para verificar se um determinado número está lá?
  - Varremos o arranjo, da esquerda para a direita
  - Se acharmos o número, então ele está no arranjo
  - Se chegarmos ao final do arranjo e não acharmos, ele não está

Busca em Arranjo

- Suponha que temos um arranjo de inteiros
- Como fazemos para verificar se um determinado número está lá?
  - Varremos o arranjo, da esquerda para a direita
  - Se acharmos o número, então ele está no arranjo
  - Se chegarmos ao final do arranjo e não acharmos, ele não está
- Busca Sequencial!

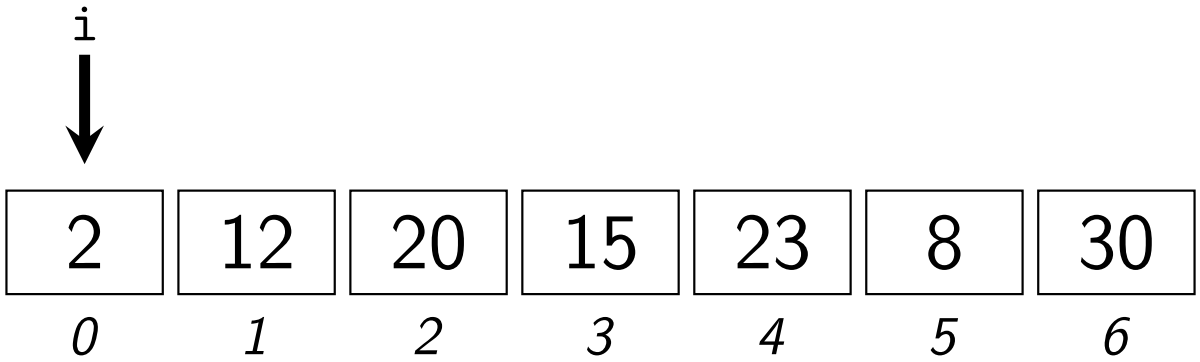
Busca Sequencial

- Ex: Buscando 15



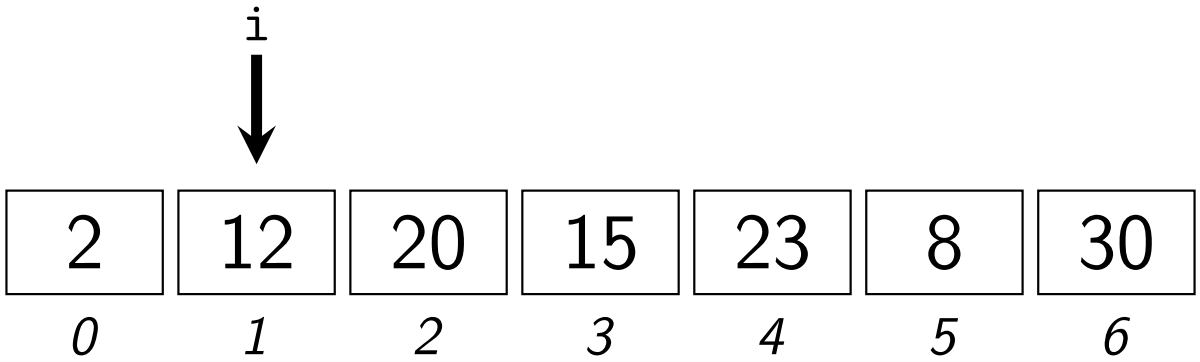
Busca Sequencial

- Ex: Buscando 15



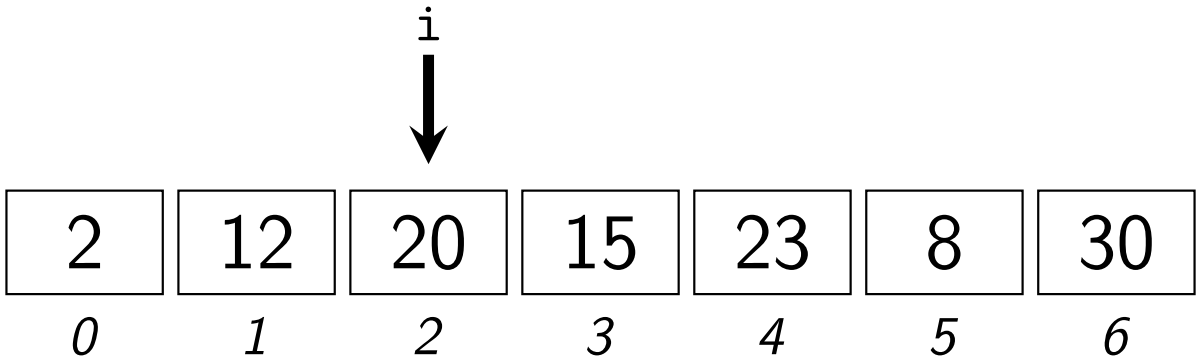
Busca Sequencial

- Ex: Buscando 15



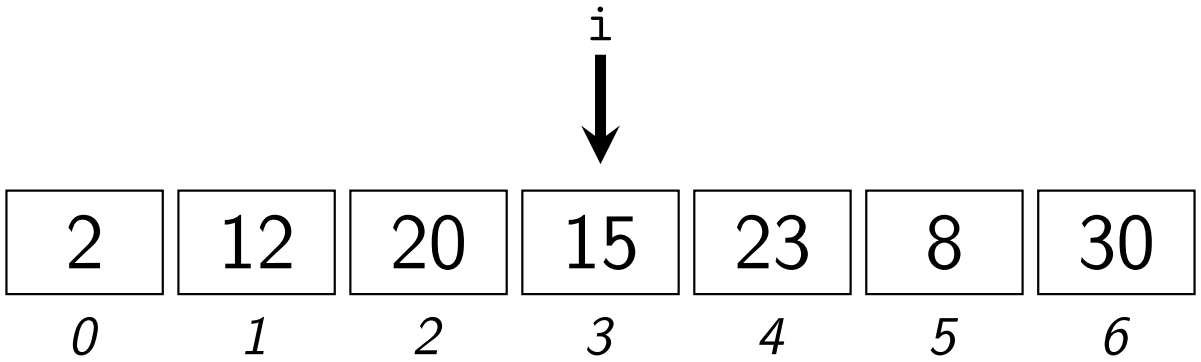
Busca Sequencial

- Ex: Buscando 15



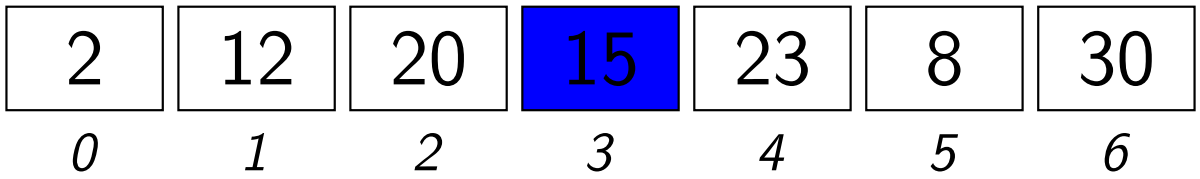
Busca Sequencial

- Ex: Buscando 15



Busca Sequencial

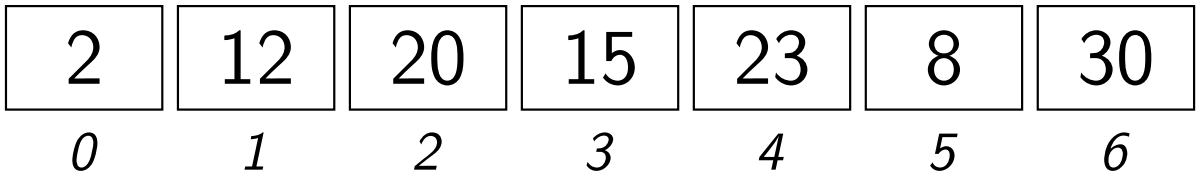
- Ex: Buscando 15



Encontrou! Índice 3 (quarta posição).

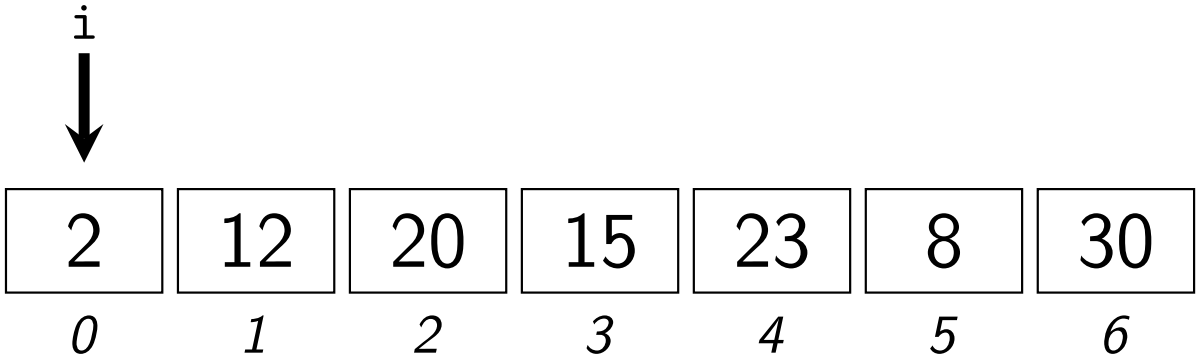
Busca Sequencial

- E o 16?



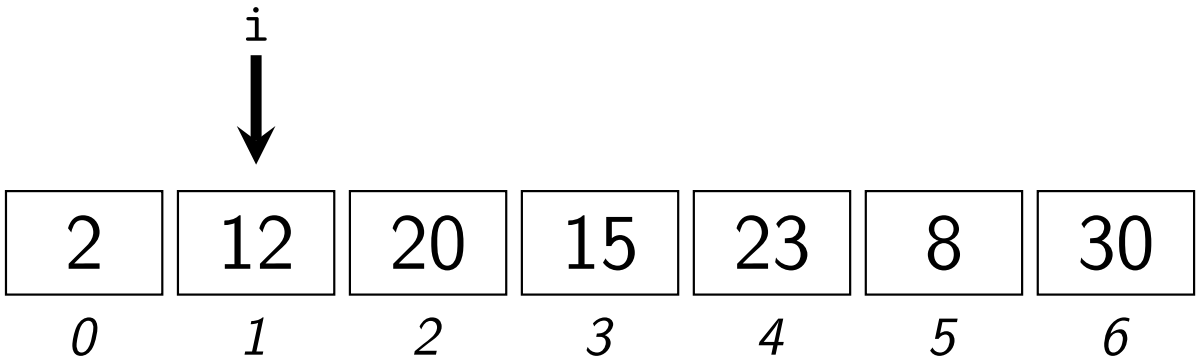
Busca Sequencial

- E o 16?



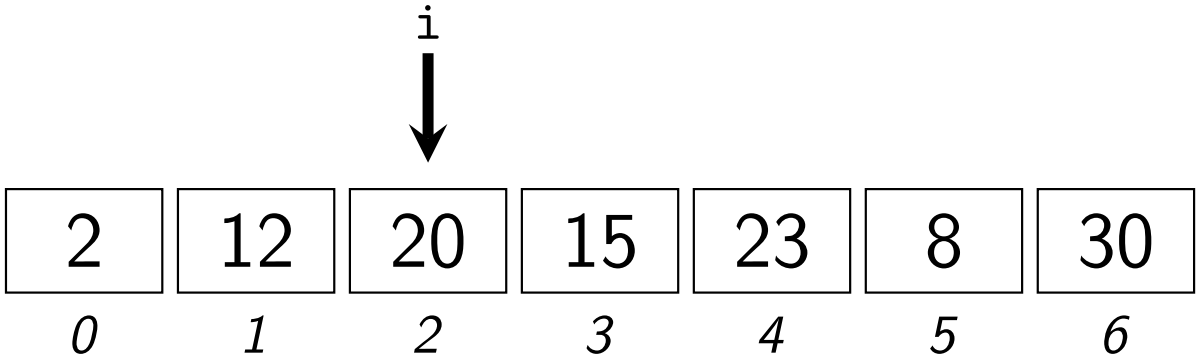
Busca Sequencial

- E o 16?



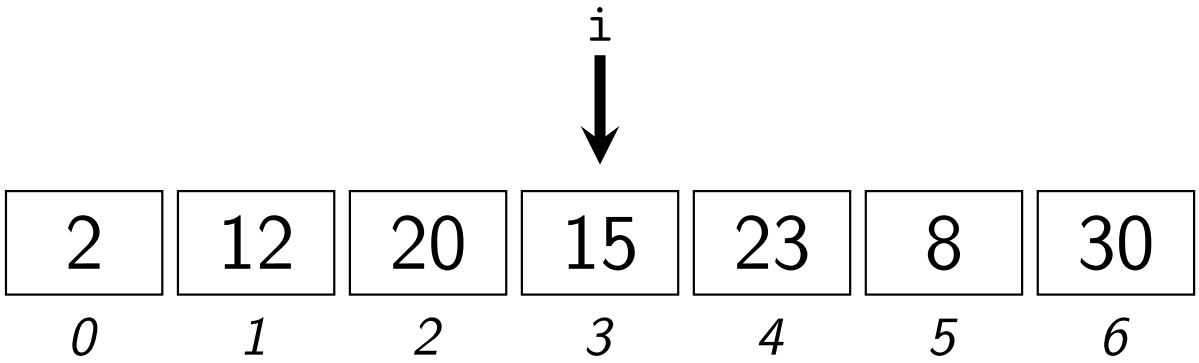
Busca Sequencial

- E o 16?



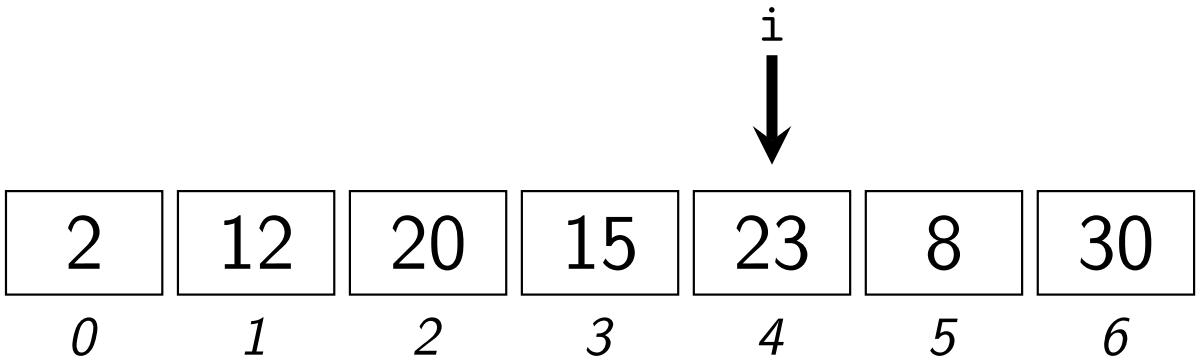
Busca Sequencial

- E o 16?



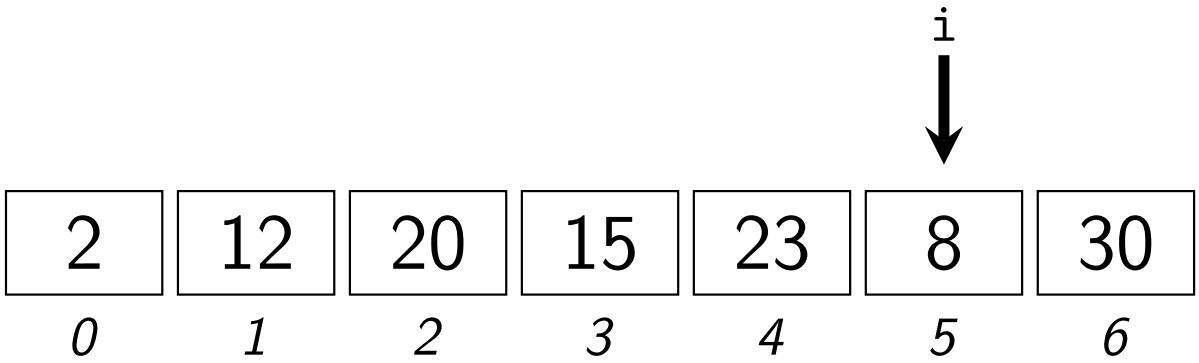
Busca Sequencial

- E o 16?



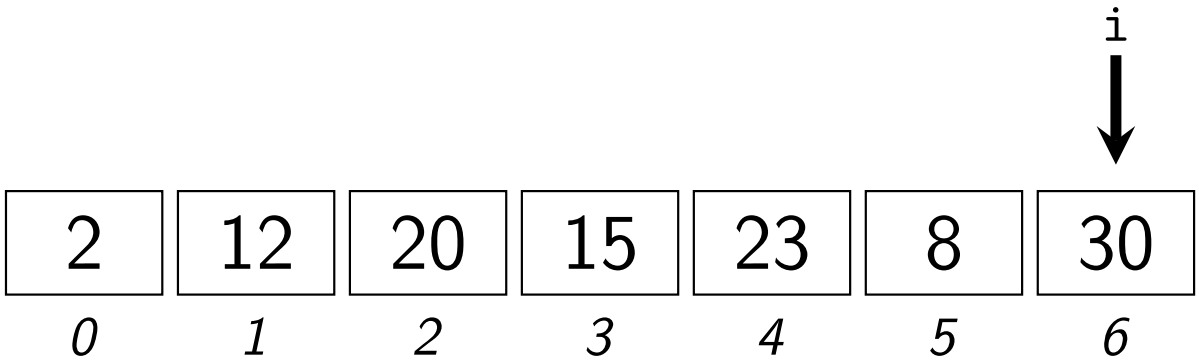
Busca Sequencial

- E o 16?



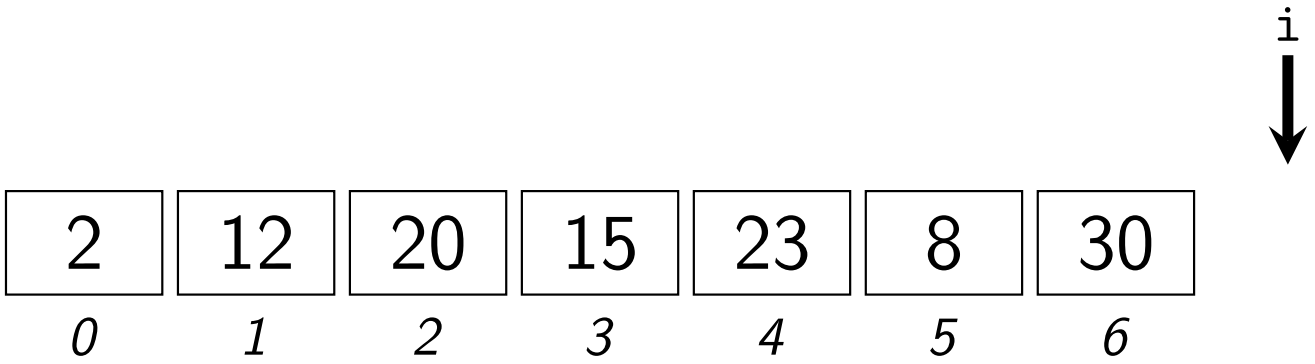
Busca Sequencial

- E o 16?



Busca Sequencial

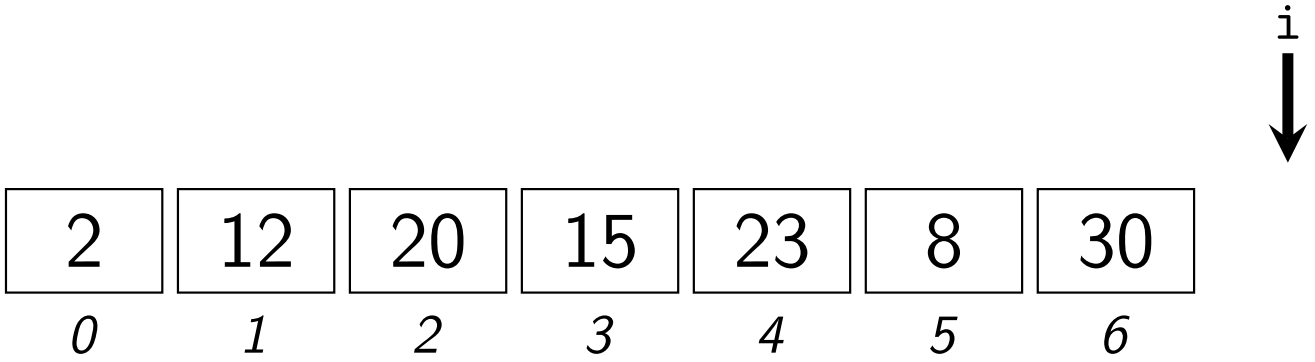
- E o 16?



Não encontrou! Como sabemos?

Busca Sequencial

- E o 16?



Não encontrou! Como sabemos?

`i == 7`, ou seja, `i == tamanho`.

Busca Sequencial

- Note como a função foi implementada

```
int buscaSeq(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++)
    if (arr[i] == el) return i;
  return -1;
}

int main() {
  int v[] = {9, 8, 4, 6, 3, 4};
  printf("%i\n", buscaSeq(v, 6, 4));
  printf("%i\n", buscaSeq(v, 6, 12));
  return 0;
}
```

Busca Sequencial

- Note como a função foi implementada

```
int buscaSeq(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++)
    if (arr[i] == el) return i;
  return -1;
}

int main() {
  int v[] = {9, 8, 4, 6, 3, 4};
  printf("%i\n", buscaSeq(v, 6, 4));
  printf("%i\n", buscaSeq(v, 6, 12));
  return 0;
}
```

Busca Sequencial

- Note como a função foi implementada
  - Ela já dispõe, em seus parâmetros, de toda a informação de que precisa para executar

```
int buscaSeq(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++)
    if (arr[i] == el) return i;
  return -1;
}

int main() {
  int v[] = {9, 8, 4, 6, 3, 4};
  printf("%i\n", buscaSeq(v, 6, 4));
  printf("%i\n", buscaSeq(v, 6, 12));
  return 0;
}
```

Busca Sequencial

- Note como a função foi implementada
  - Ela já dispõe, em seus parâmetros, de toda a informação de que precisa para executar
    - Do arranjo propriamente dito, de seu tamanho e do valor que está sendo buscado

```
int buscaSeq(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++)
    if (arr[i] == el) return i;
  return -1;
}

int main() {
  int v[] = {9, 8, 4, 6, 3, 4};
  printf("%i\n", buscaSeq(v, 6, 4));
  printf("%i\n", buscaSeq(v, 6, 12));
  return 0;
}
```

Busca Sequencial

- Note como a função foi implementada
  - Ela já dispõe, em seus parâmetros, de toda a informação de que precisa para executar
    - Do arranjo propriamente dito, de seu tamanho e do valor que está sendo buscado

```
int buscaSeq(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++)
    if (arr[i] == el) return i;
  return -1;
}

int main() {
  int v[] = {9, 8, 4, 6, 3, 4};
  printf("%i\n", buscaSeq(v, 6, 4));
  printf("%i\n", buscaSeq(v, 6, 12));
  return 0;
}
```

Busca Sequencial

- Note como a função foi implementada
  - Ela já dispõe, em seus parâmetros, de toda a informação de que precisa para executar
    - Do arranjo propriamente dito, de seu tamanho e do valor que está sendo buscado

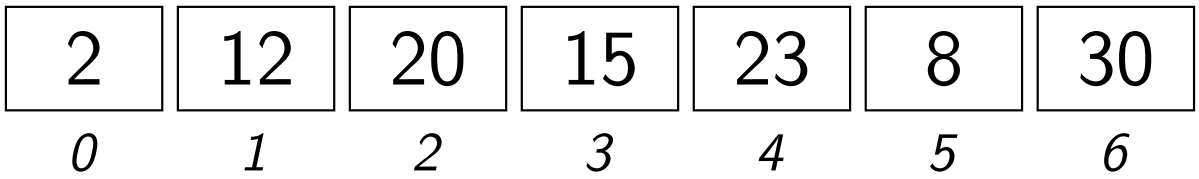
```
int buscaSeq(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++)
    if (arr[i] == el) return i;
  return -1;
}

int main() {
  int v[] = {9, 8, 4, 6, 3, 4};
  printf("%i\n", buscaSeq(v, 6, 4));
  printf("%i\n", buscaSeq(v, 6, 12));
  return 0;
}
```

Saída

```
2
-1
```

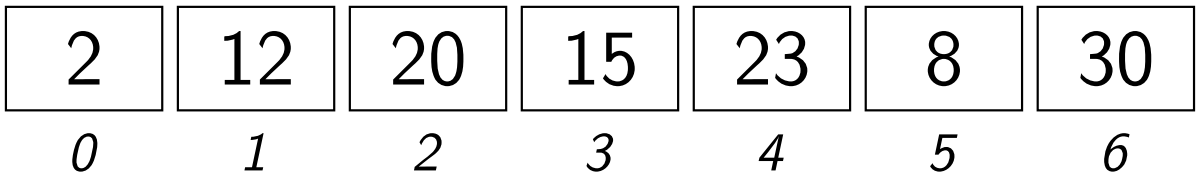
Busca Sequencial

- O que podemos fazer para melhorar a busca por 9?



Busca Sequencial

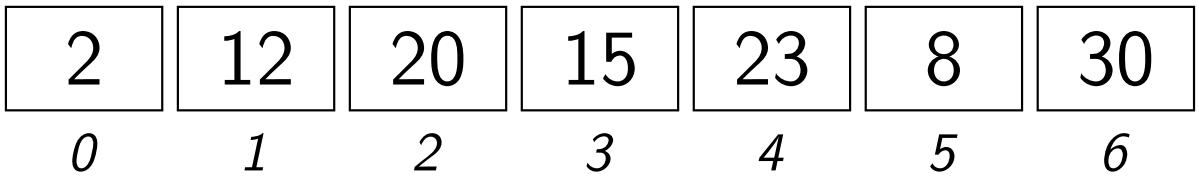
- O que podemos fazer para melhorar a busca por 9?



Nada, realmente...

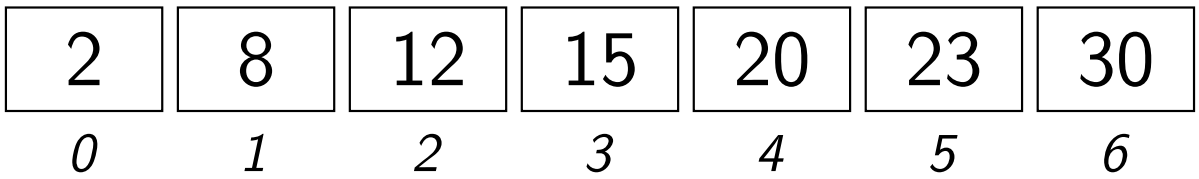
Busca Sequencial

- O que podemos fazer para melhorar a busca por 9?



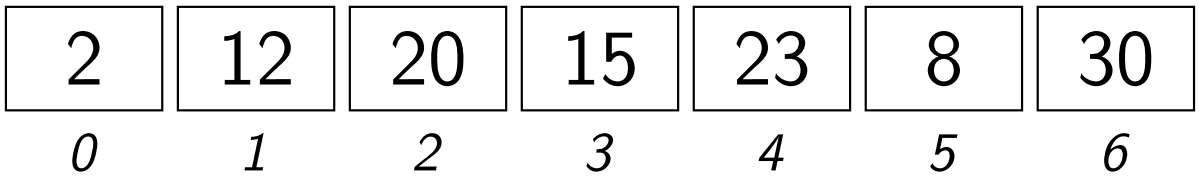
Nada, realmente...

- E se o arranjo estivesse ordenado?



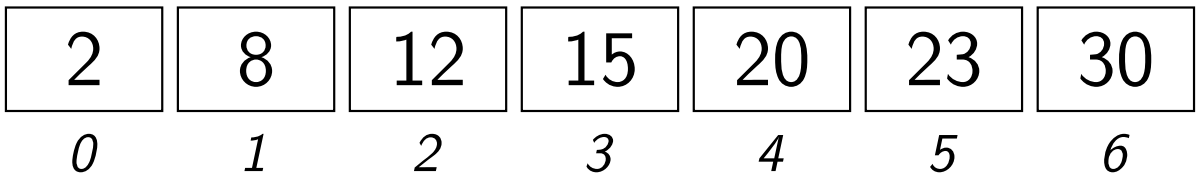
Busca Sequencial

- O que podemos fazer para melhorar a busca por 9?



Nada, realmente...

- E se o arranjo estivesse ordenado?



  - Poderíamos parar a busca assim que encontrássemos um número maior que ele

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}


int main() {
  int v[] = {-78,-4,0,32,52,55,63,69,125,200};
  printf("%i\n", buscaSeq2(v, 10, 23));
  printf("%i\n", buscaSeq2(v, 10, 8));
  return 0;
}
```

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}


int main() {
  int v[] = {-78,-4,0,32,52,55,63,69,125,200};
  printf("%i\n", buscaSeq2(v, 10, 23));
  printf("%i\n", buscaSeq2(v, 10, 8));
  return 0;
}
```

- Com o arranjo ordenado, potencialmente executamos menos comparações no caso do elemento não estar no arranjo

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}


int main() {
  int v[] = {-78,-4,0,32,52,55,63,69,125,200};
  printf("%i\n", buscaSeq2(v, 10, 23));
  printf("%i\n", buscaSeq2(v, 10, 8));
  return 0;
}
```

- Com o arranjo ordenado, potencialmente executamos menos comparações no caso do elemento não estar no arranjo

- Estar ordenado também torna fácil algumas tarefas:

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}


int main() {
  int v[] = {-78,-4,0,32,52,55,63,69,125,200};
  printf("%i\n", buscaSeq2(v, 10, 23));
  printf("%i\n", buscaSeq2(v, 10, 8));
  return 0;
}
```

- Com o arranjo ordenado, potencialmente executamos menos comparações no caso do elemento não estar no arranjo

- Estar ordenado também torna fácil algumas tarefas:
  - Busca pelo menor elemento:

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}


int main() {
  int v[] = {-78,-4,0,32,52,55,63,69,125,200};
  printf("%i\n", buscaSeq2(v, 10, 23));
  printf("%i\n", buscaSeq2(v, 10, 8));
  return 0;
}
```

- Com o arranjo ordenado, potencialmente executamos menos comparações no caso do elemento não estar no arranjo

- Estar ordenado também torna fácil algumas tarefas:
  - Busca pelo menor elemento: `v[0]`

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}


int main() {
  int v[] = {-78,-4,0,32,52,55,63,69,125,200};
  printf("%i\n", buscaSeq2(v, 10, 23));
  printf("%i\n", buscaSeq2(v, 10, 8));
  return 0;
}
```

- Com o arranjo ordenado, potencialmente executamos menos comparações no caso do elemento não estar no arranjo

- Estar ordenado também torna fácil algumas tarefas:
  - Busca pelo menor elemento: `v[0]`
  - Busca pelo maior elemento:

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}


int main() {
  int v[] = {-78,-4,0,32,52,55,63,69,125,200};
  printf("%i\n", buscaSeq2(v, 10, 23));
  printf("%i\n", buscaSeq2(v, 10, 8));
  return 0;
}
```

- Com o arranjo ordenado, potencialmente executamos menos comparações no caso do elemento não estar no arranjo

- Estar ordenado também torna fácil algumas tarefas:
  - Busca pelo menor elemento: `v[0]`
  - Busca pelo maior elemento: `v[tamanho-1]`

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}
```

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}
```

- A cada iteração estamos verificando duas condições

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}
```

- A cada iteração estamos verificando duas condições
- Conseguimos fazer melhor?

Busca em Arranjo Ordenado

```
int buscaSeq2(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] == el) return i;
    if (arr[i] > el) break;
  }
  return -1;
}
```

```
int buscaSeq3(int arr[], int tam, int el) {
  int i;
  for (i=0; i<tam; i++) {
    if (arr[i] >= el) {
        if (arr[i] == el) return i;
        else return -1;
    }
  }
  return -1;
}
```

- A cada iteração estamos verificando duas condições
- Conseguimos fazer melhor?
- Sim: uma única condição (condição adicional apenas quando a primeira condição for verdadeira [uma única vez durante a execução da função])

Busca em Arranjo Ordenado

- Vimos que se o arranjo estiver ordenado, buscas ficam em geral mais rápidas

Busca em Arranjo Ordenado

- Vimos que se o arranjo estiver ordenado, buscas ficam em geral mais rápidas
- Paramos a busca assim que uma das condições for satisfeita:

Busca em Arranjo Ordenado

- Vimos que se o arranjo estiver ordenado, buscas ficam em geral mais rápidas
- Paramos a busca assim que uma das condições for satisfeita:
  - Encontramos o elemento buscado

Busca em Arranjo Ordenado

- Vimos que se o arranjo estiver ordenado, buscas ficam em geral mais rápidas
- Paramos a busca assim que uma das condições for satisfeita:
  - Encontramos o elemento buscado
  - Chegamos ao fim do arranjo

Busca em Arranjo Ordenado

- Vimos que se o arranjo estiver ordenado, buscas ficam em geral mais rápidas
- Paramos a busca assim que uma das condições for satisfeita:
  - Encontramos o elemento buscado
  - Chegamos ao fim do arranjo
  - (Diferencial!) Encontramos um elemento maior que o buscado

Busca em Arranjo Ordenado

- Ainda assim, no pior caso, teremos que olhar o arranjo inteiro, quando:

Busca em Arranjo Ordenado

- Ainda assim, no pior caso, teremos que olhar o arranjo inteiro, quando:
  - O elemento buscado for o último

Busca em Arranjo Ordenado

- Ainda assim, no pior caso, teremos que olhar o arranjo inteiro, quando:
  - O elemento buscado for o último
  - O elemento buscado não estiver no arranjo, mas for maior que o último

Busca em Arranjo Ordenado

- Ainda assim, no pior caso, teremos que olhar o arranjo inteiro, quando:
  - O elemento buscado for o último
  - O elemento buscado não estiver no arranjo, mas for maior que o último
- Não teria um modo melhor?

Busca Binária

- Algoritmo:

Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo

Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior

Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo

Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo

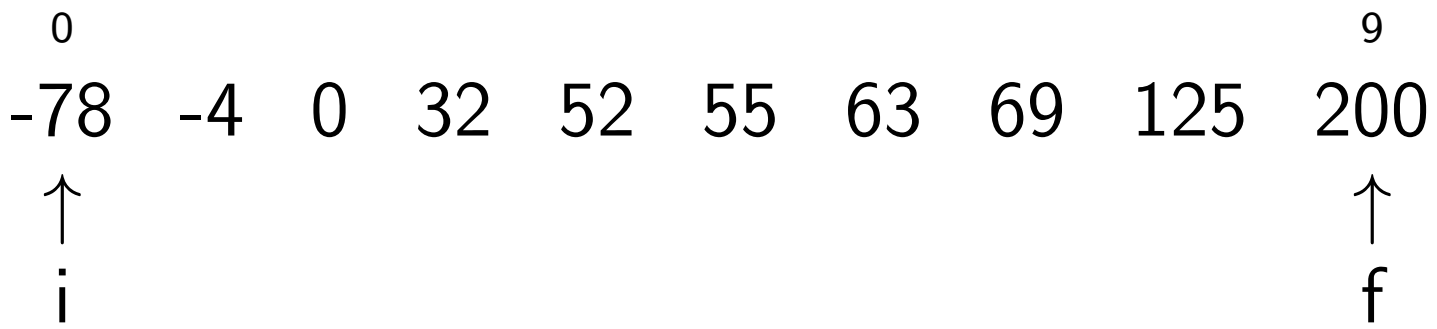
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 52

0									9
-78	-4	0	32	52	55	63	69	125	200

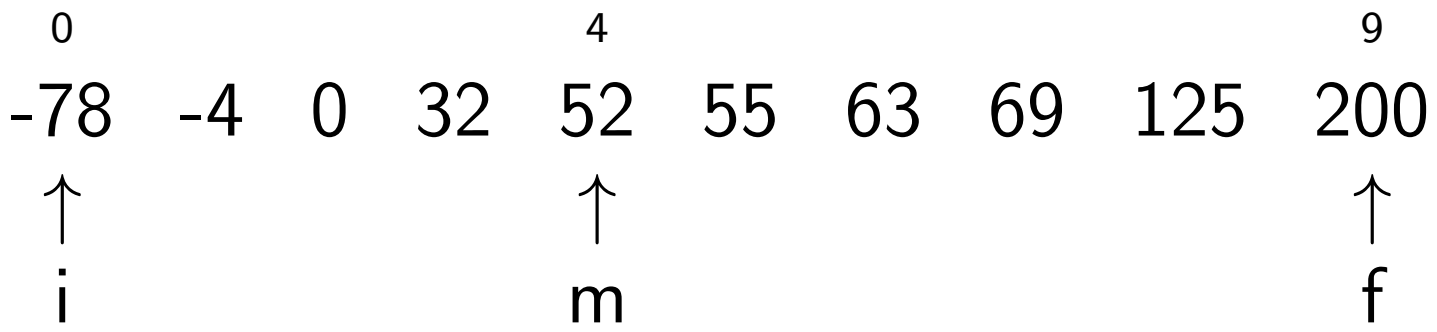
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 52



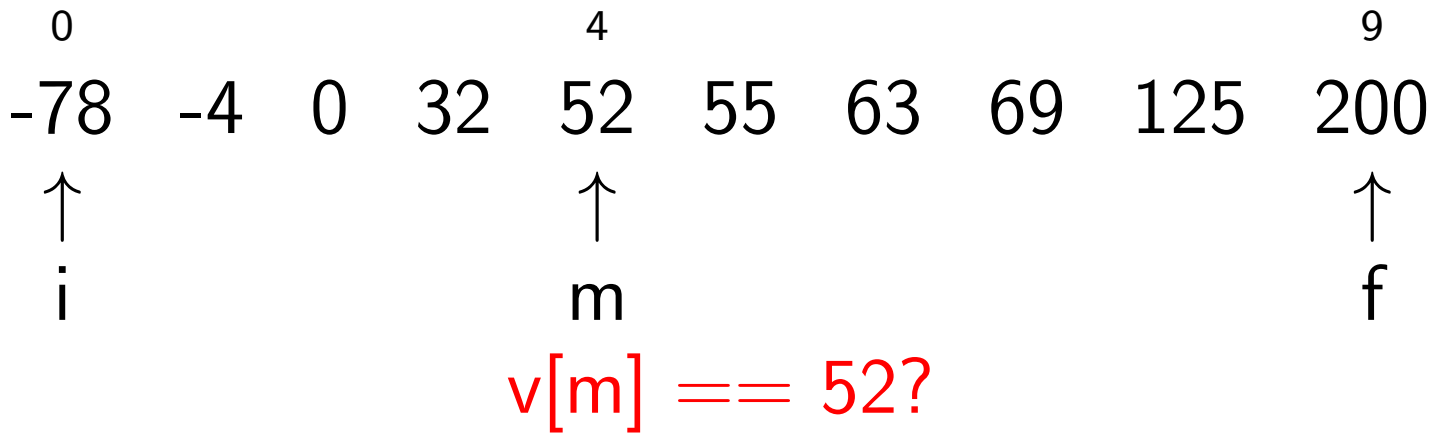
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 52



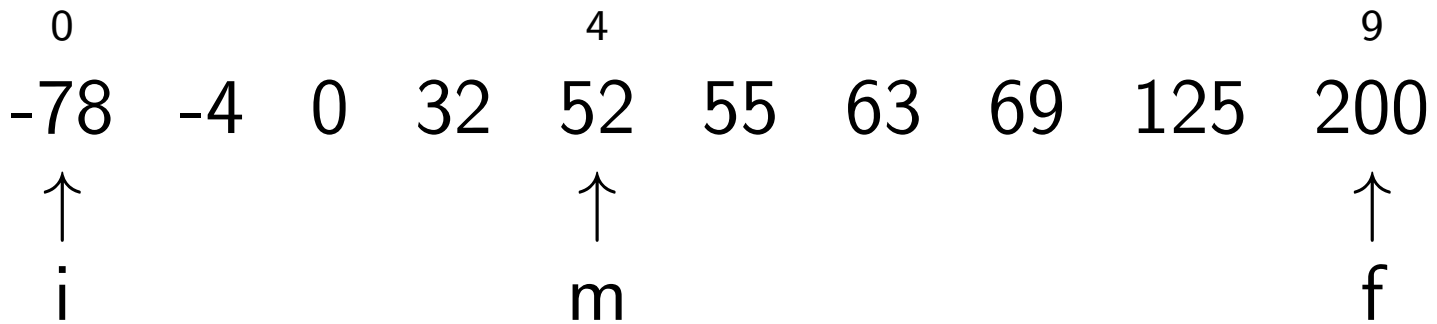
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 52



Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 52



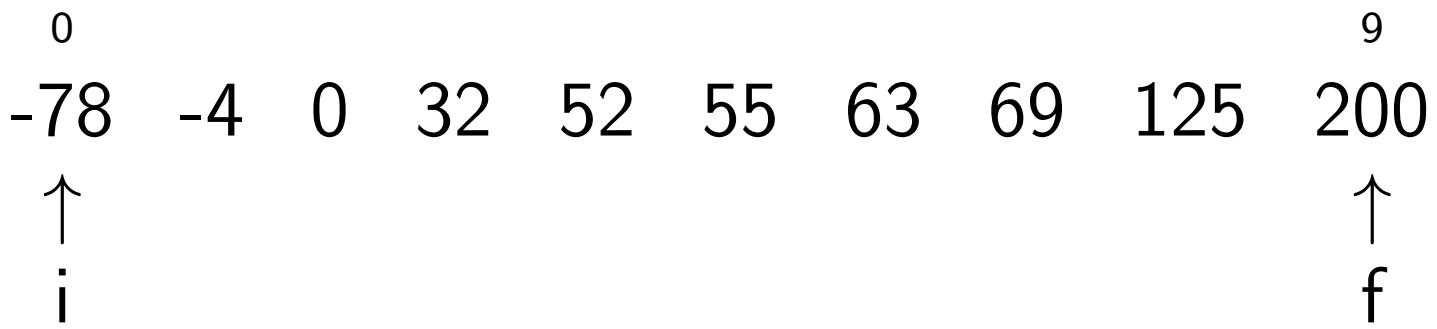
v[m] == 52? Achou com apenas 1 acesso

Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55

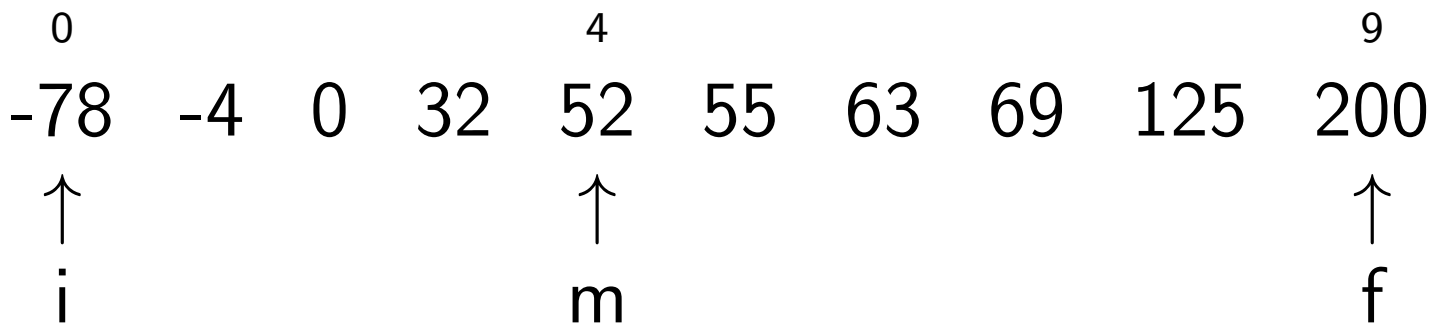
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



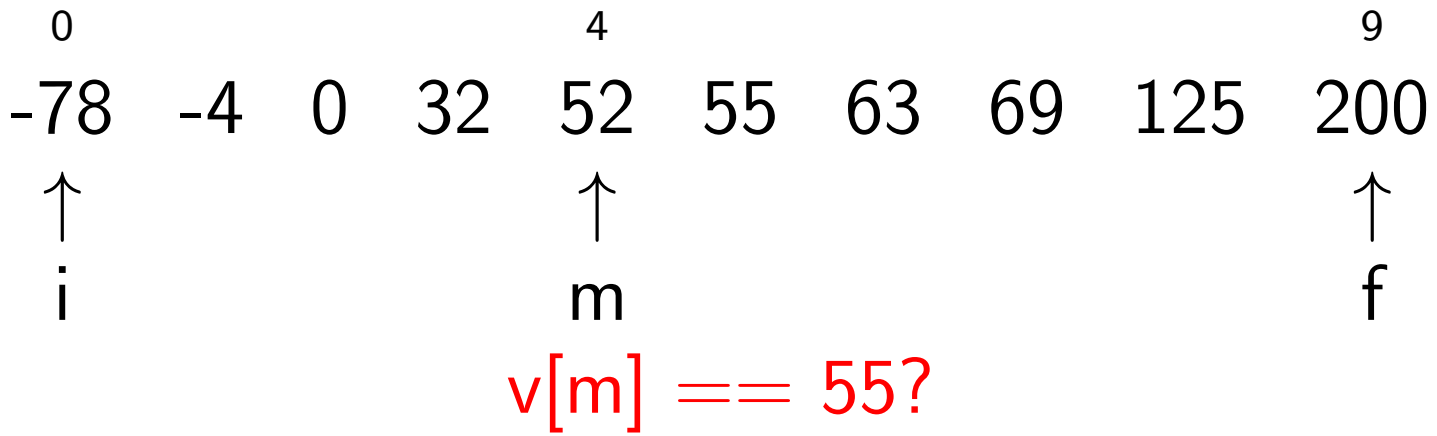
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



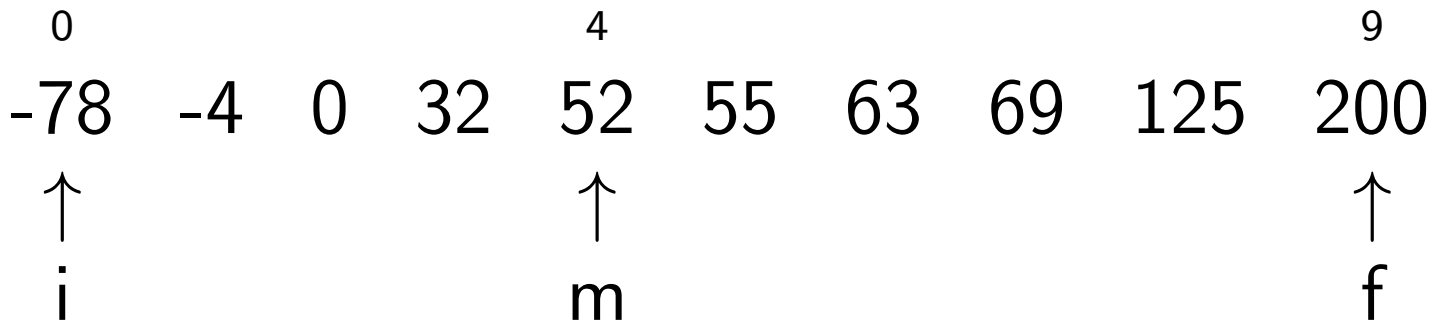
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



Busca Binária

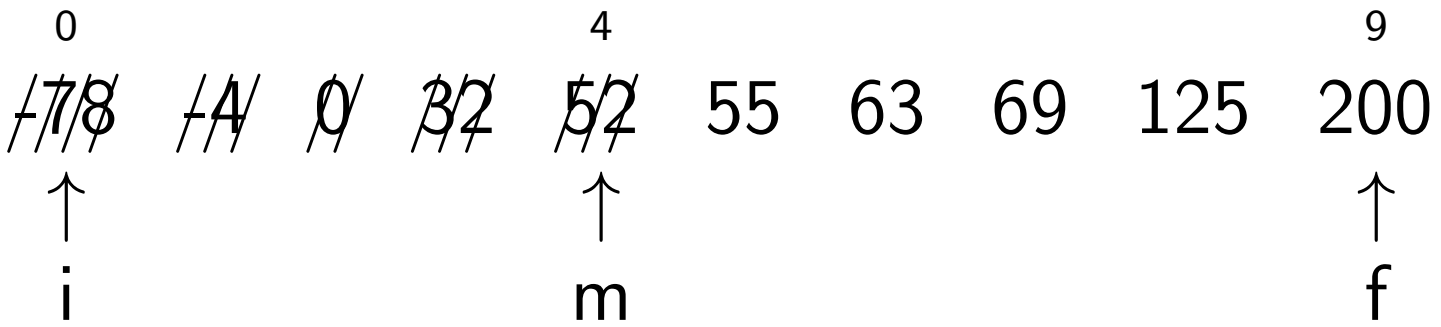
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55?

Busca Binária

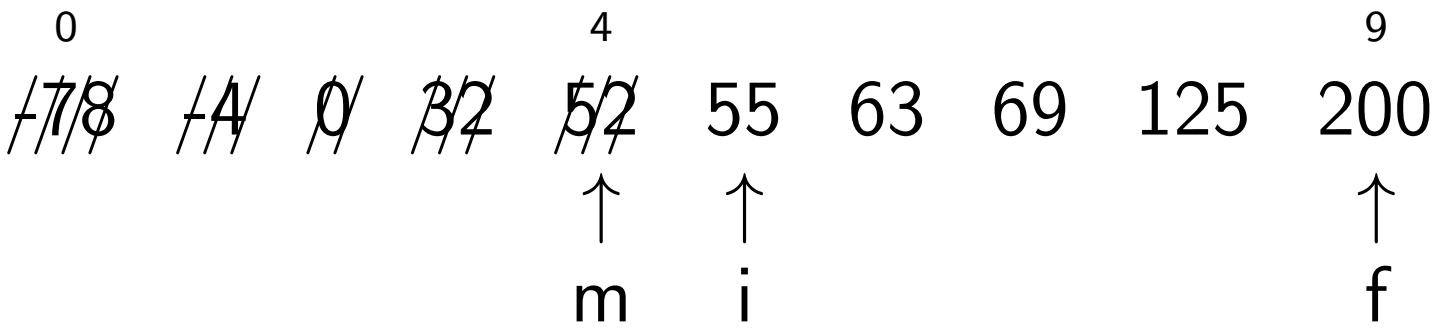
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55? Sim.

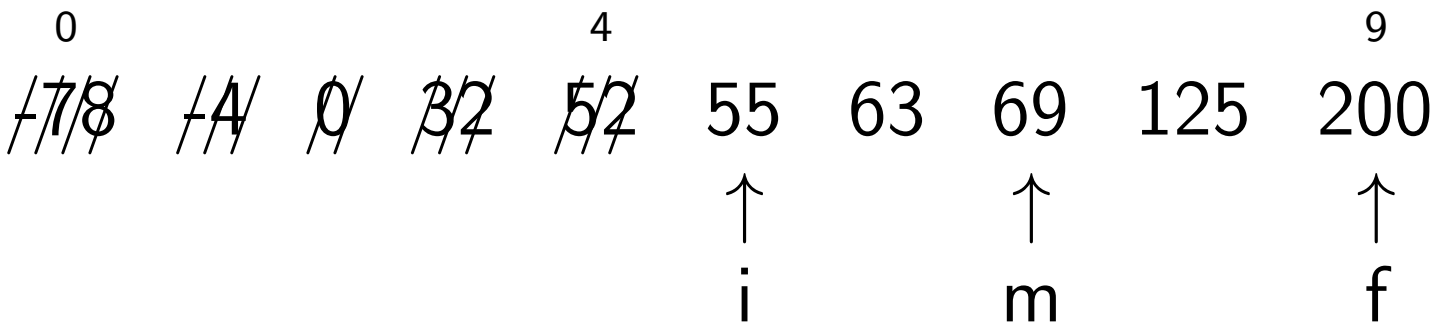
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



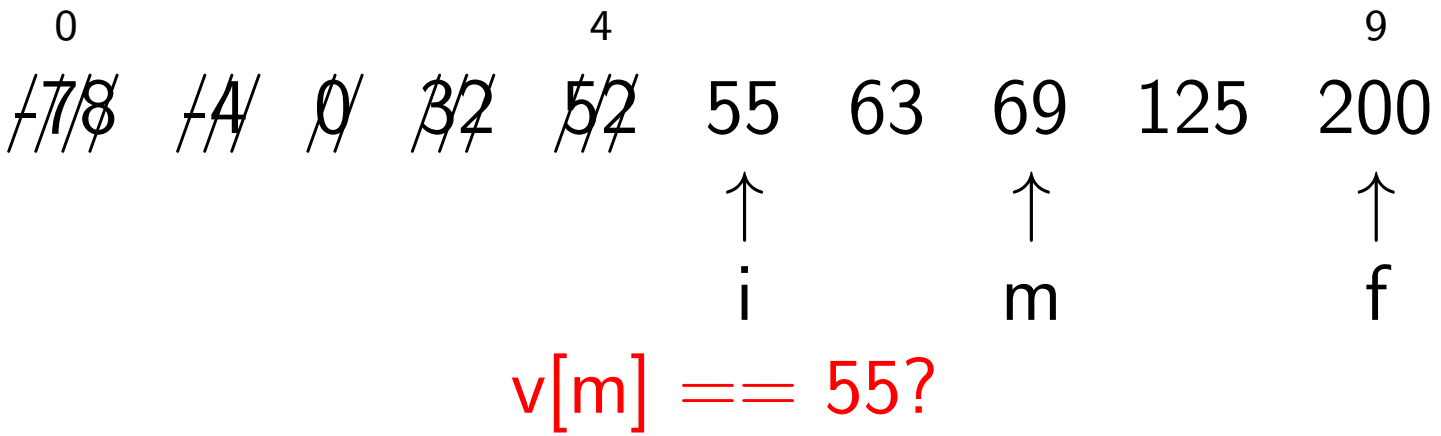
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



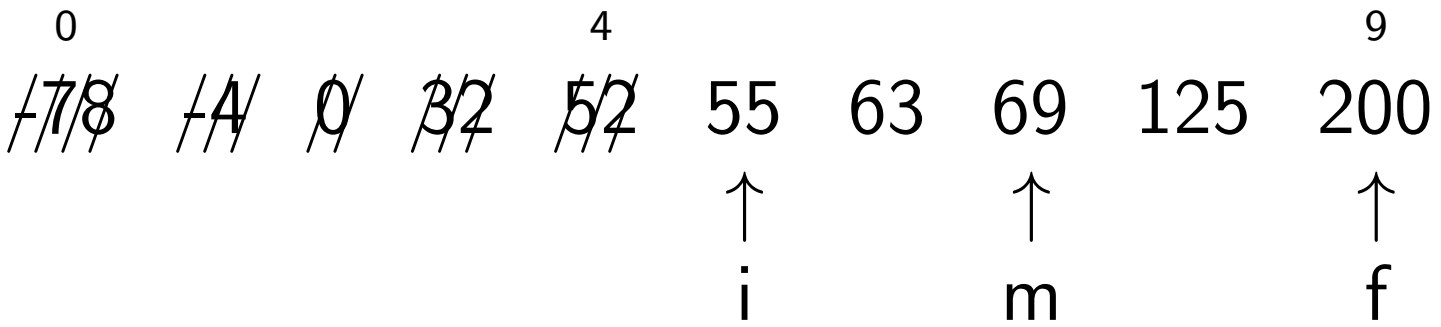
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



Busca Binária

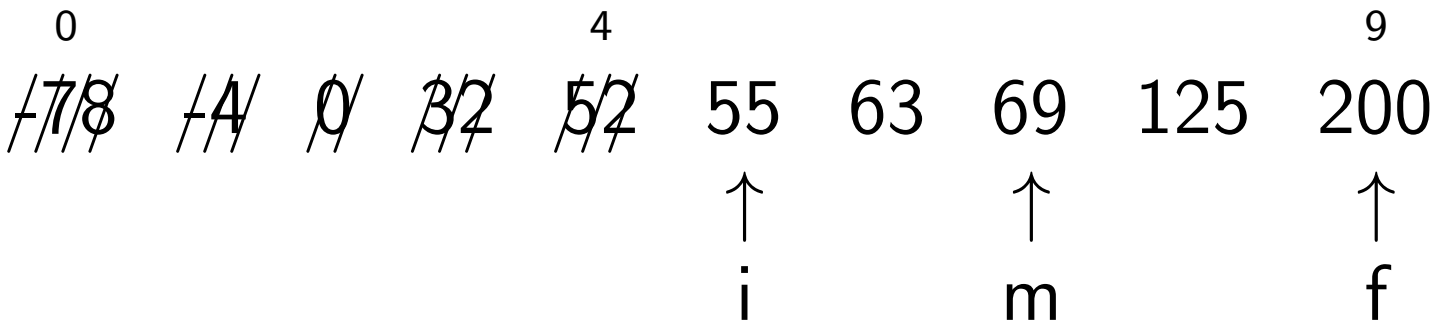
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55?

Busca Binária

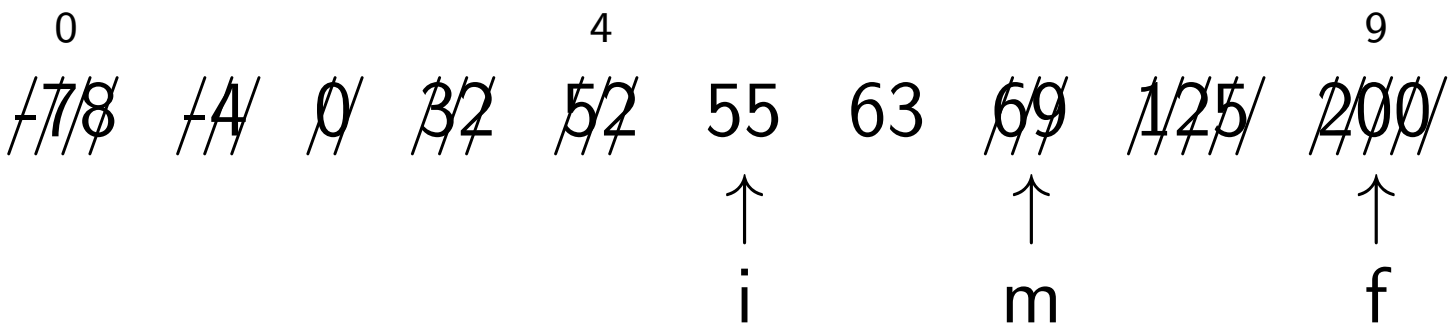
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55? Não.

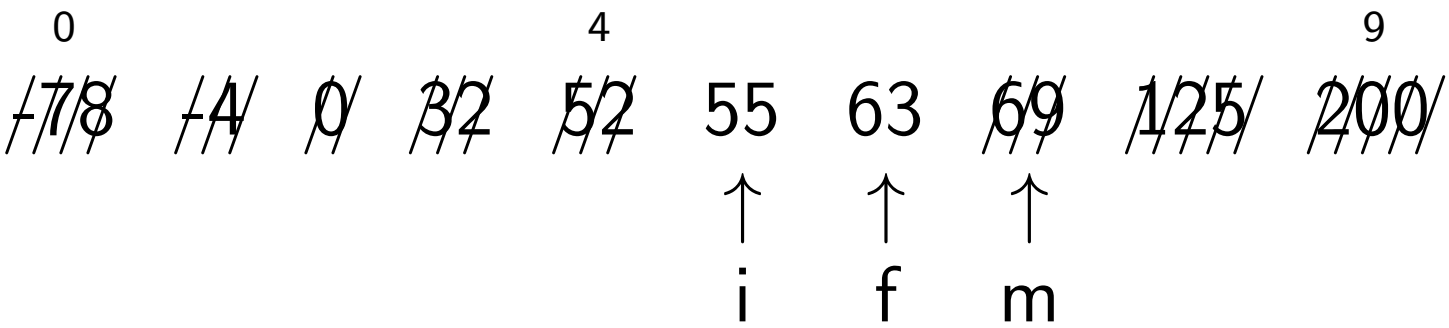
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



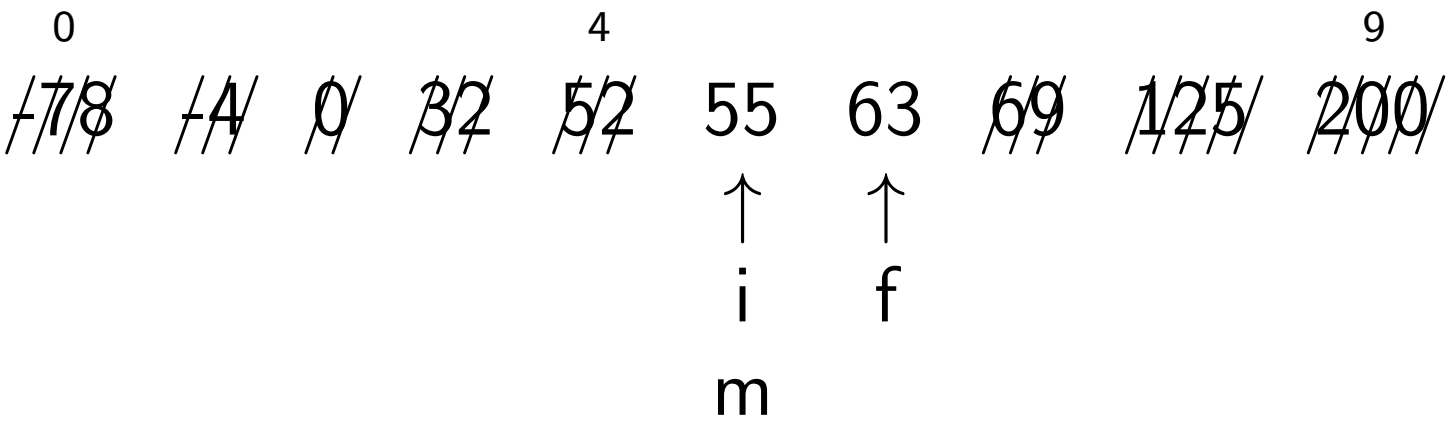
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



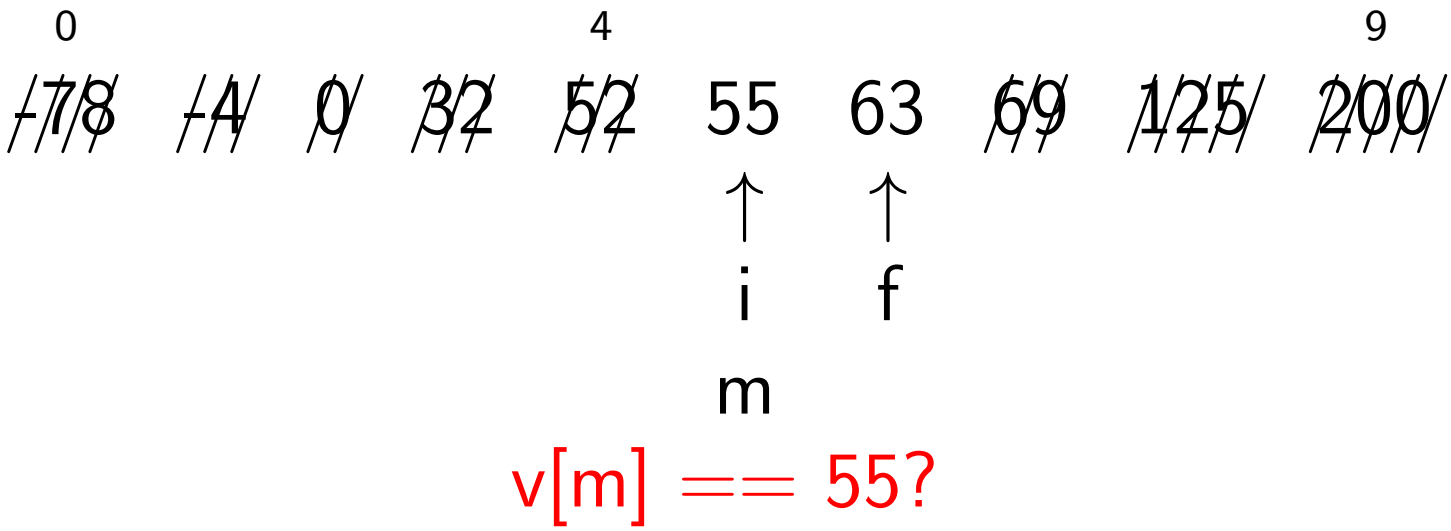
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



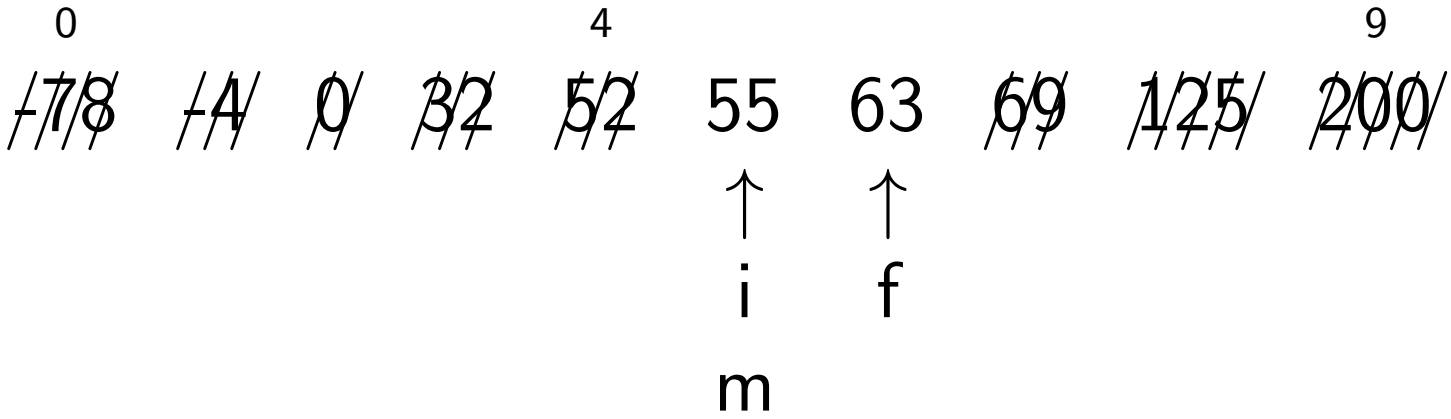
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



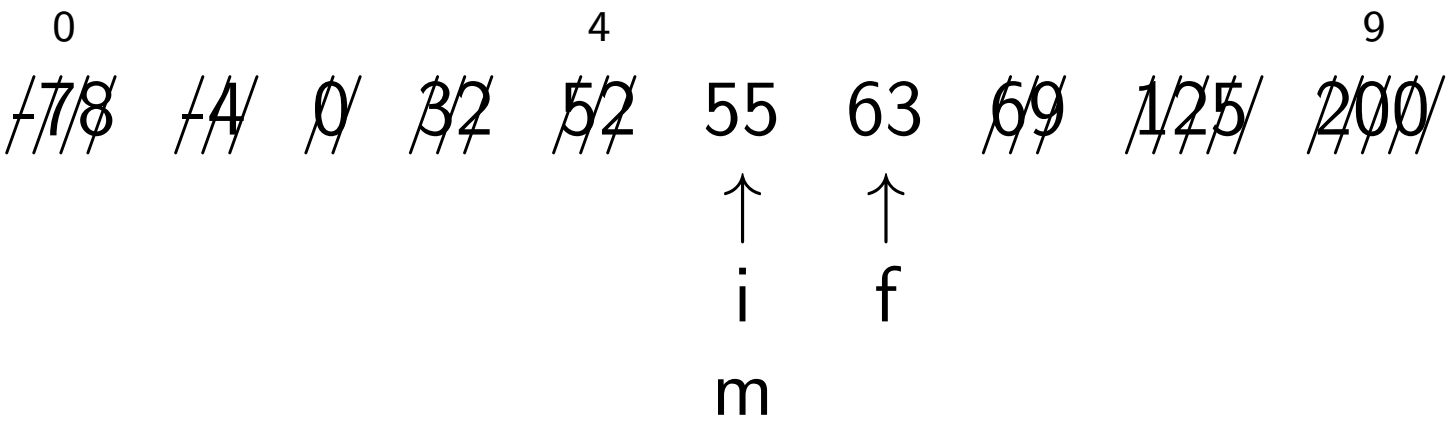
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



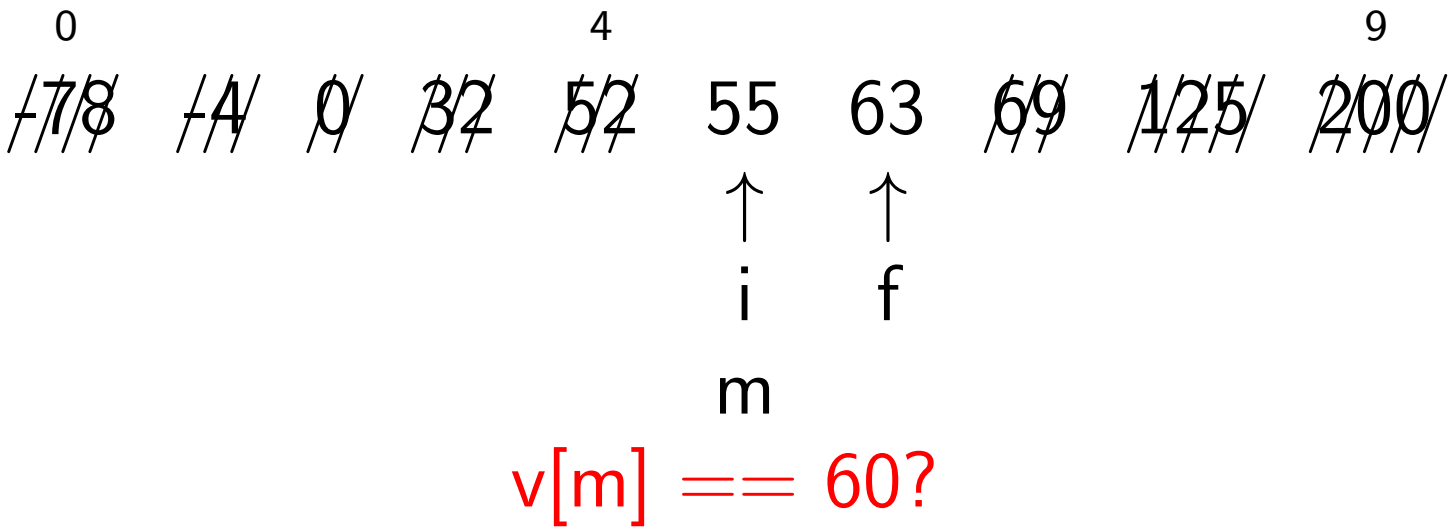
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



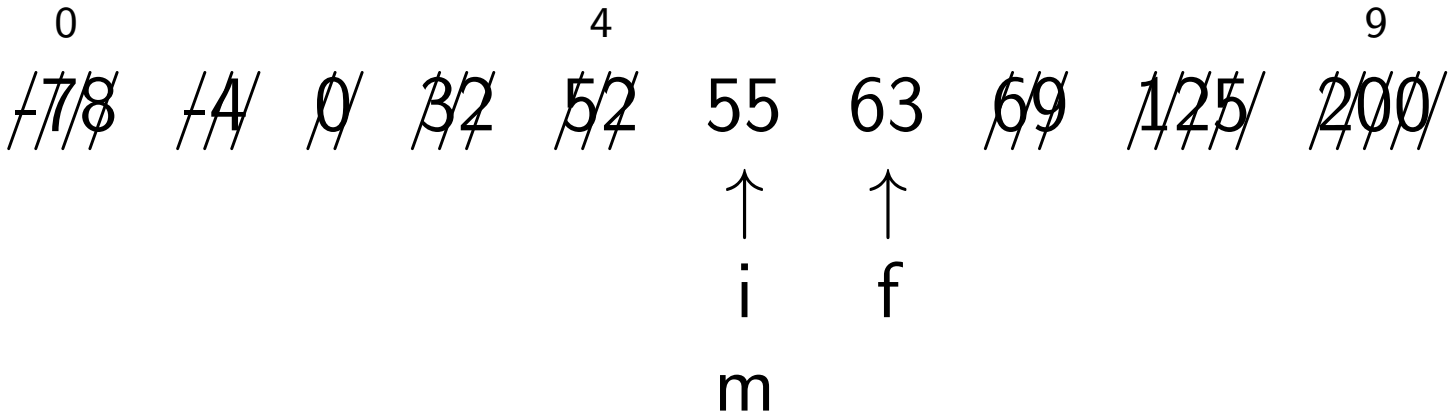
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



Busca Binária

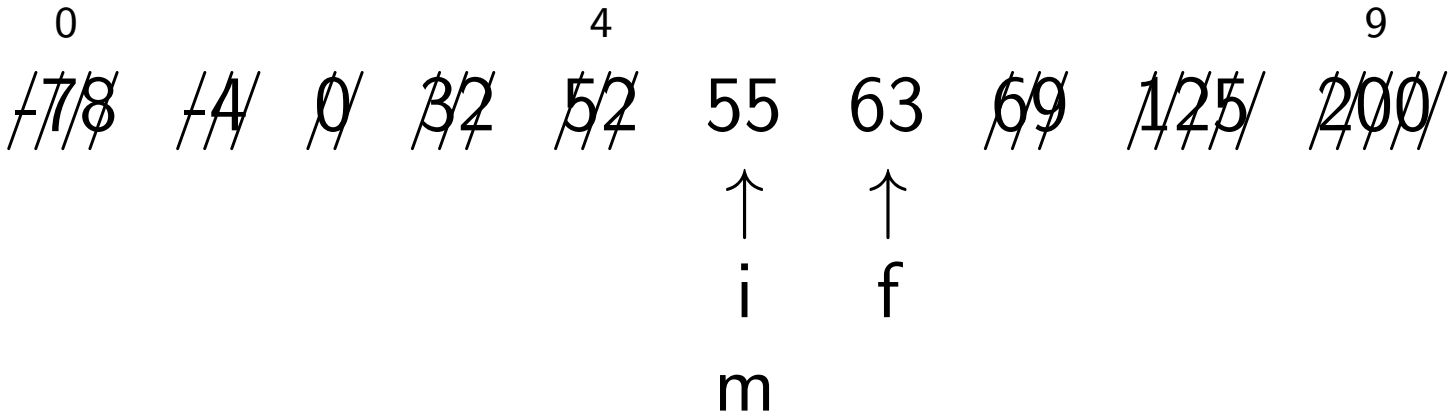
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



$v[m] == 60?$ Não. $v[m] < 60?$

Busca Binária

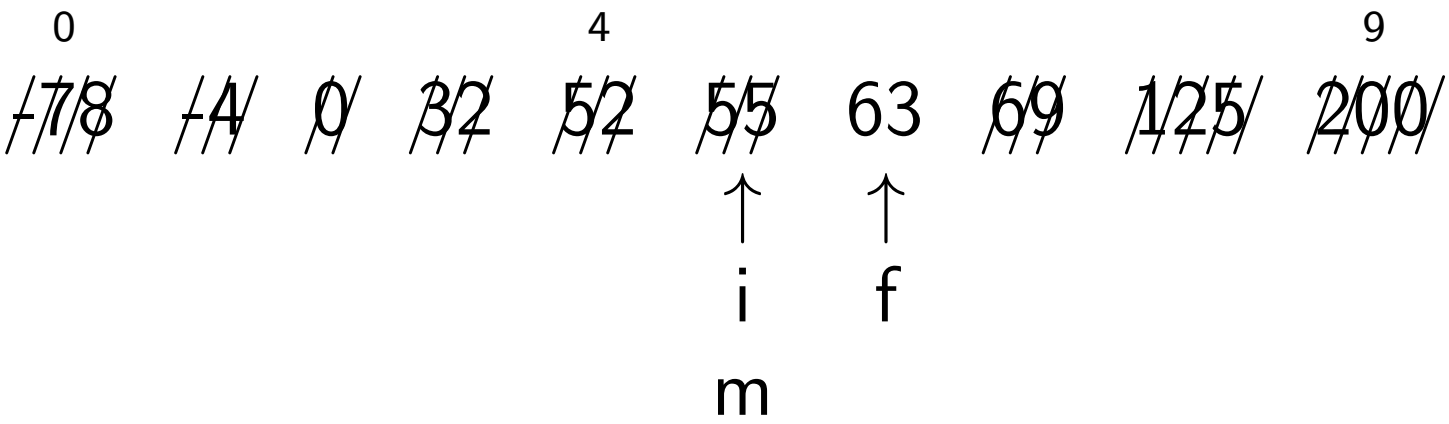
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



v[m] == 60? Não. v[m] < 60? Sim

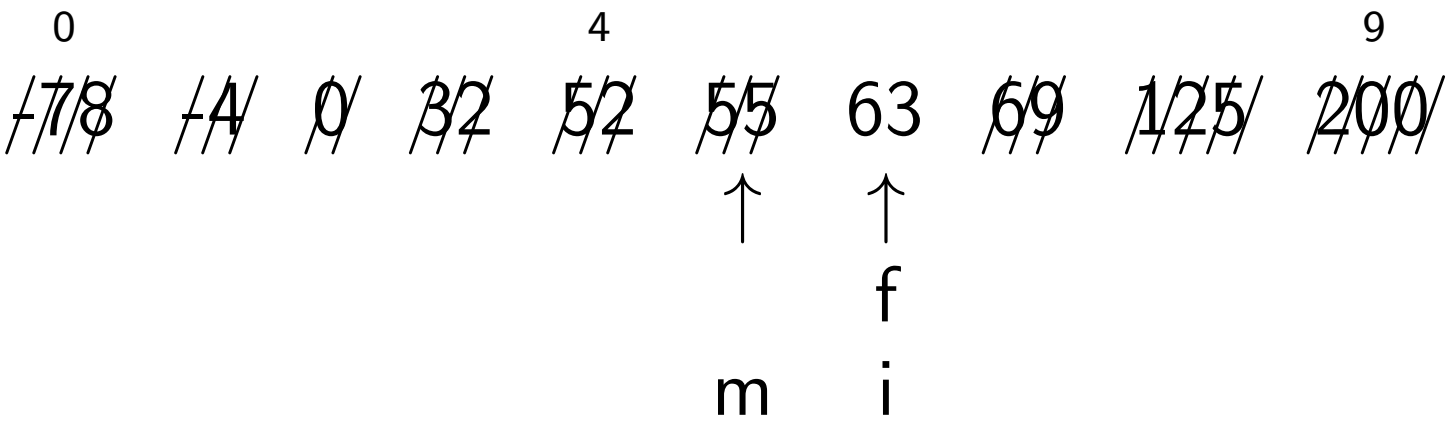
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



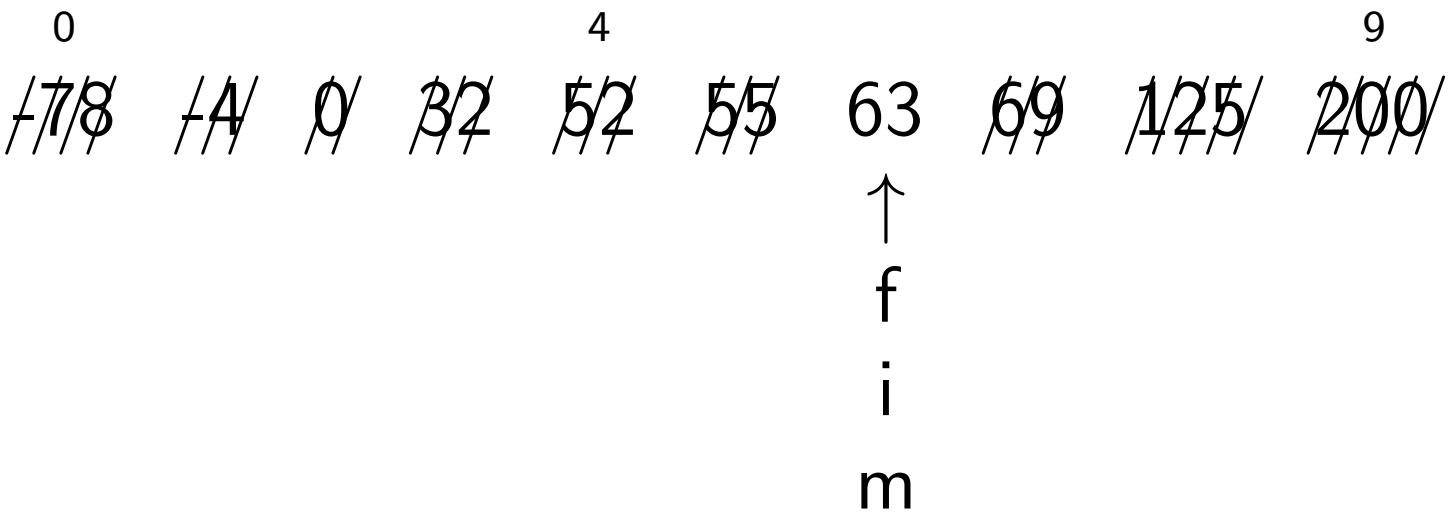
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



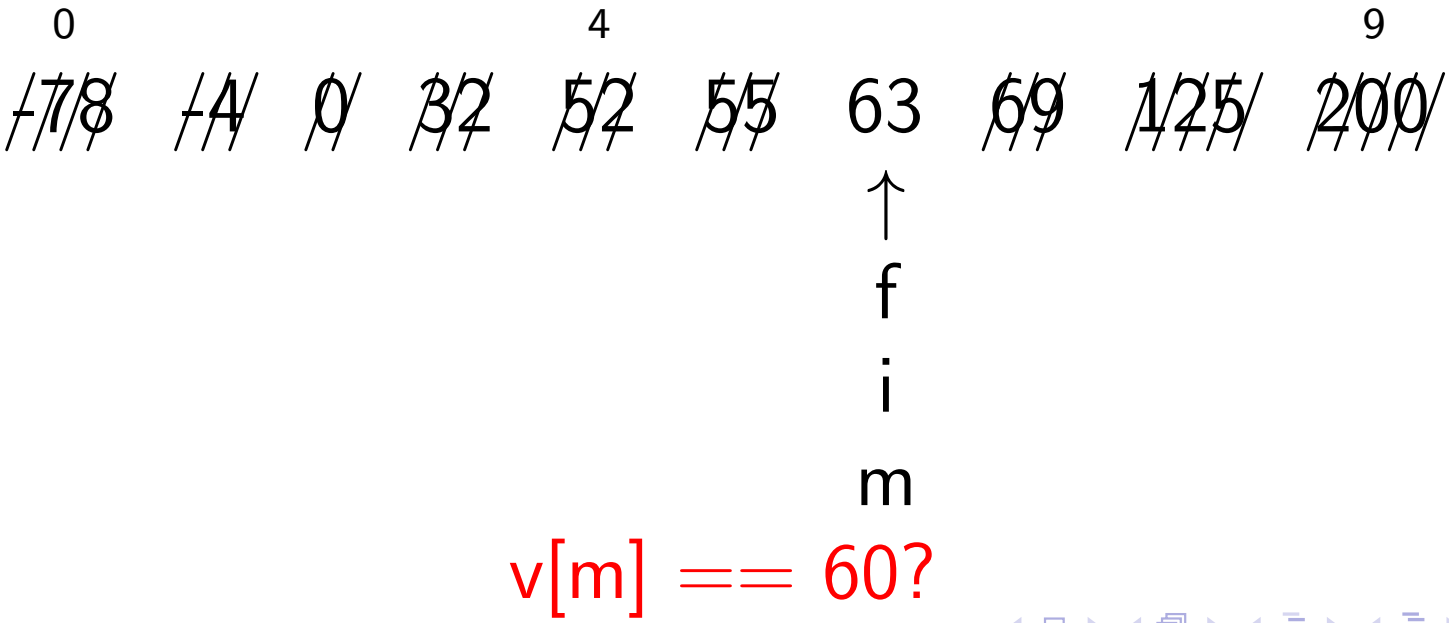
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



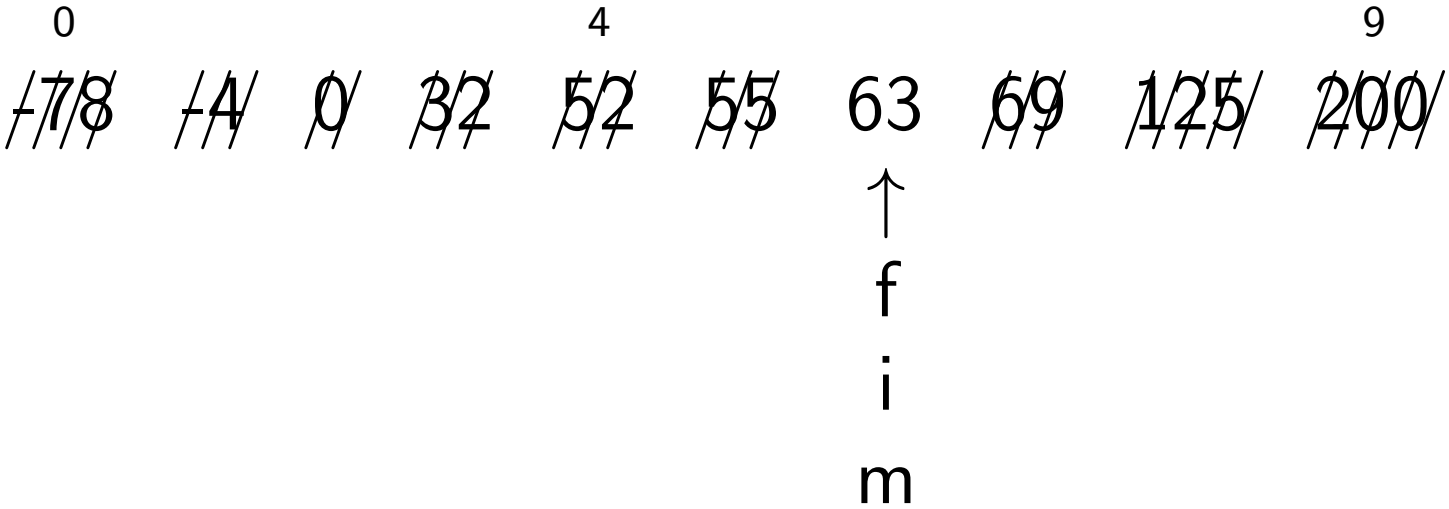
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



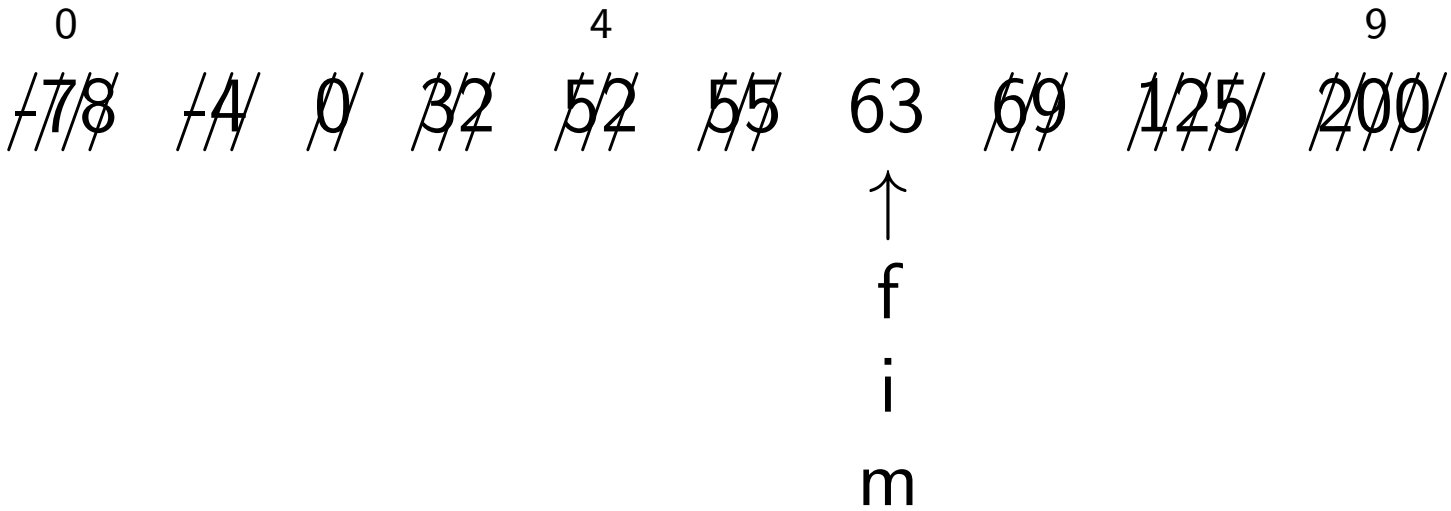
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



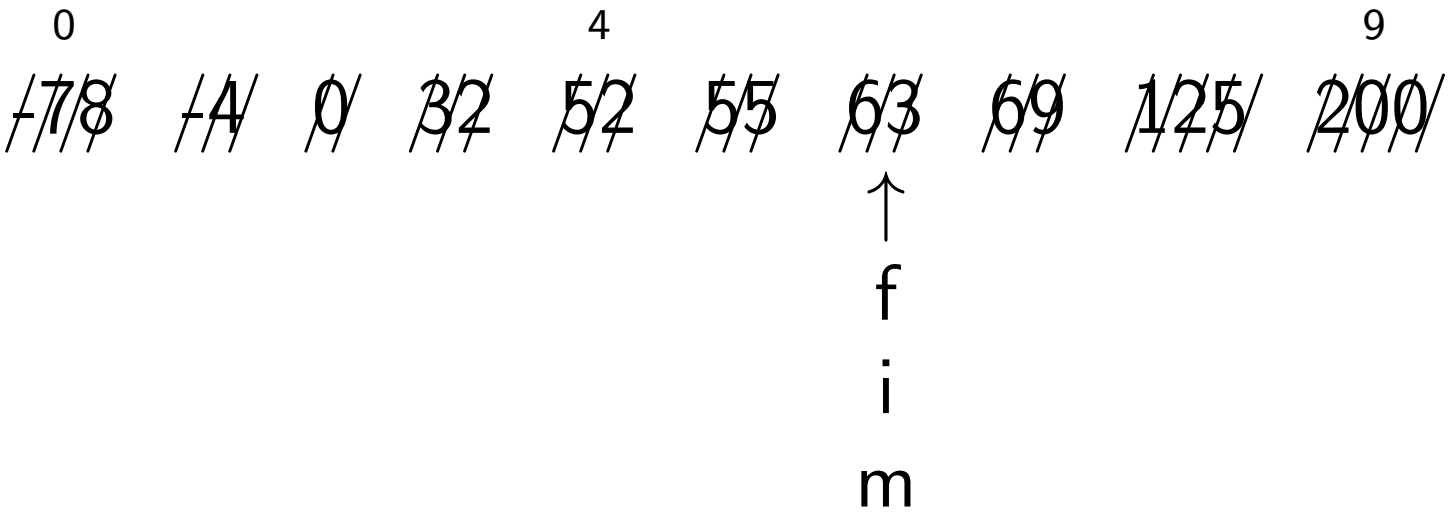
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



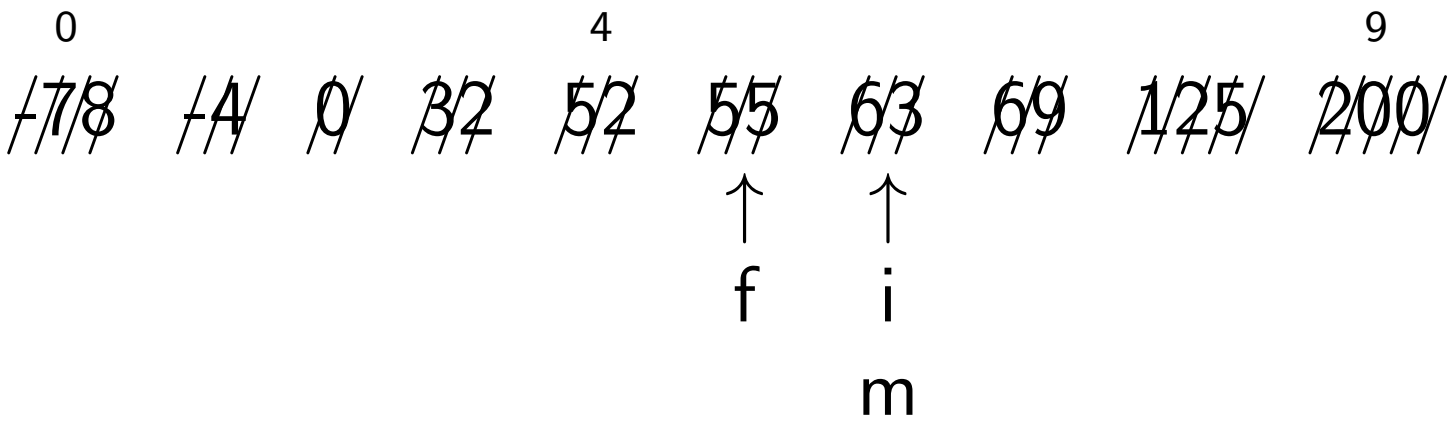
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



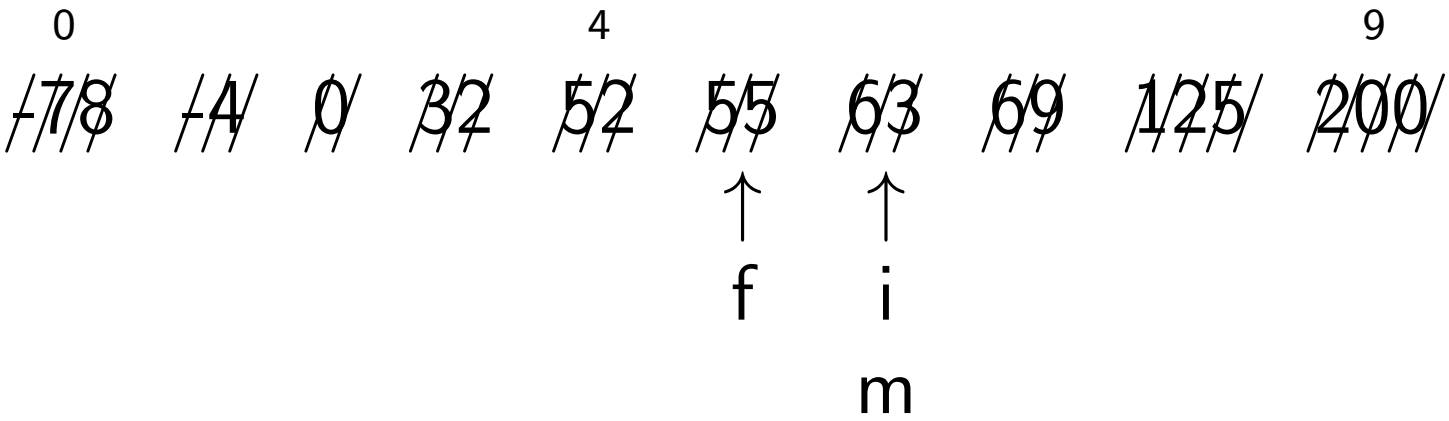
Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



Busca Binária

- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



Inconsistência. O elemento não está no arranjo

Busca Binária

- Então...

```
int buscaBin(int arr[],
             int tam, int el){
  int fim = tam-1;
  int ini = 0;
  while (ini <= fim) {
    int meio = (fim + ini)/2;
    if (arr[meio] < el)
      ini = meio + 1;
    else {
      if (arr[meio] > el)
         fim = meio - 1;
      else return meio;
    }
  }
  return -1;
}
```

Busca Binária

- Então...
- Note que retornamos o índice no arranjo do elemento buscado, ou -1 em caso de erro

```
int buscaBin(int arr[],
             int tam, int el){
  int fim = tam-1;
  int ini = 0;
  while (ini <= fim) {
    int meio = (fim + ini)/2;
    if (arr[meio] < el)
      ini = meio + 1;
    else {
      if (arr[meio] > el)
        fim = meio - 1;
      else return meio;
    }
  }
  return -1;
}
```

Busca Binária

- Sabemos que fazemos um máximo de n comparações com busca sequencial

Busca Binária

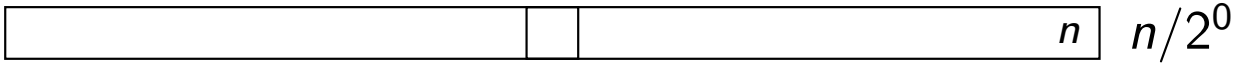
- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo

Busca Binária

- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo
- E quantas fazemos com a busca binária?

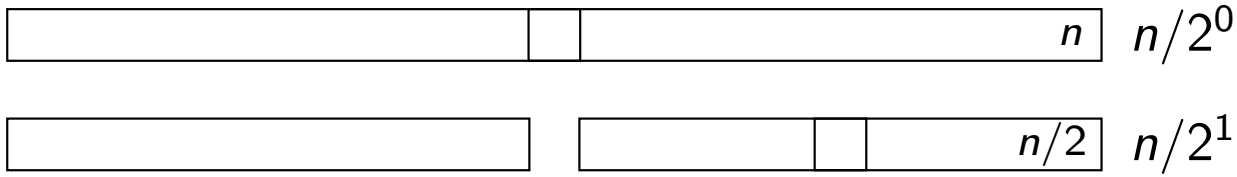
Busca Binária

- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo
- E quantas fazemos com a busca binária?



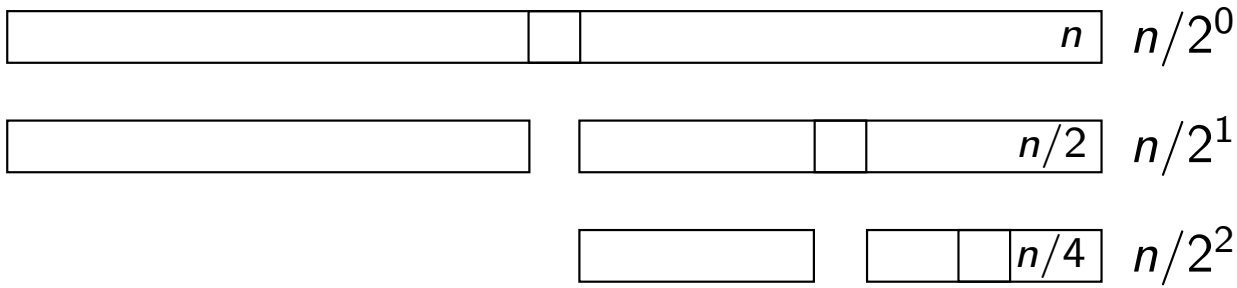
Busca Binária

- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo
- E quantas fazemos com a busca binária?



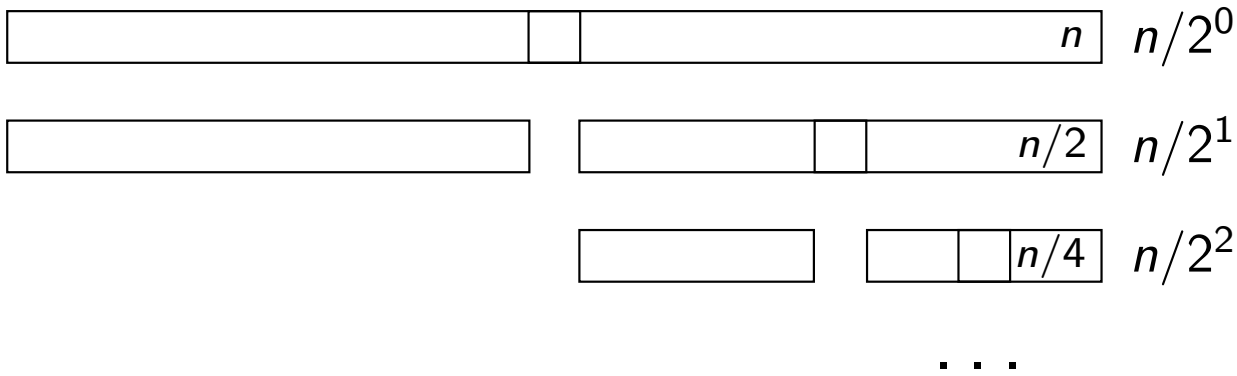
Busca Binária

- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo
- E quantas fazemos com a busca binária?



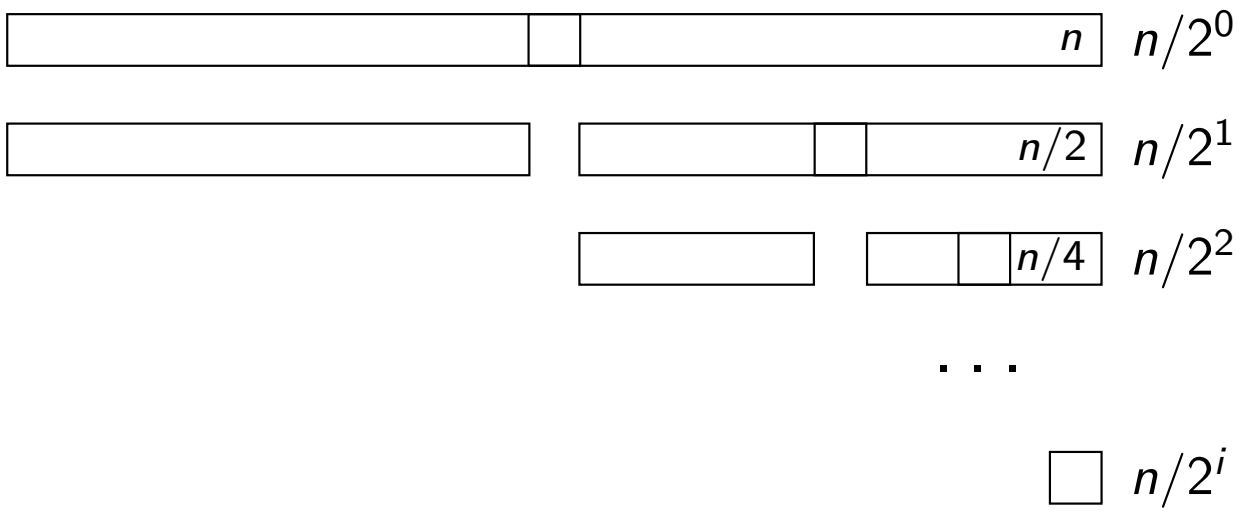
Busca Binária

- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo
- E quantas fazemos com a busca binária?

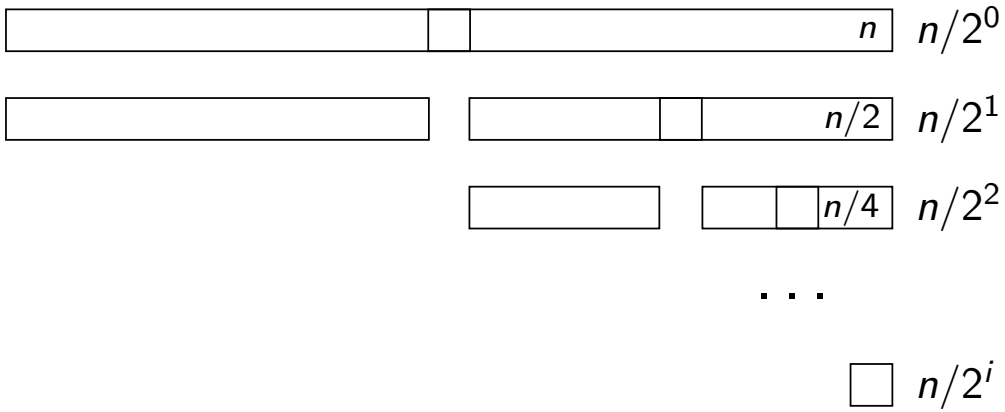


Busca Binária

- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo
- E quantas fazemos com a busca binária?

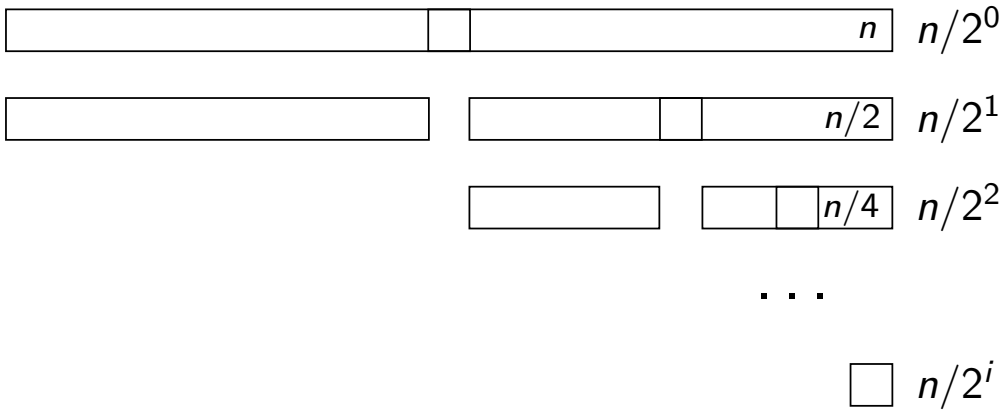


Busca Binária



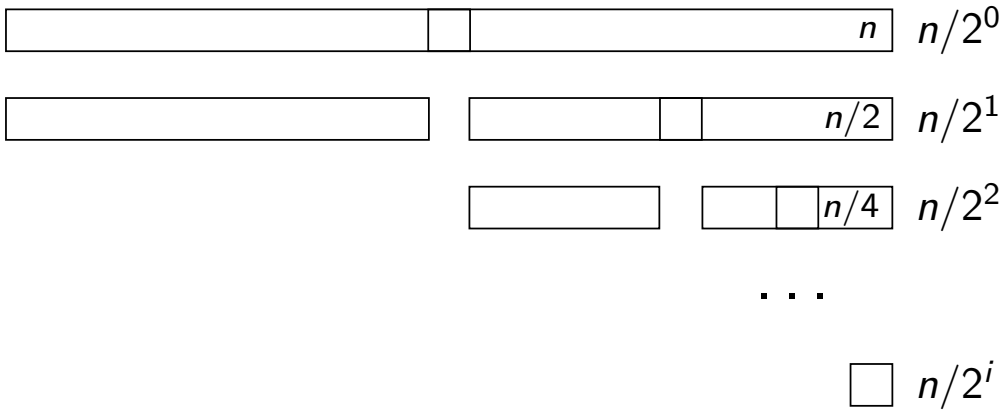
- Temos $i + 1$ comparações, sendo a última feita com o arranjo de tamanho 1

Busca Binária



- Temos $i + 1$ comparações, sendo a última feita com o arranjo de tamanho 1
- A relação entre n e i é tal que, após i comparações, o arranjo terá $n/2^i$ elementos.

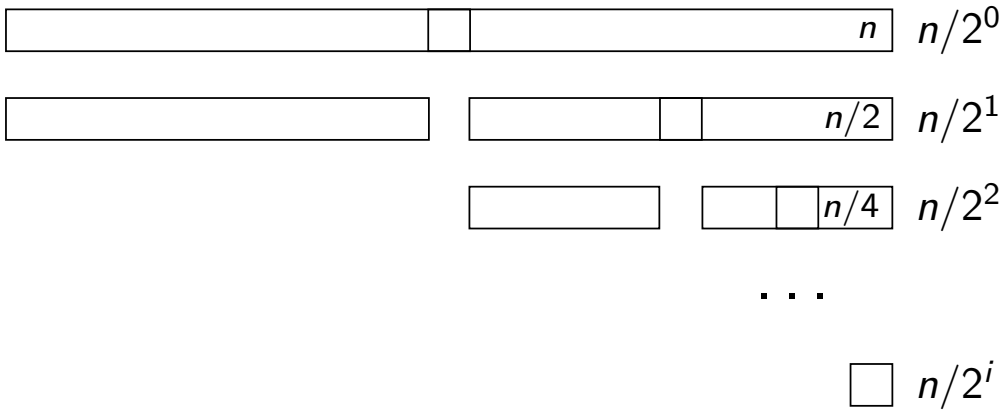
Busca Binária



- Temos $i + 1$ comparações, sendo a última feita com o arranjo de tamanho 1
- A relação entre n e i é tal que, após i comparações, o arranjo terá $n/2^i$ elementos.

- Como no último nível há 1 elemento, então $n/2^i = 1 \Rightarrow n = 2^i \Rightarrow log_2(n) = i$

Busca Binária



- Temos $i + 1$ comparações, sendo a última feita com o arranjo de tamanho 1
- A relação entre n e i é tal que, após i comparações, o arranjo terá $n/2^i$ elementos.

- Como no último nível há 1 elemento, então $n/2^i = 1 \Rightarrow n = 2^i \Rightarrow log_2(n) = i$
- Assim temos $log_2(n) + 1$ comparações

Busca Sequencial × Busca Binária

Sequencial

Binária

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro

Binária

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro
  - 1 comparação (arranjo ordenado ou não)

Binária

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro
  - 1 comparação (arranjo ordenado ou não)

Binária

- Melhor caso: O elemento é o do meio

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro
  - 1 comparação (arranjo ordenado ou não)

Binária

- Melhor caso: O elemento é o do meio
  - 1 comparação

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro
  - 1 comparação (arranjo ordenado ou não)
- Pior caso: O elemento não está no arranjo e é maior que todos

Binária

- Melhor caso: O elemento é o do meio
  - 1 comparação

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro
  - 1 comparação (arranjo ordenado ou não)
- Pior caso: O elemento não está no arranjo e é maior que todos
  - n comparações (arranjo ordenado ou não)

Binária

- Melhor caso: O elemento é o do meio
  - 1 comparação

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro
  - 1 comparação (arranjo ordenado ou não)
- Pior caso: O elemento não está no arranjo e é maior que todos
  - n comparações (arranjo ordenado ou não)

Binária

- Melhor caso: O elemento é o do meio
  - 1 comparação
- Pior caso: O elemento não está no arranjo

Busca Sequencial × Busca Binária

Sequencial

- Melhor caso: O elemento é o primeiro
  - 1 comparação (arranjo ordenado ou não)
- Pior caso: O elemento não está no arranjo e é maior que todos
  - n comparações (arranjo ordenado ou não)

Binária

- Melhor caso: O elemento é o do meio
  - 1 comparação
- Pior caso: O elemento não está no arranjo
  - $log_2(n) + 1$ comparações

Busca Sequencial $\times$ Busca Binária

Observação:

- $log_2(n) + 1 < n$ para $n \geq 3$
  - Para 1 e 2, $log_2(n) + 1 = n$

Busca Sequencial × Busca Binária

Observação:

- $log_2(n) + 1 < n$ para $n \geq 3$
  - Para 1 e 2, $log_2(n) + 1 = n$
- No pior caso, a busca binária é pelo menos tão boa quanto a sequencial, mas apenas para arranjos de tamanho mínimo.

Busca Sequencial × Busca Binária

Observação:

- $log_2(n) + 1 < n$ para $n \geq 3$
  - Para 1 e 2, $log_2(n) + 1 = n$
- No pior caso, a busca binária é pelo menos tão boa quanto a sequencial, mas apenas para arranjos de tamanho mínimo.
- Para os demais, ela é melhor

Complexidade Computacional

- Estudo do esforço computacional despendido para que o algoritmo seja executado

Complexidade Computacional

- Estudo do esforço computacional despendido para que o algoritmo seja executado
- Pode ser avaliado o melhor ou o pior caso, ou ainda o caso médio.

Complexidade Computacional

- Estudo do esforço computacional despendido para que o algoritmo seja executado
- Pode ser avaliado o melhor ou o pior caso, ou ainda o caso médio.
  - Normalmente é considerado o pior caso

Complexidade Computacional

- Estudo do esforço computacional despendido para que o algoritmo seja executado
- Pode ser avaliado o melhor ou o pior caso, ou ainda o caso médio.
  - Normalmente é considerado o pior caso
- Importância: decisão de qual algoritmo usar dependendo do requisito do seu sistema

Complexidade Computacional

Exemplo:

Complexidade Computacional

Exemplo:

- Lista telefônica de São Paulo: $\approx$ 18 milhões de entradas

Complexidade Computacional

Exemplo:

- Lista telefônica de São Paulo: $\approx$ 18 milhões de entradas
- Se cada comparação (a um elemento do arranjo) gasta 10 μs (10 milionésimos de segundo), como ficam os piores casos?

Complexidade Computacional

Exemplo:

- Lista telefônica de São Paulo: $\approx$ 18 milhões de entradas
- Se cada comparação (a um elemento do arranjo) gasta 10 μs (10 milionésimos de segundo), como ficam os piores casos?
  - Busca sequencial:

Complexidade Computacional

Exemplo:

- Lista telefônica de São Paulo: $\approx$ 18 milhões de entradas
- Se cada comparação (a um elemento do arranjo) gasta 10 μs (10 milionésimos de segundo), como ficam os piores casos?
  - Busca sequencial: 10/1000000 * 18000000 = 180s = 3 minutos

Complexidade Computacional

Exemplo:

- Lista telefônica de São Paulo: $\approx$ 18 milhões de entradas
- Se cada comparação (a um elemento do arranjo) gasta 10 μs (10 milionésimos de segundo), como ficam os piores casos?
  - Busca sequencial: 10/1000000 * 18000000 = 180s = 3 minutos
  - Busca binária:

Complexidade Computacional

Exemplo:

- Lista telefônica de São Paulo: $\approx$ 18 milhões de entradas
- Se cada comparação (a um elemento do arranjo) gasta 10 μs (10 milionésimos de segundo), como ficam os piores casos?
  - Busca sequencial: 10/1000000 * 18000000 = 180s = 3 minutos
  - Busca binária: 10/1000000 * log_2 18000000 = 0.000241s = 0,24 milisegundos

Quando Usar Busca Binária?

Qual o problema da busca binária?

- Precisa que o arranjo esteja ordenado

Quando Usar Busca Binária?

Qual o problema da busca binária?

- Precisa que o arranjo esteja ordenado
- Então, quando vale realmente a pena usá-la?

Quando Usar Busca Binária?

Qual o problema da busca binária?

- Precisa que o arranjo esteja ordenado
- Então, quando vale realmente a pena usá-la?
  - Quando há muitas buscas

Quando Usar Busca Binária?

Qual o problema da busca binária?

- Precisa que o arranjo esteja ordenado
- Então, quando vale realmente a pena usá-la?
  - Quando há muitas buscas
  - Quando os dados sofrem pouca alteração na chave de busca

Quando Usar Busca Binária?

Qual o problema da busca binária?

- Precisa que o arranjo esteja ordenado
- Então, quando vale realmente a pena usá-la?
  - Quando há muitas buscas
  - Quando os dados sofrem pouca alteração na chave de busca
  - Quando as inserções/deleções não são frequentes

Quando Usar Busca Binária?

Qual o problema da busca binária?

- Precisa que o arranjo esteja ordenado
- Então, quando vale realmente a pena usá-la?
  - Quando há muitas buscas
  - Quando os dados sofrem pouca alteração na chave de busca
  - Quando as inserções/deleções não são frequentes
- Em suma, quando a ordem não é mudada com frequência

Aula 28 – Busca Sequencial e Binária

Norton T. Roman & Luciano A. Digiampietri