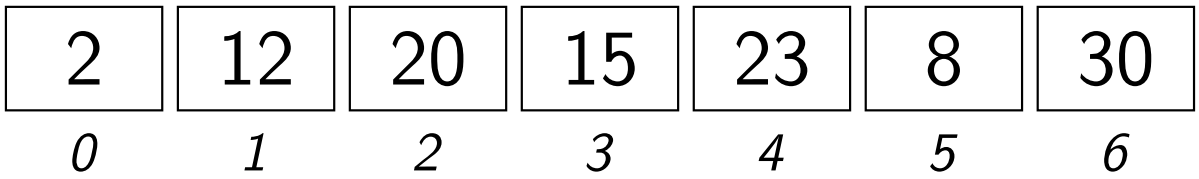
Aula 17 – Busca Binária e Ordenação

Norton Trevisan Roman

22 de maio de 2018

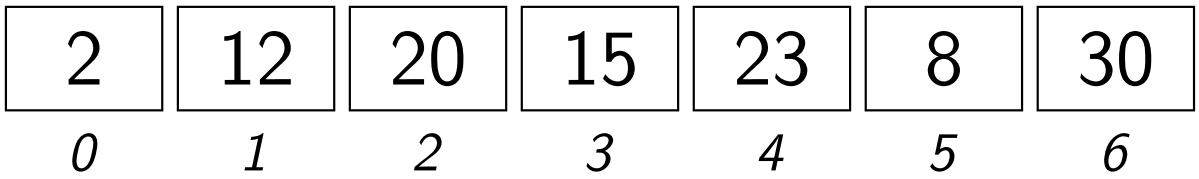
Busca Sequencial

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



Busca Sequencial

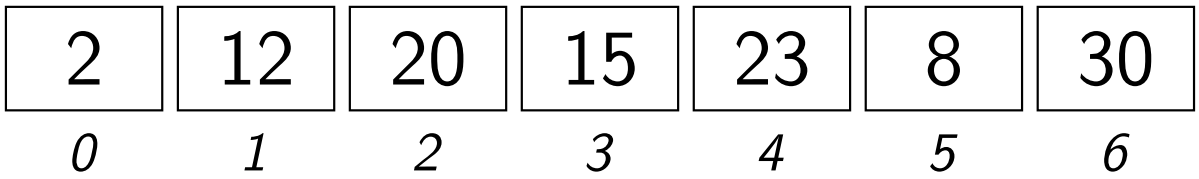
- O que podemos fazer para melhorar a busca por 9?



Nada, realmente...

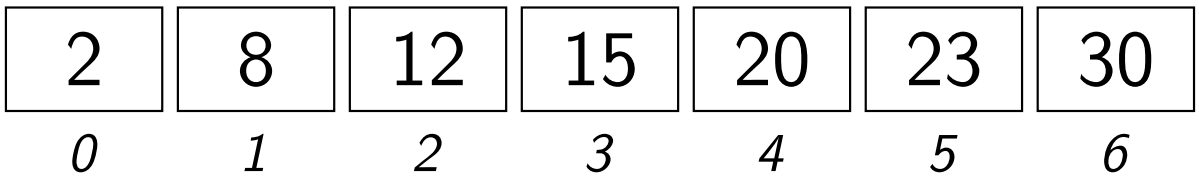
Busca Sequencial

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



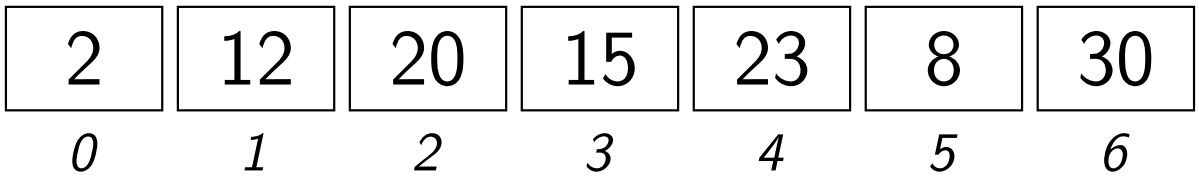
Nada, realmente...

- E se o arranjo estivesse ordenado?



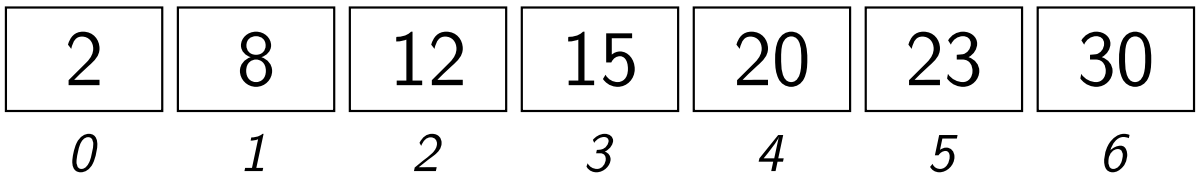
Busca Sequencial

- 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

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

public static void main(String[] args) {
  int[] v = {-78,-4,0,32,52,55,63,69,125,
                                      200};
  System.out.println(buscaSeq(v, 23));
  System.out.println(buscaSeq(v, 8));
}
```

Busca em Arranjo Ordenado

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

public static void main(String[] args) {
  int[] v = {-78,-4,0,32,52,55,63,69,125,
                                      200};
  System.out.println(buscaSeq(v, 23));
  System.out.println(buscaSeq(v, 8));
}
```

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

Busca em Arranjo Ordenado

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

public static void main(String[] args) {
  int[] v = {-78,-4,0,32,52,55,63,69,125,
                                      200};
  System.out.println(buscaSeq(v, 23));
  System.out.println(buscaSeq(v, 8));
}
```

- 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

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

public static void main(String[] args) {
  int[] v = {-78,-4,0,32,52,55,63,69,125,
                                      200};
  System.out.println(buscaSeq(v, 23));
  System.out.println(buscaSeq(v, 8));
}
```

- 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

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

public static void main(String[] args) {
  int[] v = {-78,-4,0,32,52,55,63,69,125,
                                      200};
  System.out.println(buscaSeq(v, 23));
  System.out.println(buscaSeq(v, 8));
}
```

- 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

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

public static void main(String[] args) {
  int[] v = {-78,-4,0,32,52,55,63,69,125,
                                      200};
  System.out.println(buscaSeq(v, 23));
  System.out.println(buscaSeq(v, 8));
}
```

- 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

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

public static void main(String[] args) {
  int[] v = {-78,-4,0,32,52,55,63,69,125,
                                      200};
  System.out.println(buscaSeq(v, 23));
  System.out.println(buscaSeq(v, 8));
}
```

- 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[v.length-1]`

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

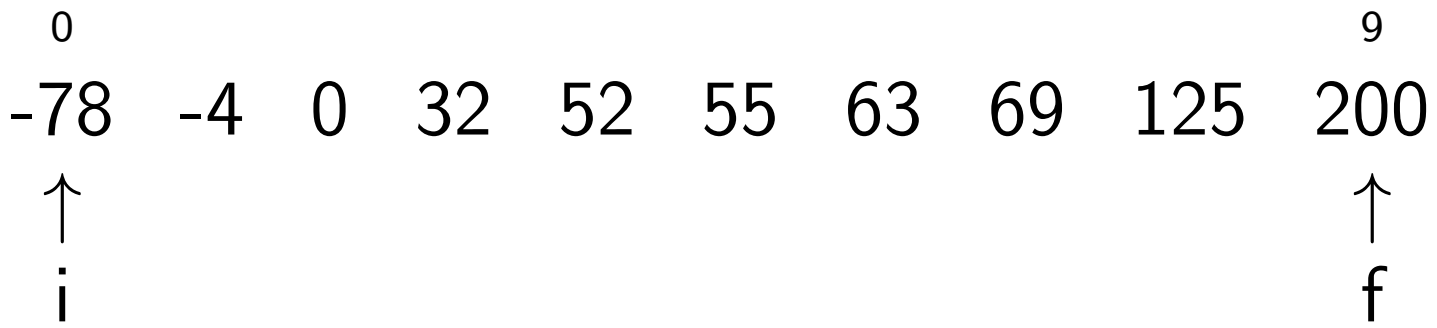
Busca Binária

- 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

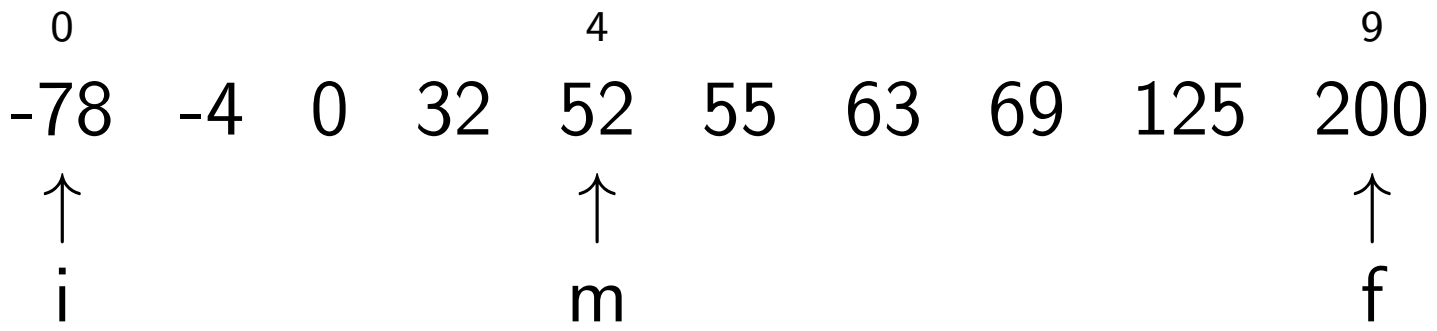
Busca Binária

- 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



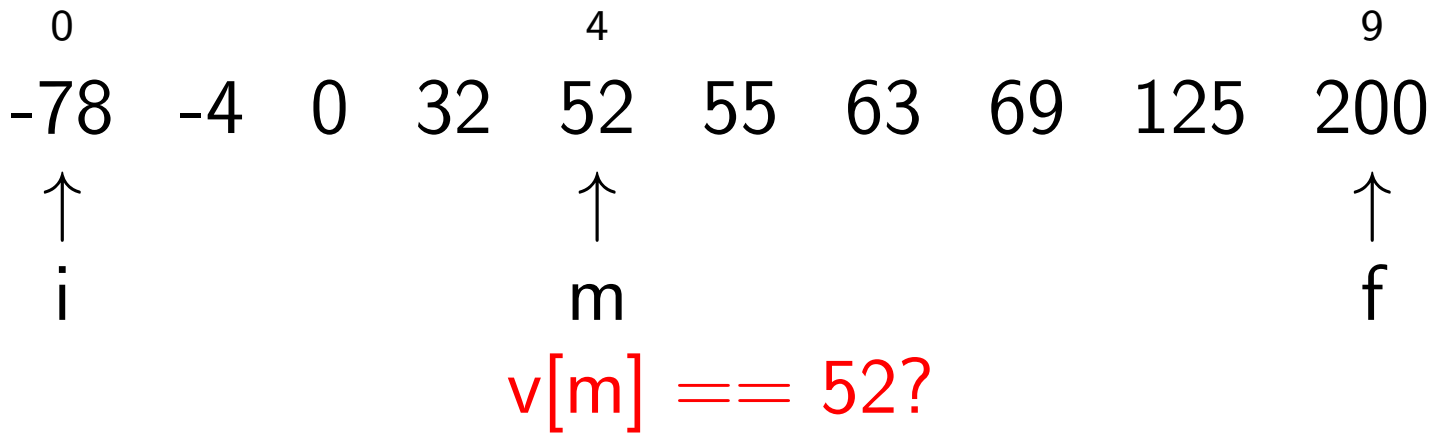
Busca Binária

- 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



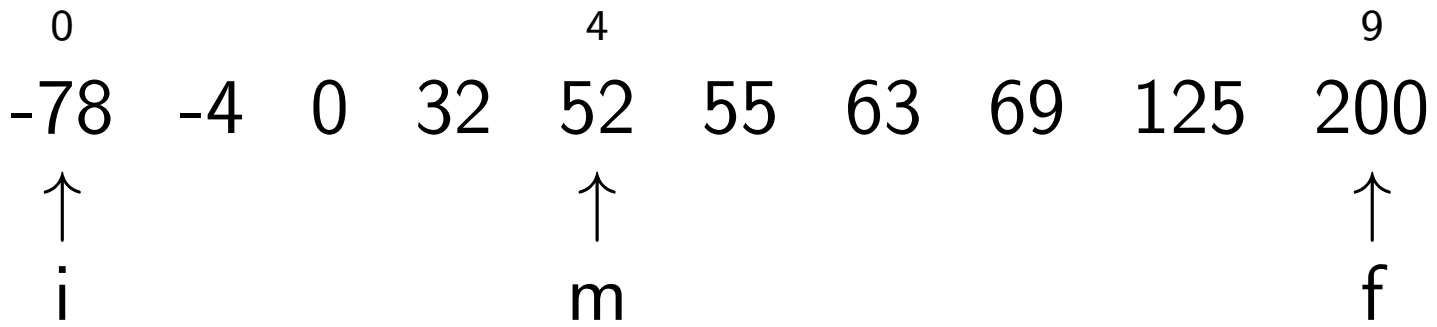
Busca Binária

- 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

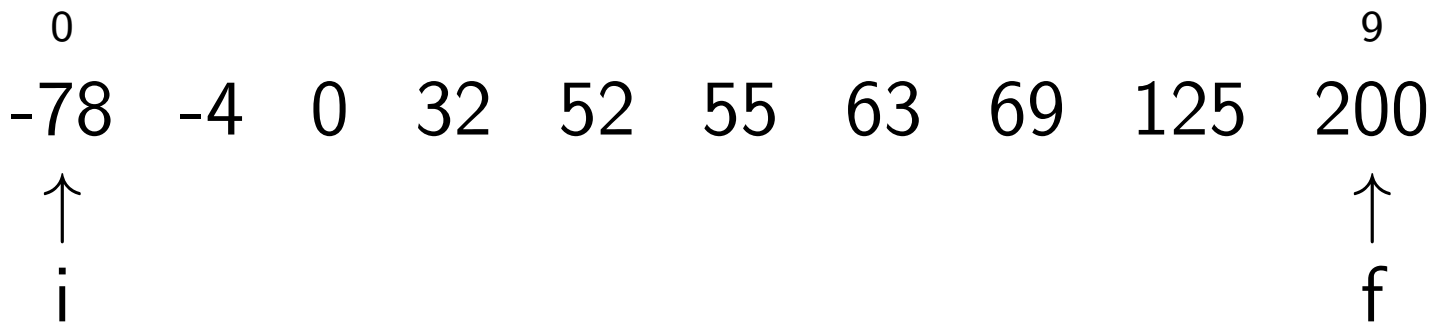
Busca Binária

- 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

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

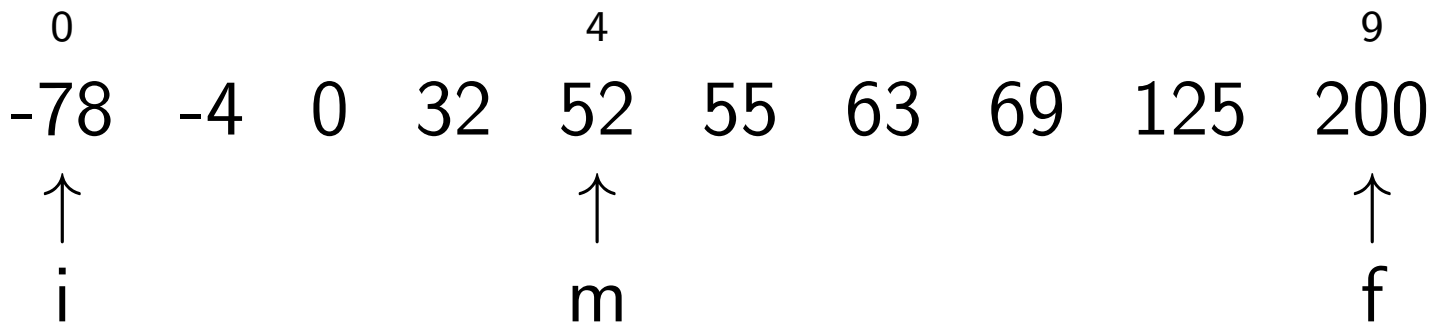
Busca Binária

- 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 55



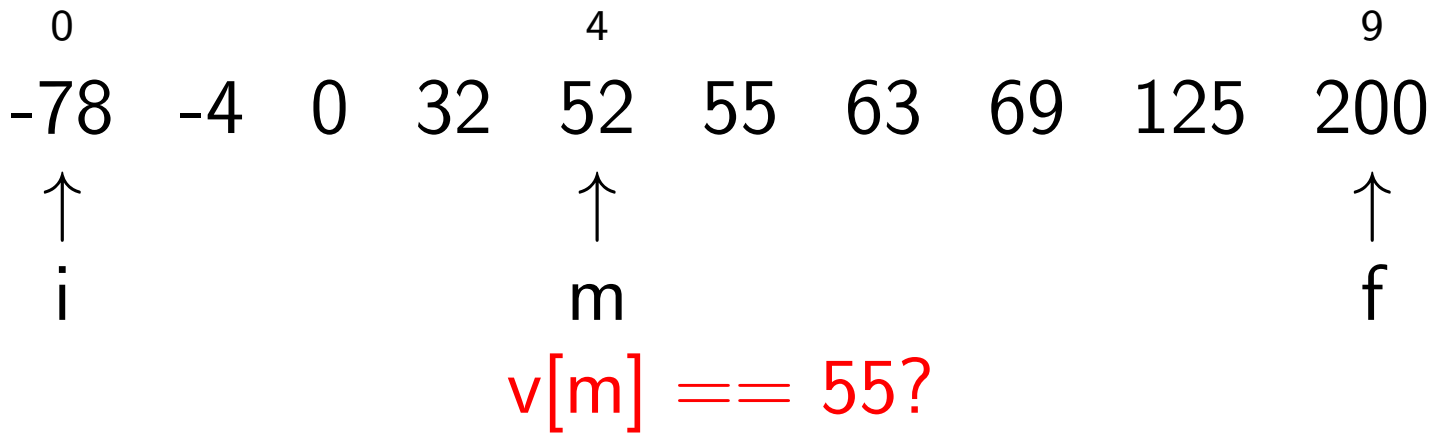
Busca Binária

- 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



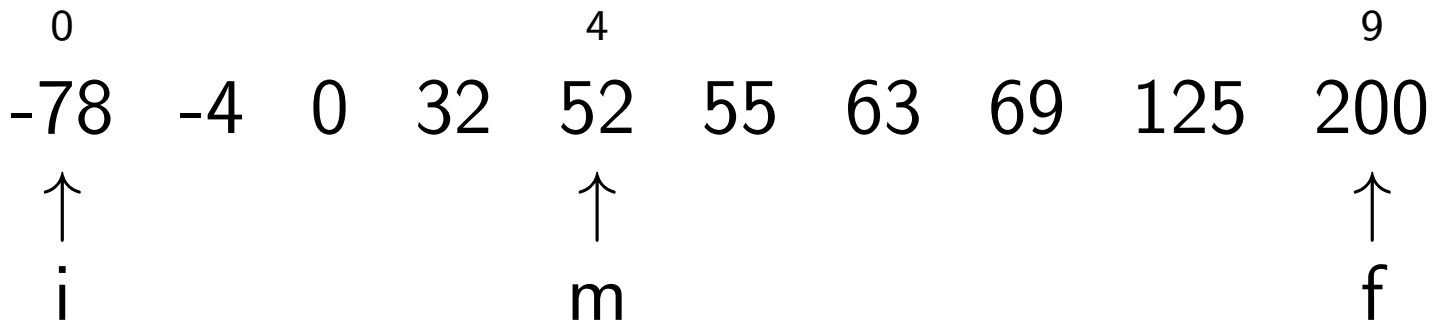
Busca Binária

- 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

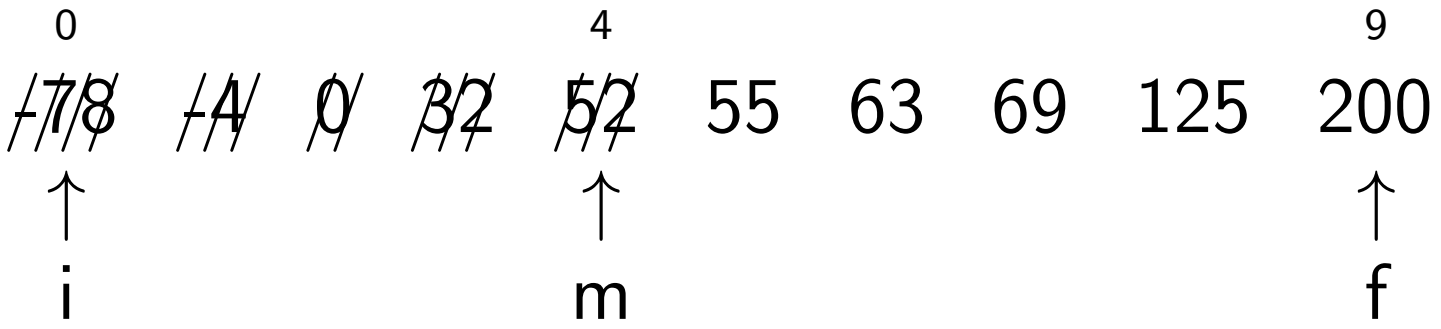
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55?

Busca Binária

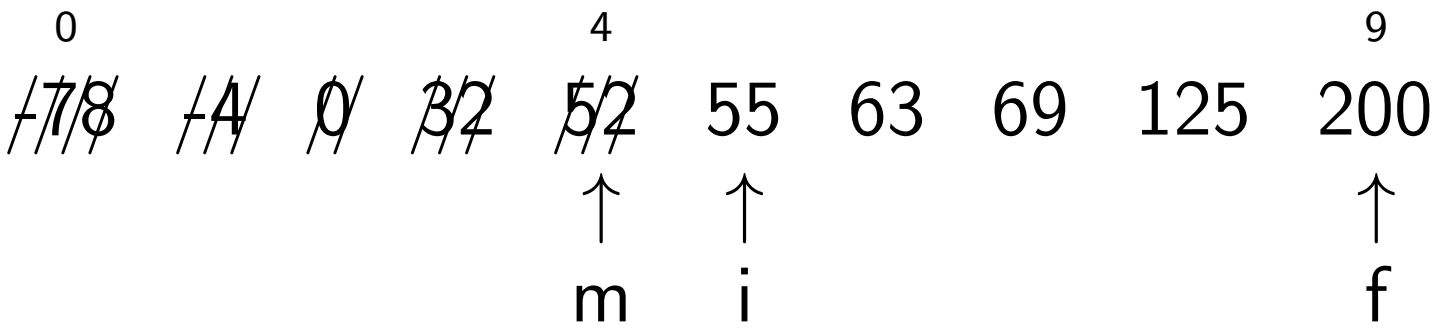
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55? Sim.

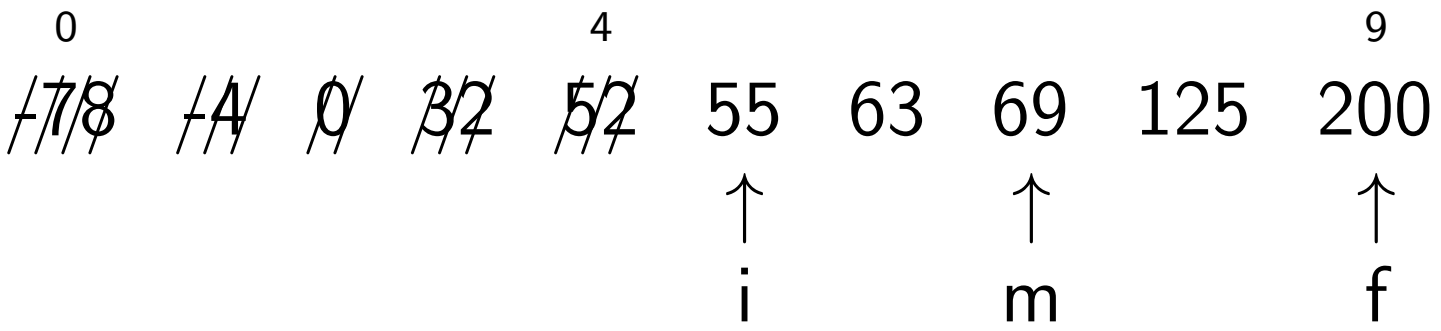
Busca Binária

- 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



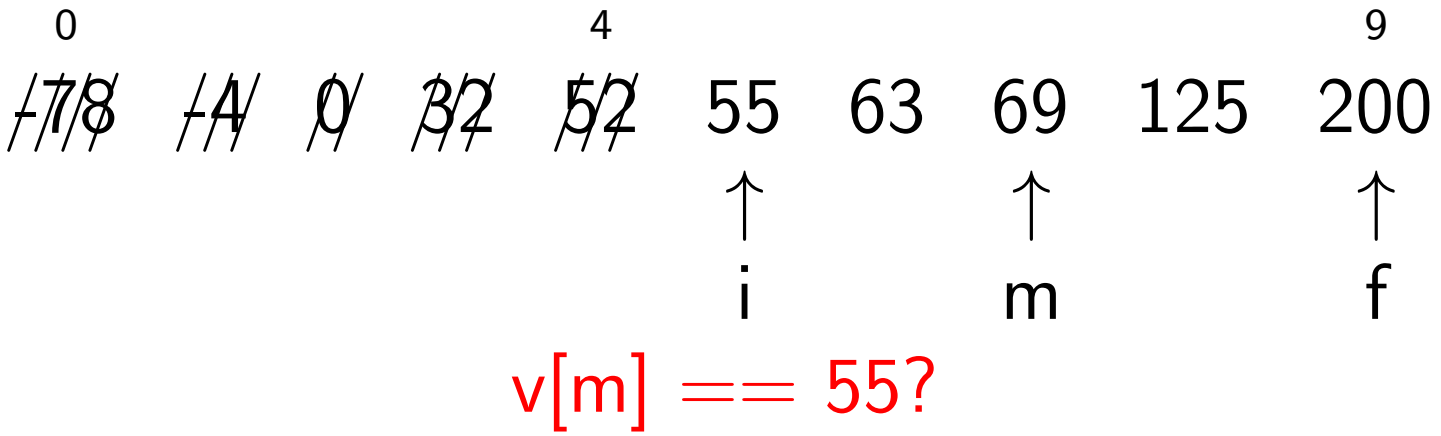
Busca Binária

- 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



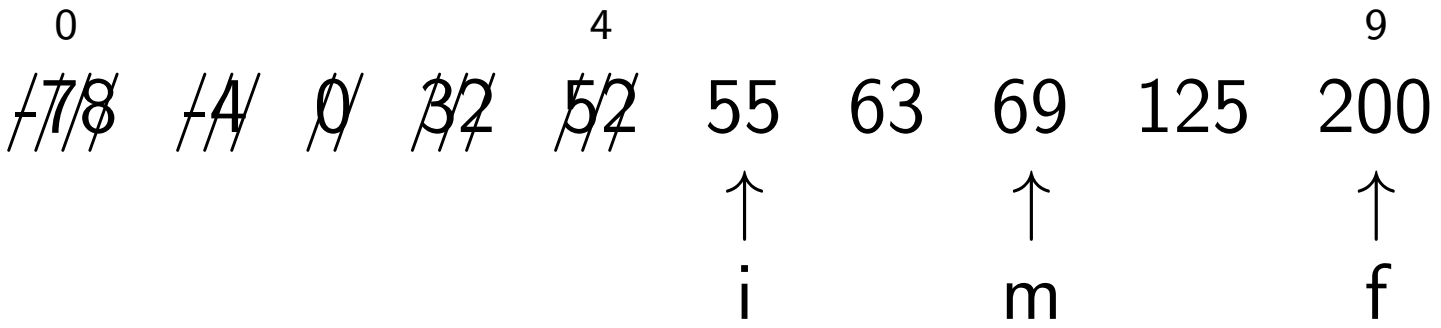
Busca Binária

- 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

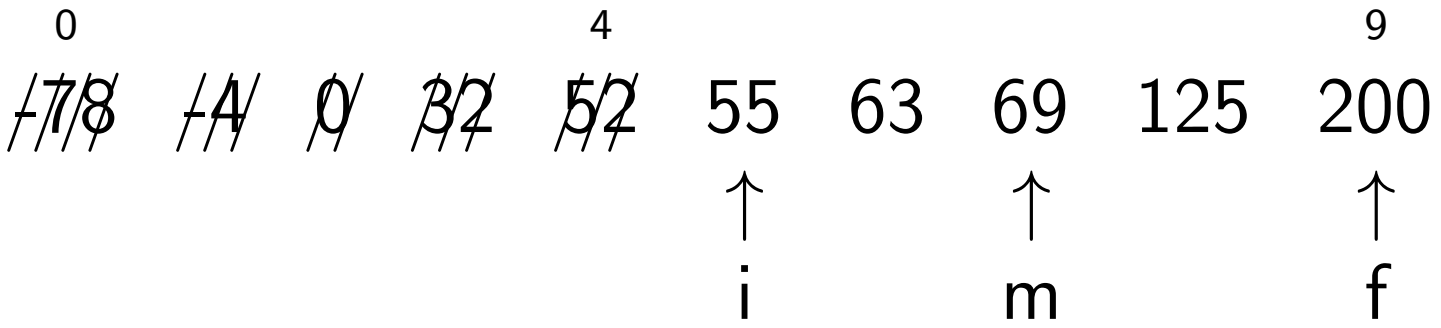
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55?

Busca Binária

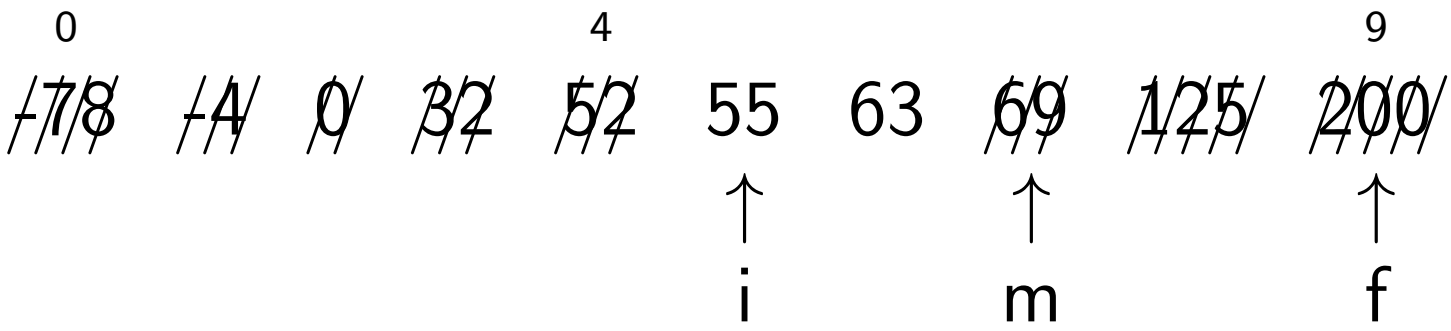
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 55



v[m] == 55? Não. v[m] < 55? Não.

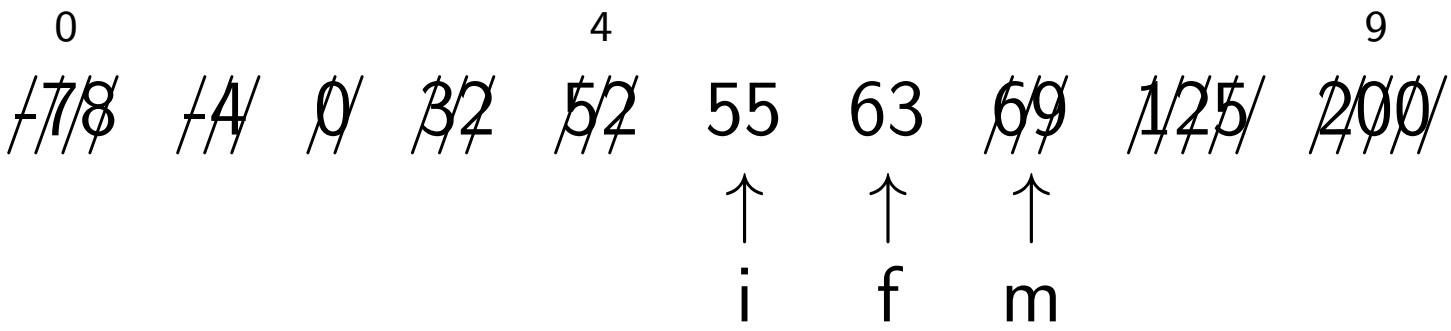
Busca Binária

- 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



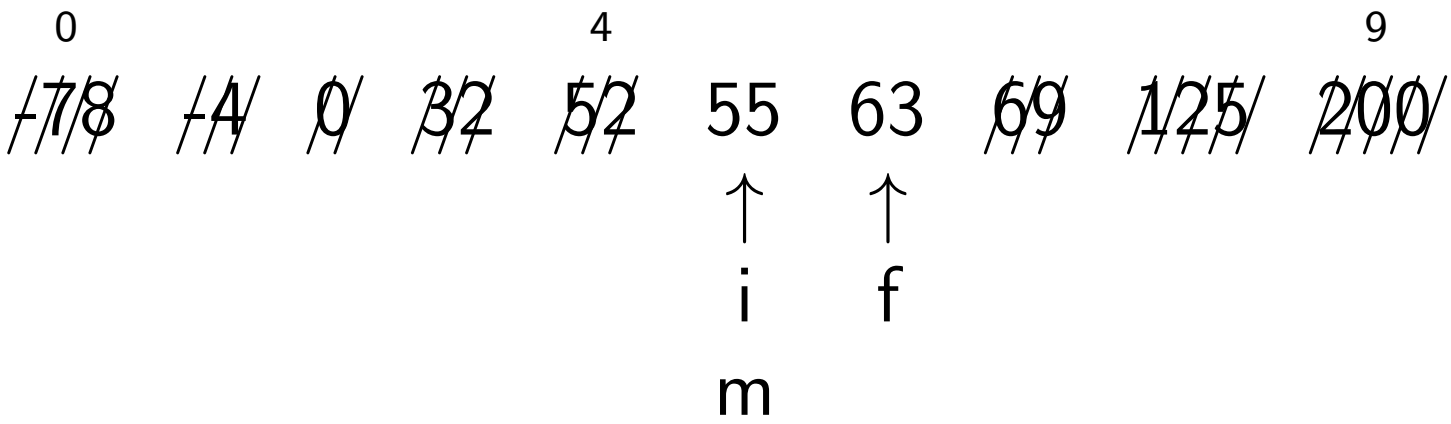
Busca Binária

- 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



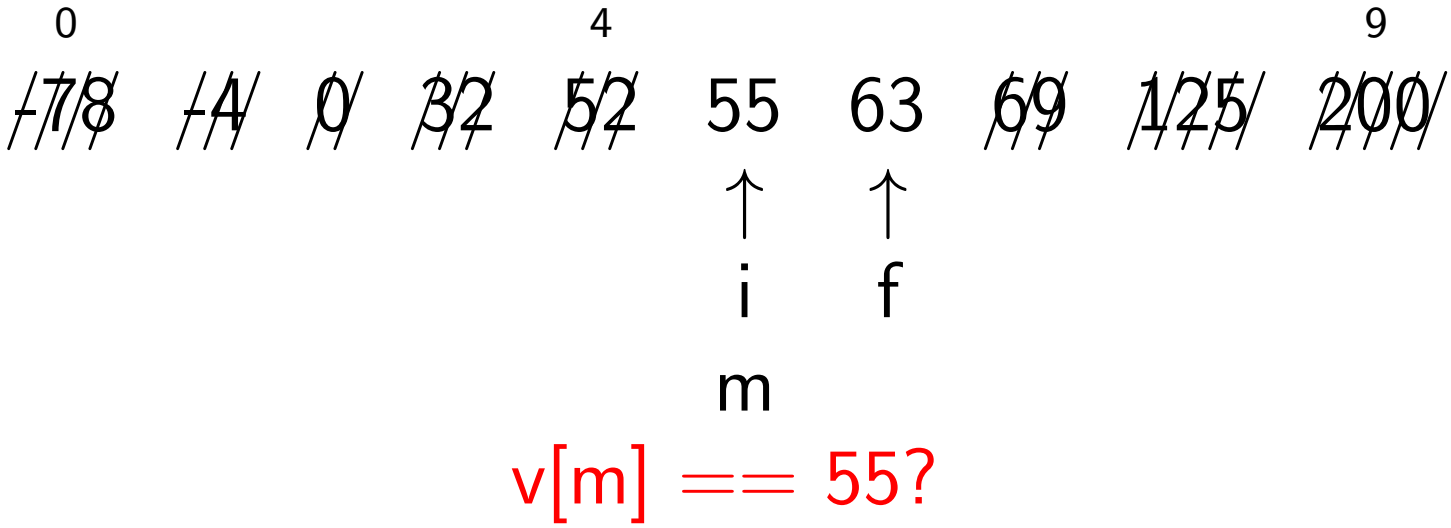
Busca Binária

- 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



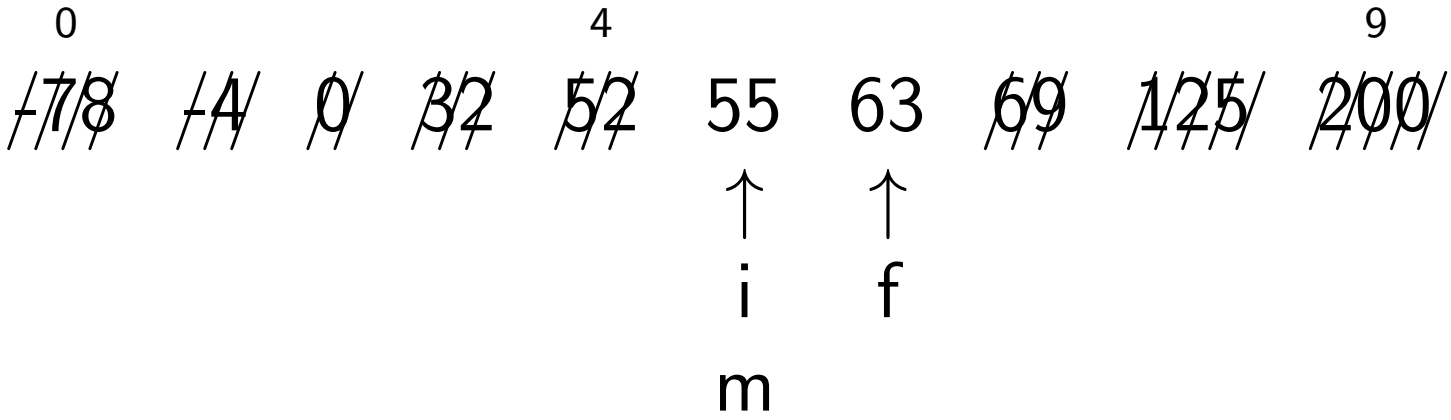
Busca Binária

- 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



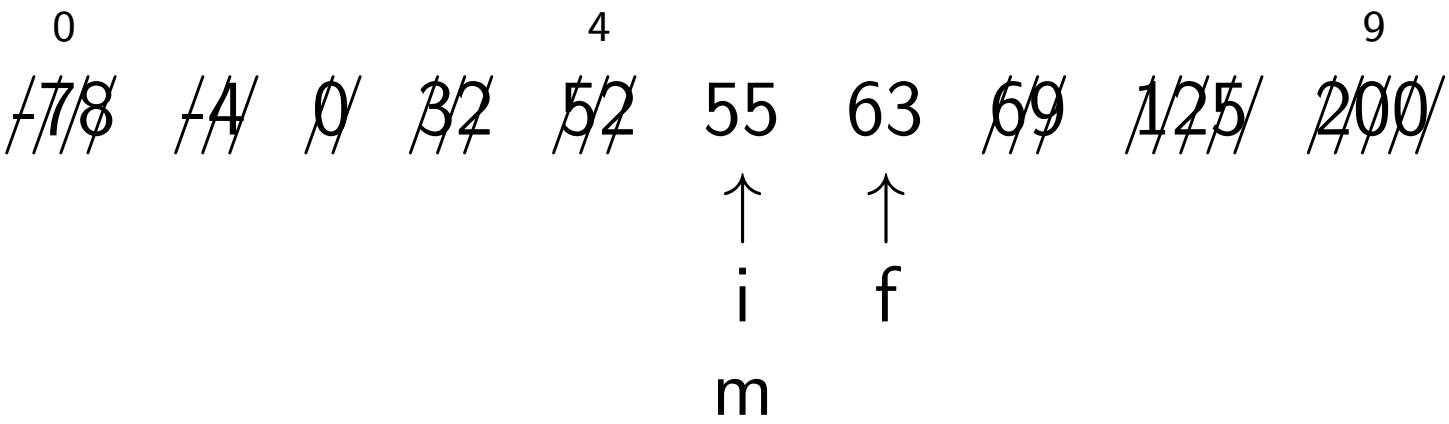
Busca Binária

- 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



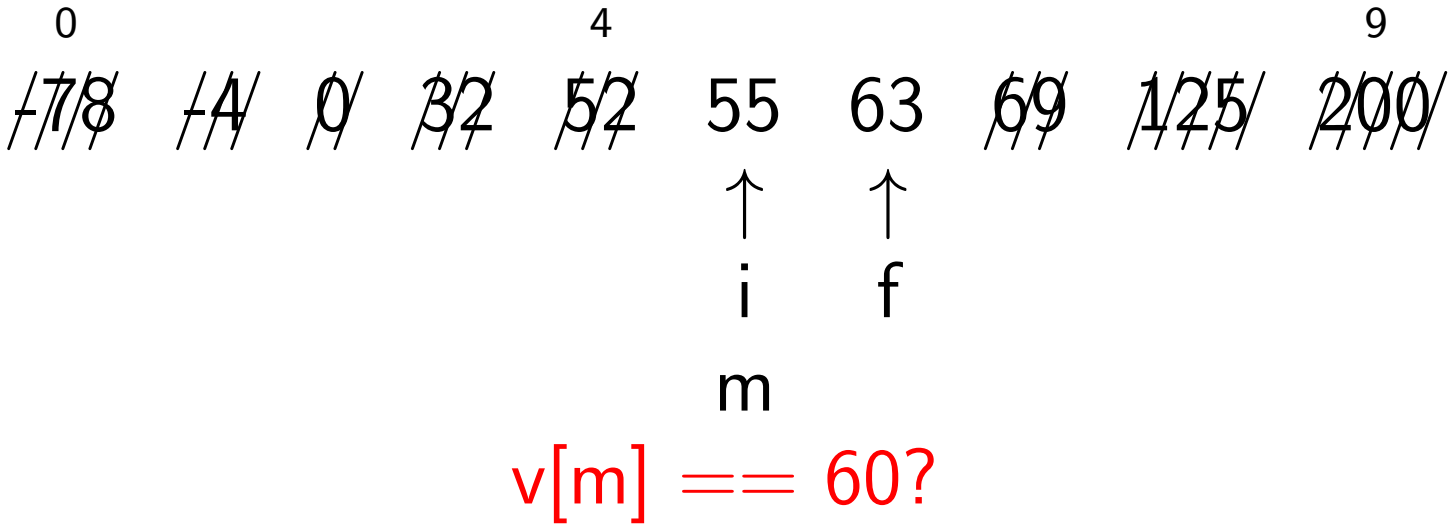
Busca Binária

- 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



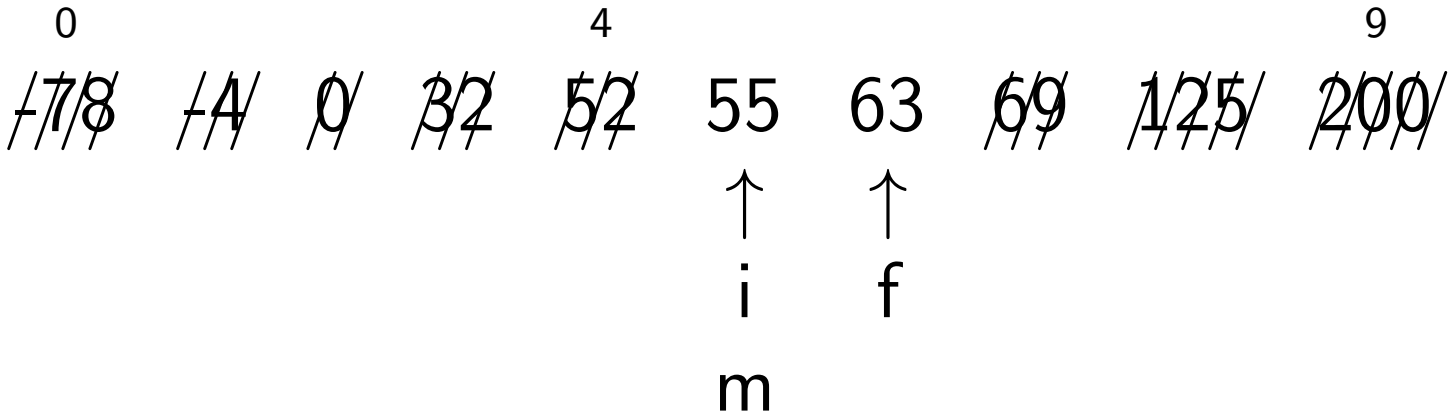
Busca Binária

- 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



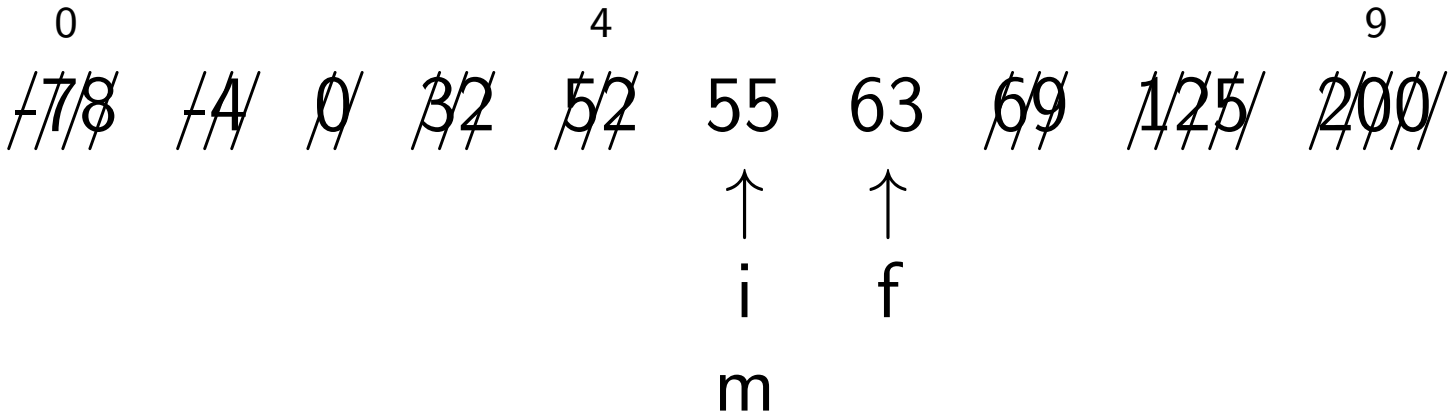
Busca Binária

- 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



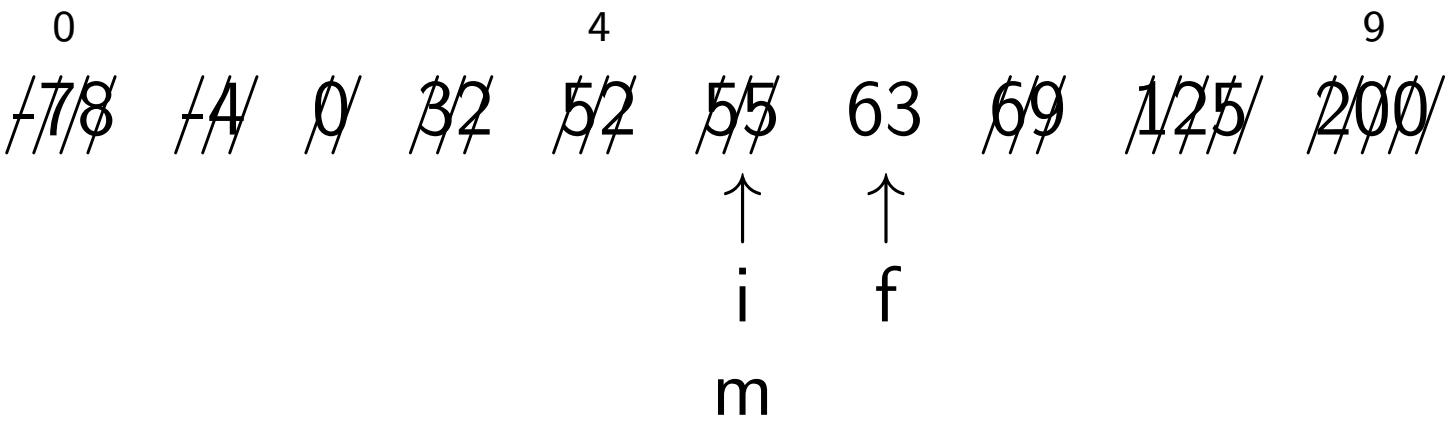
Busca Binária

- 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



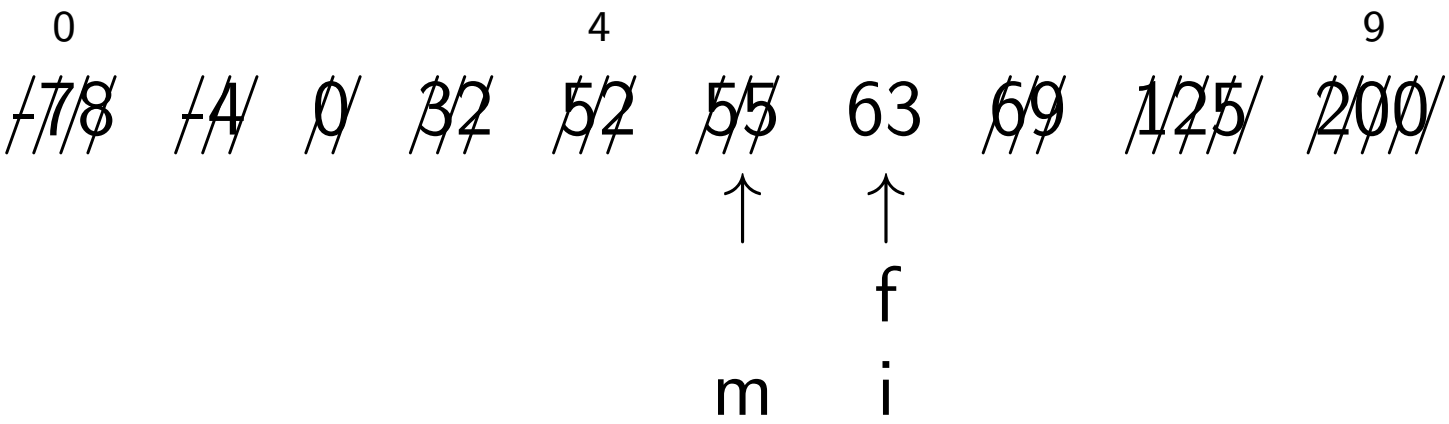
Busca Binária

- 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



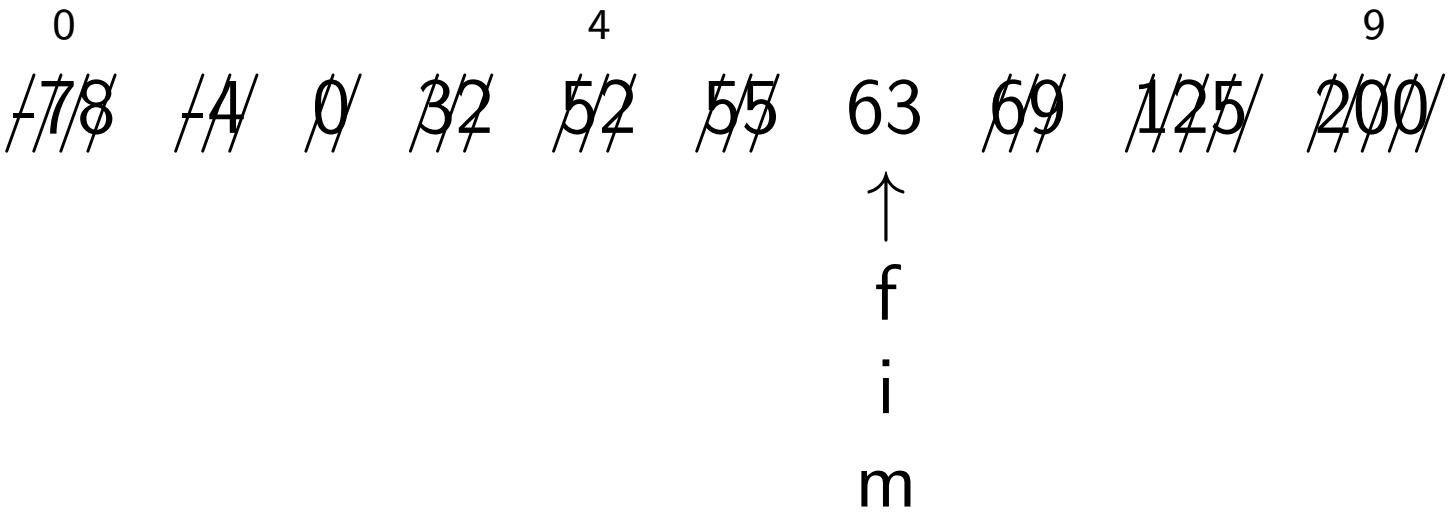
Busca Binária

- 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



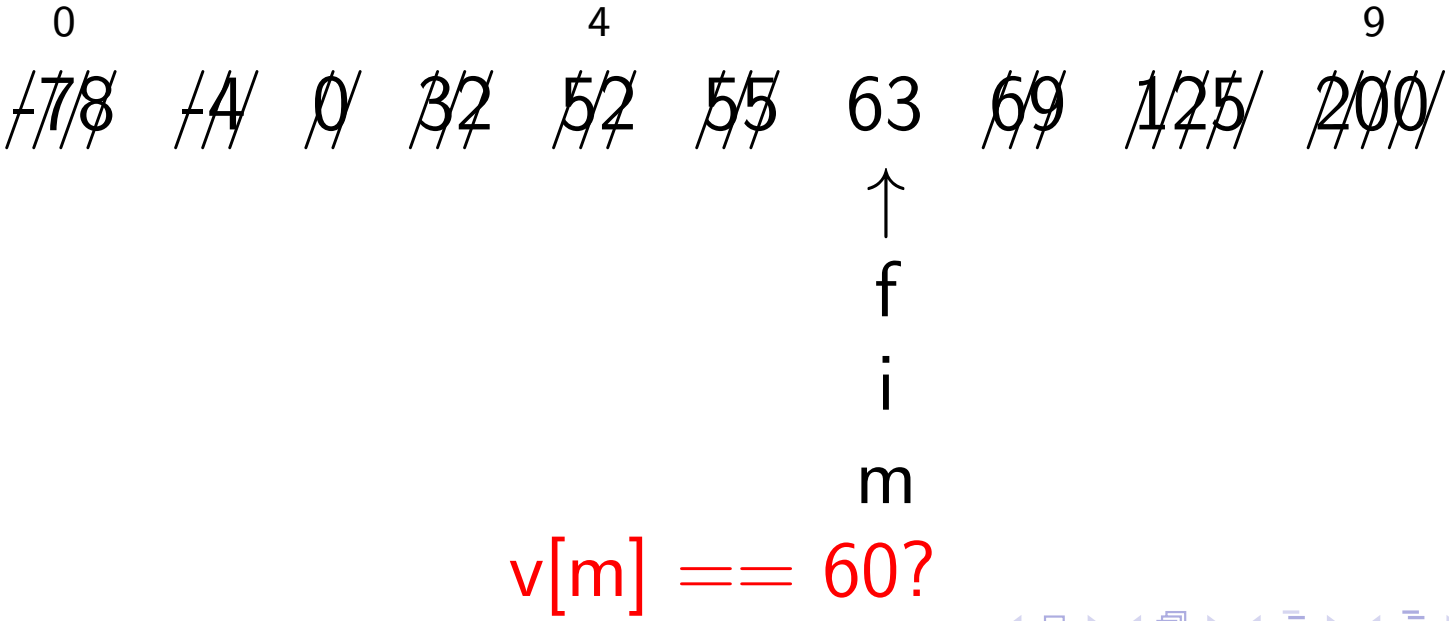
Busca Binária

- 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



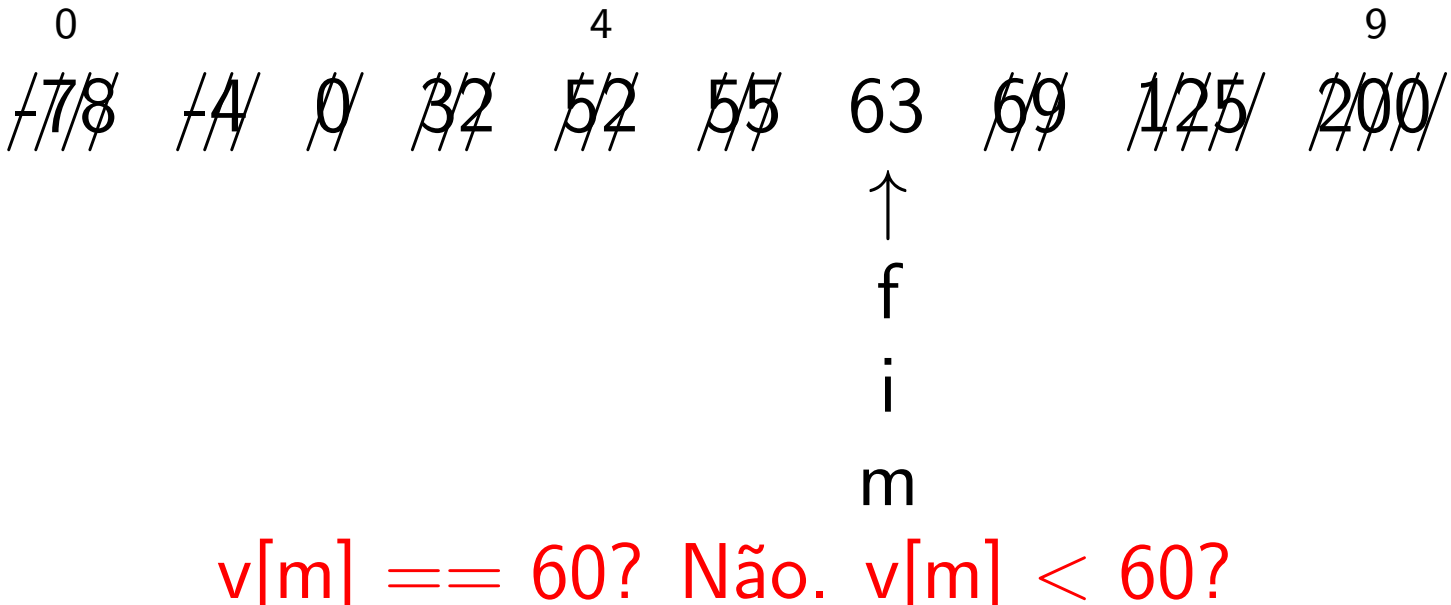
Busca Binária

- 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



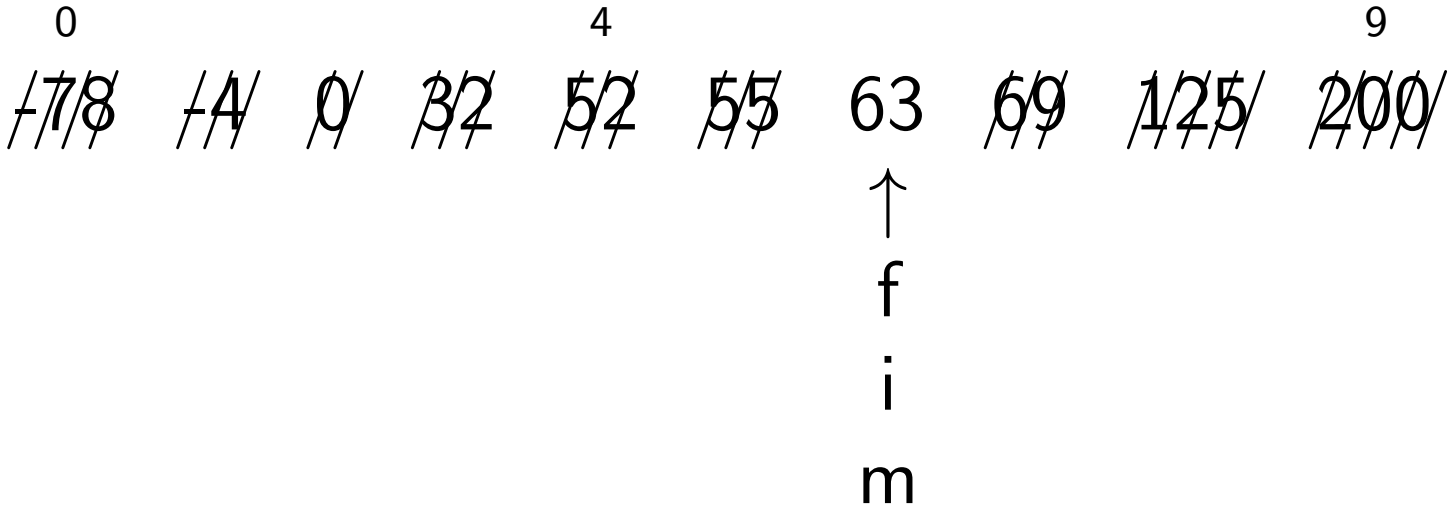
Busca Binária

- 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

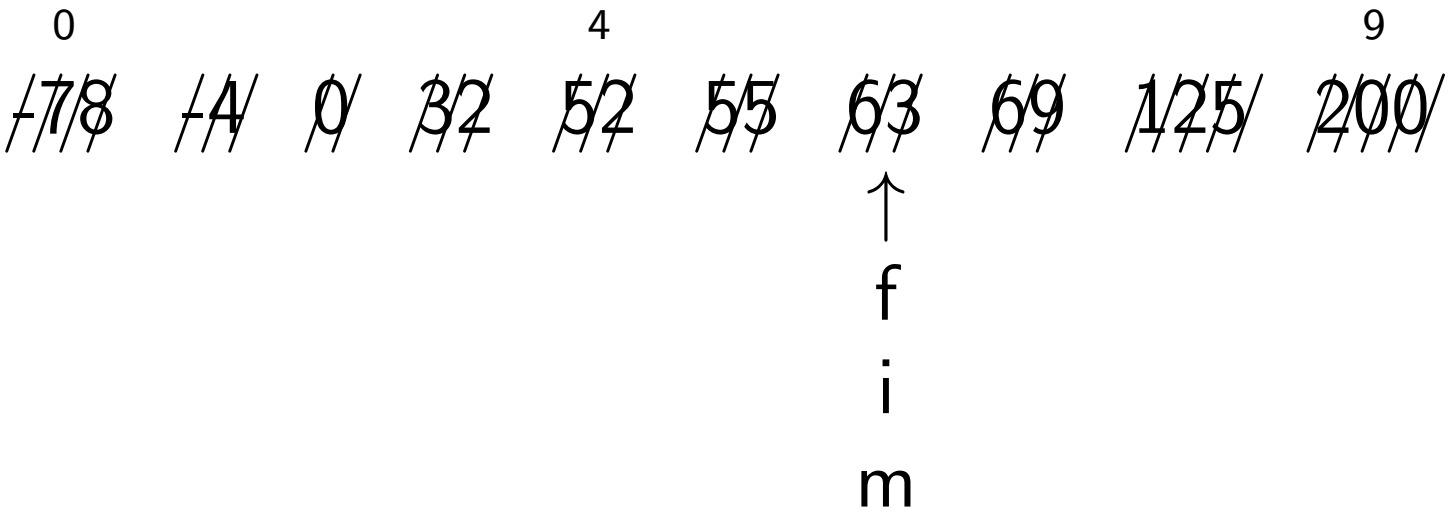
- Algoritmo:
  - Verifique se o elemento buscado é o do meio do arranjo
  - Se não for, verifique se é maior
    - Se for, repita a busca na metade direita do arranjo
    - Se não for, repita a busca na metade esquerda do arranjo
- Ex: Buscando 60



v[m] == 60? Não. v[m] < 60? Não

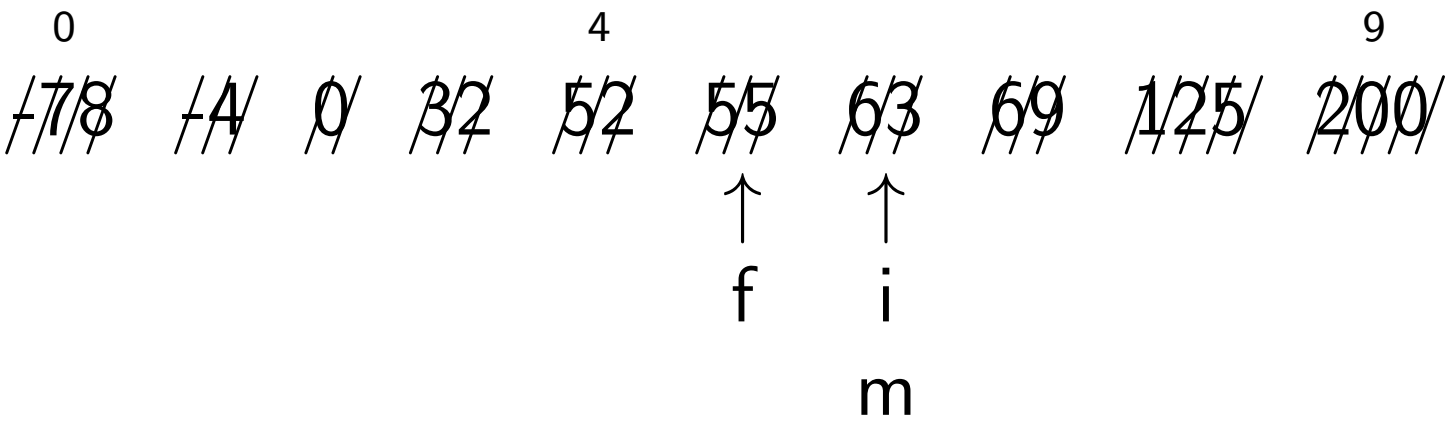
Busca Binária

- 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 60



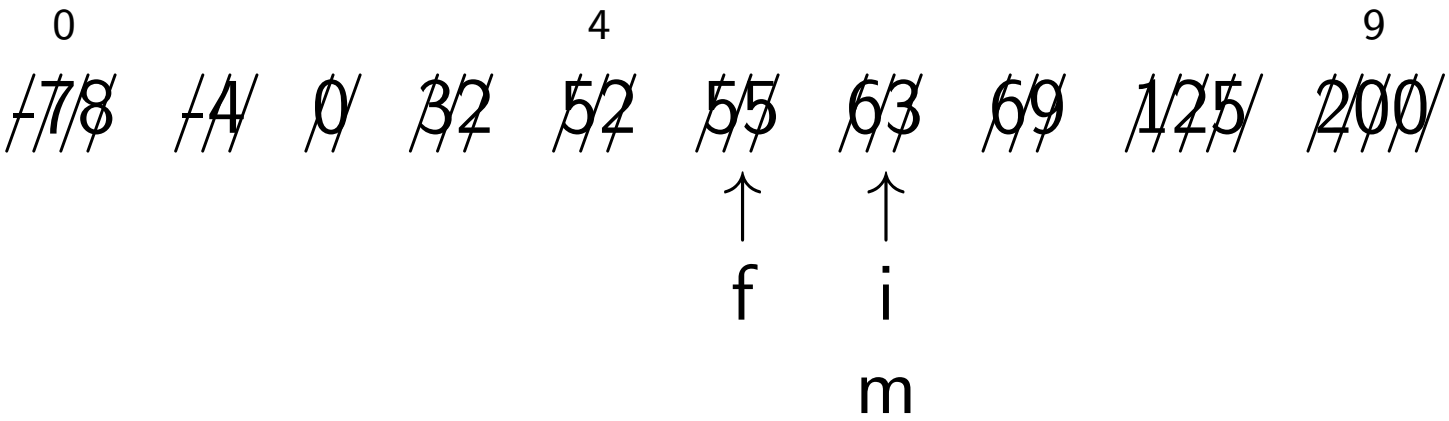
Busca Binária

- 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...

```
static int buscaBin(int[] arr, int el)
{
  int fim = arr.length-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

```
static int buscaBin(int[] arr, int el)
{
  int fim = arr.length-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
- E sua versão com objetos?

```
static int buscaBin(int[] arr, int el)
{
  int fim = arr.length-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

```
static int buscaBin(Residencia[] arr,
                    double area) {
  int fim = arr.length-1;
  int ini = 0;
  while (ini <= fim) {
    int meio = (fim + ini)/2;
    if (arr[meio].area() < area)
      ini = meio + 1;
    else
      if (arr[meio].area() > area)
        fim = meio-1;
      else return(meio);
  }
  return(-1);
}
```

Busca Binária

```
static int buscaBin(Residencia[] arr,
                    double area) {
  int fim = arr.length-1;
  int ini = 0;
  while (ini <= fim) {
    int meio = (fim + ini)/2;
    if (arr[meio].area() < area)
      ini = meio + 1;
    else
      if (arr[meio].area() > area)
        fim = meio-1;
      else return(meio);
  }
  return(-1);
}
```

- Note os nomes iguais nas 2 versões

Busca Binária

```
static int buscaBin(Residencia[] arr,
                    double area) {
  int fim = arr.length-1;
  int ini = 0;
  while (ini <= fim) {
    int meio = (fim + ini)/2;
    if (arr[meio].area() < area)
      ini = meio + 1;
    else
      if (arr[meio].area() > area)
        fim = meio-1;
      else return(meio);
  }
  return(-1);
}
```

- Note os nomes iguais nas 2 versões
  - Funciona porque os parâmetros são diferentes

Busca Binária

```
static int buscaBin(Residencia[] arr,
                    double area) {
  int fim = arr.length-1;
  int ini = 0;
  while (ini <= fim) {
    int meio = (fim + ini)/2;
    if (arr[meio].area() < area)
      ini = meio + 1;
    else
      if (arr[meio].area() > area)
        fim = meio-1;
      else return(meio);
  }
  return(-1);
}
```

- Note os nomes iguais nas 2 versões
  - Funciona porque os parâmetros são diferentes
- Nesse caso, procuramos alguma residência com uma determinada área

Busca Binária

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

Busca Binária

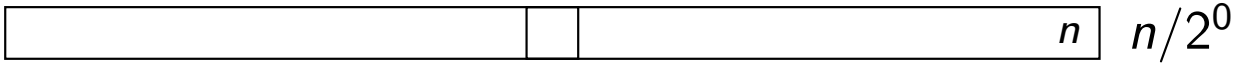
- Sabemos que fazemos um máximo de n comparações com busca sequencial
  - Onde n é o número de elementos do arranjo

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?

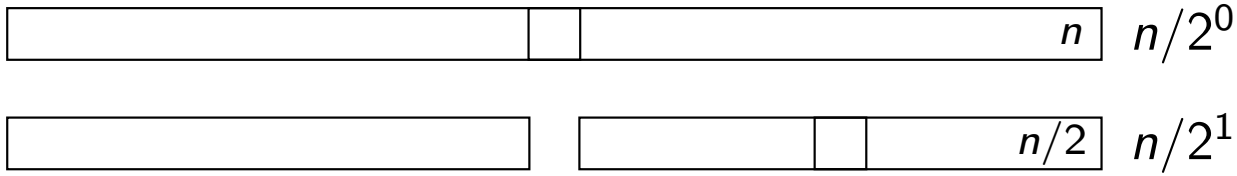
Busca Binária

- 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?



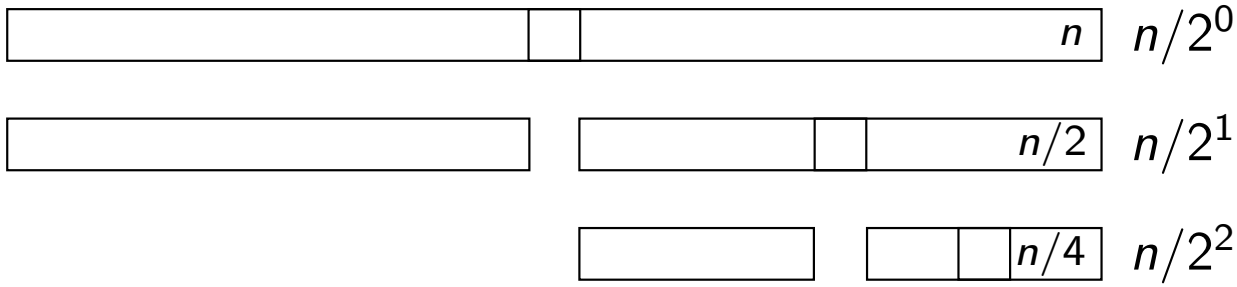
Busca Binária

- 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?



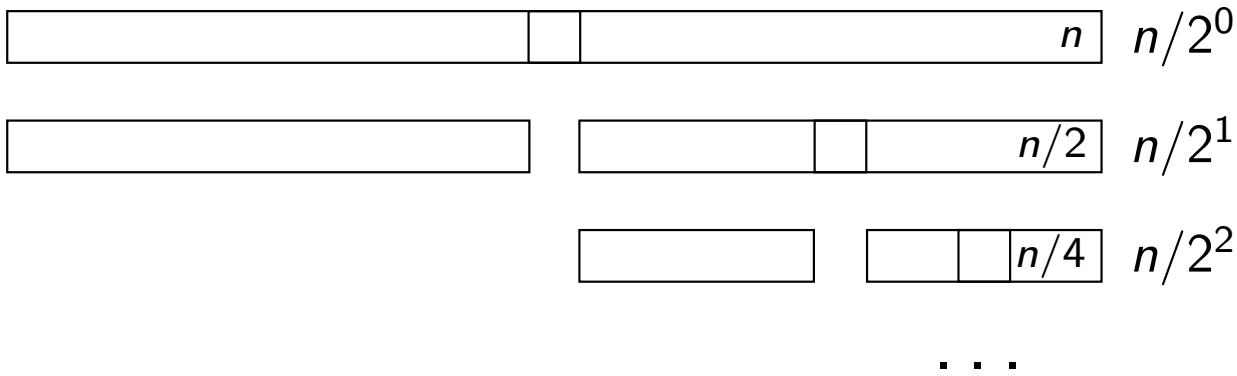
Busca Binária

- 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?



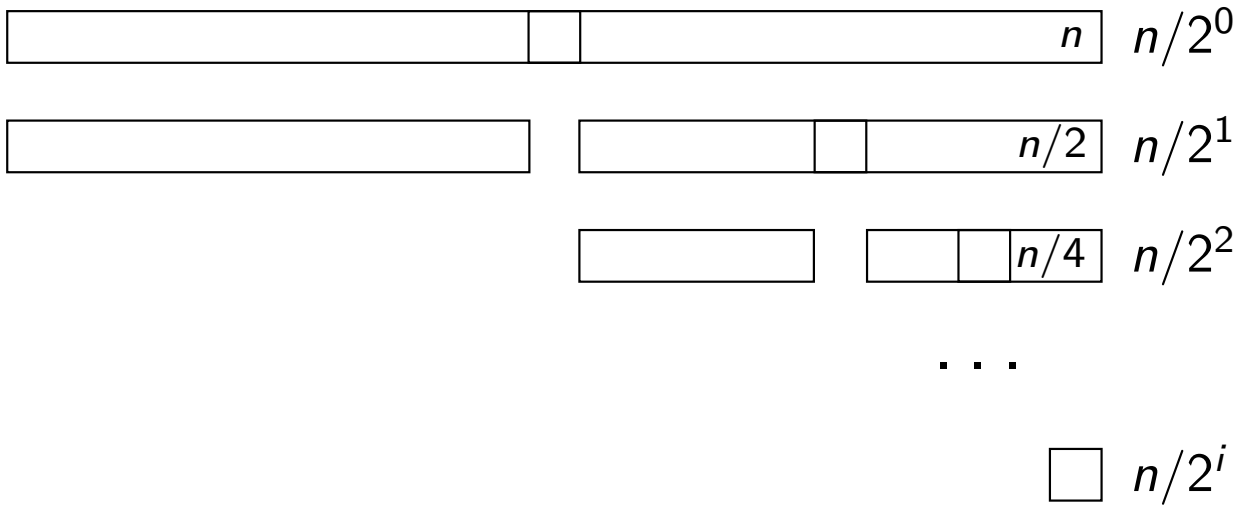
Busca Binária

- 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?

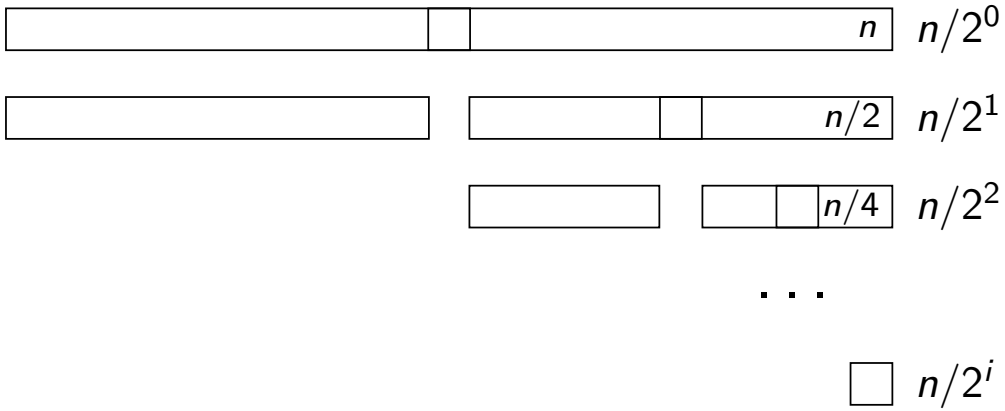


Busca Binária

- 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?

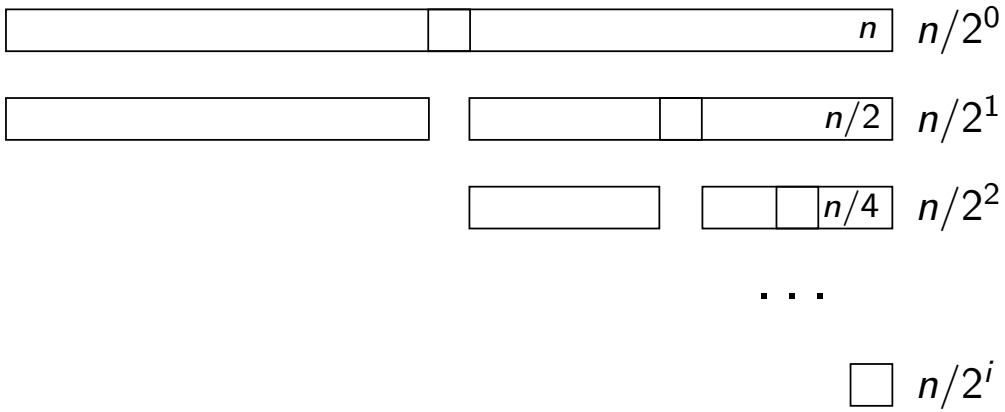


Busca Binária



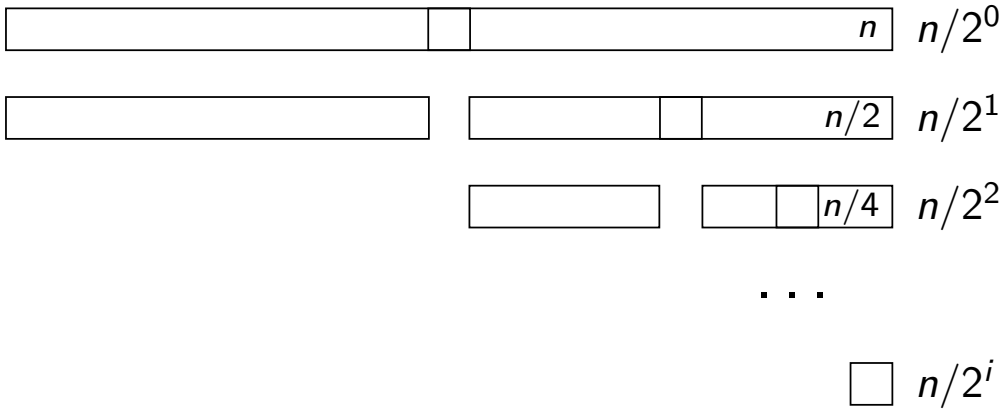
- Temos $i + 1$ comparações, sendo a última feita com o arranjo de tamanho 1

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.

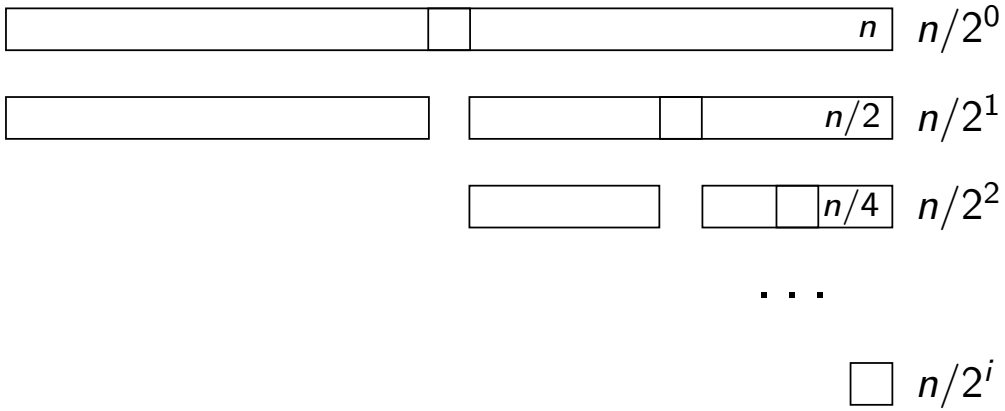
Busca Binária



- 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 freqüentes

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 freqüentes
- Em suma, quando a ordem não é mudada com freqüência

Quando Usar Busca Binária?

Chave de busca?

- A chave de busca é o que se quer buscar

Quando Usar Busca Binária?

Chave de busca?

- A chave de busca é o que se quer buscar
- Ex: no caso das residências, era a área

Quando Usar Busca Binária?

Chave de busca?

- A chave de busca é o que se quer buscar
- Ex: no caso das residências, era a área
- Se a área ficar mudando, muda a ordem e o arranjo deve ser reordenado

Quando Usar Busca Binária?

Chave de busca?

- A chave de busca é o que se quer buscar
- Ex: no caso das residências, era a área
- Se a área ficar mudando, muda a ordem e o arranjo deve ser reordenado
- Gera um custo extra que acaba se sobrepondo ao ganho com a busca

Mistério

- O que este código faz? Tente com `v = {4,1,3,5}`

```
static void ???(int[] v) {
  for (int ult = v.length-1; ult>0; ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}
```

Mistério

- O que este código faz? Tente com v = {4,1,3,5}

```
static void ???(int[] v) {
  for (int ult = v.length-1; ult>0; ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}
```

Ordenou v

Método da Bolha (Bubble Sort)

```
static void bolha(int[] v) {
  for (int ult = v.length-1;
                      ult>0; ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}
```

Método da Bolha (Bubble Sort)

- Percorra todo o arranjo tomando seus elementos adjacentes para a par

```
static void bolha(int[] v) {
  for (int ult = v.length-1;
                        ult>0; ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}
```

Método da Bolha (Bubble Sort)

- Percorra todo o arranjo tomando seus elementos adjacentes para a par
- Se os elemento no par estiverem ordenados, siga ao próximo par

```
static void bolha(int[] v) {
  for (int ult = v.length-1;
                       ult>0; ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}
```

Método da Bolha (Bubble Sort)

- Percorra todo o arranjo tomando seus elementos adjacentes para a par
- Se os elemento no par estiverem ordenados, siga ao próximo par
- Senão, troque-os de lugar

```
static void bolha(int[] v) {
  for (int ult = v.length-1;
                       ult>0; ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}
```

Método da Bolha (Bubble Sort)

- Percorra todo o arranjo tomando seus elementos adjacentes para a par
- Se os elemento no par estiverem ordenados, siga ao próximo par
- Senão, troque-os de lugar
- Repita a operação até que nenhuma troca possa ser feita no arranjo inteiro

```
static void bolha(int[] v) {
  for (int ult = v.length-1;
                        ult>0; ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}
```

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

9 8 4 6 3

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

9 8 4 6 3

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$\underline{9 \quad 8} \quad 4 \quad 6 \quad 3$$

- Comparando os dois primeiros números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$\underline{8 \quad 9} \quad 4 \quad 6 \quad 3$$

- Trocando porque $8 < 9$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$8 \quad \underline{9 \quad 4} \quad 6 \quad 3$$

- Comparando segundo e terceiro números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$8 \quad \underline{4 \quad 9} \quad 6 \quad 3$$

- Trocando porque $4 < 9$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$8 \quad 4 \quad \underline{9 \quad 6} \quad 3$$

- Comparando terceiro e quarto números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$8 \quad 4 \quad \underline{6 \quad 9} \quad 3$$

- Trocando porque $6 < 9$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$8 \quad 4 \quad 6 \quad \underline{9 \quad 3}$$

- Comparando quarto e quinto números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$8 \quad 4 \quad 6 \quad \underline{3 \quad 9}$$

- Trocando porque $3 < 9$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Primeira passada completa. Último elemento fixado:Executando a segunda passada:

$$8 \quad 4 \quad 6 \quad \underline{3 \quad \mathbf{9}}$$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

8	4	6	3	**9**
8	4	6	3	**9**

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{ccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ \underline{8} & \underline{4} & 6 & 3 & \mathbf{9} \end{array}$$

- Comparando os dois primeiros números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{ccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ \underline{4} & \underline{8} & 6 & 3 & \mathbf{9} \end{array}$$

- Trocando porque $4 < 8$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{cccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & \underline{8} & \underline{6} & 3 & \mathbf{9} \end{array}$$

- Comparando segundo e terceiro números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{cccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & \underline{6} & \underline{8} & 3 & \mathbf{9} \end{array}$$

- Trocando porque $6<8$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{ccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & 6 & \underline{8} & \underline{3} & \mathbf{9} \end{array}$$

- Comparando terceiro e quarto números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{cccc} 8 & 4 & 6 & 3 \\ 4 & 6 & 3 & 8 \end{array} \quad \mathbf{9}$$

- Trocando porque $3<8$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Segunda passada completa. Último elemento fixado:

$$\begin{array}{cccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & 6 & 3 & \mathbf{8} & \mathbf{9} \end{array}$$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

8	4	6	3	**9**
4	6	3	**8**	**9**
4	6	3	**8**	**9**

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

$$\begin{array}{cccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & 6 & 3 & \mathbf{8} & \mathbf{9} \\ 4 & 6 & 3 & \mathbf{8} & \mathbf{9} \end{array}$$

- Comparando os dois primeiros números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

8	4	6	3	**9**
4	6	3	**8**	**9**
4	6	3	**8**	**9**

- 6>4, logo não há necessidade de troca

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

$$\begin{array}{ccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & 6 & 3 & \mathbf{8} & \mathbf{9} \\ 4 & 6 & 3 & \mathbf{8} & \mathbf{9} \end{array}$$

- Comparando os segundo e terceiro números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

$$\begin{array}{ccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & 6 & 3 & \mathbf{8} & \mathbf{9} \\ 4 & 3 & 6 & \mathbf{8} & \mathbf{9} \end{array}$$

- Trocando porque $3<6$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Terceira passada completa. Último elemento fixado:

$$\begin{array}{ccccc} 8 & 4 & 6 & 3 & \mathbf{9} \\ 4 & 6 & 3 & \mathbf{8} & \mathbf{9} \\ 4 & 3 & \mathbf{6} & \mathbf{8} & \mathbf{9} \end{array}$$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a quarta passada:

8	4	6	3	**9**
4	6	3	**8**	**9**
4	3	**6**	**8**	**9**
4	3	**6**	**8**	**9**

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a quarta passada:

8	4	6	3	**9**
4	6	3	**8**	**9**
4	3	**6**	**8**	**9**
4	3	**6**	**8**	**9**

- Comparando os dois primeiros números

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Executando a quarta passada:

8	4	6	3	**9**
4	6	3	**8**	**9**
4	3	**6**	**8**	**9**
3	4	**6**	**8**	**9**

- Trocando porque $3<4$

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Quarta passada completa. Último elemento fixado:

8	4	6	3	**9**
4	6	3	**8**	**9**
4	3	**6**	**8**	**9**
3	**4**	**6**	**8**	**9**

Método da Bolha (Bubble Sort)

Ex: ordene em ordem crescente

- Quarta passada completa. Último elemento fixado:

8	4	6	3	**9**
4	6	3	**8**	**9**
4	3	**6**	**8**	**9**
3	**4**	**6**	**8**	**9**

- Há somente um elemento no arranjo. O algoritmo para. O arranjo está ordenado.

Método da Bolha (Bubble Sort)

- Observe que não precisamos correr sempre o arranjo até o final

```
static void bolha(int[] v) {
  for (int ult = v.length-1; ult>0;
                                    ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}

public static void main(String[] args) {
  int[] v = {55,0,-78,-4,32,200,52,63,
                                  69,125};

  bolha(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Método da Bolha (Bubble Sort)

- Observe que não precisamos correr sempre o arranjo até o final
  - Basta irmos até onde garantimos estar ordenado...

```
static void bolha(int[] v) {
  for (int ult = v.length-1; ult>0;
                                    ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}

public static void main(String[] args) {
  int[] v = {55,0,-78,-4,32,200,52,63,
                                  69,125};

  bolha(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Método da Bolha (Bubble Sort)

- Observe que não precisamos correr sempre o arranjo até o final
  - Basta irmos até onde garantimos estar ordenado...
  - Marcando esse final

```
static void bolha(int[] v) {
  for (int ult = v.length-1; ult>0;
                                    ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}

public static void main(String[] args) {
  int[] v = {55,0,-78,-4,32,200,52,63,
                                   69,125};

  bolha(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Método da Bolha (Bubble Sort)

- Observe que não precisamos correr sempre o arranjo até o final
  - Basta irmos até onde garantimos estar ordenado...
  - Marcando esse final
  - Sempre corremos de 0 a `ult`

```
static void bolha(int[] v) {
  for (int ult = v.length-1; ult>0;
                                    ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}

public static void main(String[] args) {
  int[] v = {55,0,-78,-4,32,200,52,63,
                                  69,125};

  bolha(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Método da Bolha (Bubble Sort)

- Observe que não precisamos correr sempre o arranjo até o final
  - Basta irmos até onde garantimos estar ordenado...
  - Marcando esse final
  - Sempre corremos de 0 a `ult`
  - E a cada passada, decrementamos `ult`

```
static void bolha(int[] v) {
  for (int ult = v.length-1; ult>0;
                                    ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}

public static void main(String[] args) {
  int[] v = {55,0,-78,-4,32,200,52,63,
                                   69,125};

  bolha(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Método da Bolha (Bubble Sort)

Saída

```
$ java Projeto
-78 -4 0 32 52 55 63 69 125 200
```

```
static void bolha(int[] v) {
  for (int ult = v.length-1; ult>0;
                                    ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}

public static void main(String[] args) {
  int[] v = {55,0,-78,-4,32,200,52,63,
                                  69,125};

  bolha(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Método da Bolha (Bubble Sort)

Saída

```
$ java Projeto
-78 -4 0 32 52 55 63 69 125 200
```

Cuidado!

Esse método modifica o arranjo original!

```
static void bolha(int[] v) {
  for (int ult = v.length-1; ult>0;
                                    ult--)
    for (int i=0; i<ult; i++)
      if (v[i] > v[i+1]) {
        int aux = v[i];
        v[i] = v[i+1];
        v[i+1] = aux;
      }
}

public static void main(String[] args) {
  int[] v = {55,0,-78,-4,32,200,52,63,
                                  69,125};

  bolha(v);

  for (int el : v)
    System.out.print(el+" ");
  System.out.println();
}
```

Método da Bolha (Bubble Sort)

- E se em vez de inteiros quisermos usar objetos?

Método da Bolha (Bubble Sort)

- E se em vez de inteiros quisermos usar objetos?
  - Como nossa `Residencia`

Método da Bolha (Bubble Sort)

- E se em vez de inteiros quisermos usar objetos?
  - Como nossa `Residencia`
- O problema está em comparar os objetos

Método da Bolha (Bubble Sort)

- E se em vez de inteiros quisermos usar objetos?
  - Como nossa `Residencia`
- O problema está em comparar os objetos
  - Não podemos fazer `obj1 == obj2`

Método da Bolha (Bubble Sort)

- E se em vez de inteiros quisermos usar objetos?
  - Como nossa `Residencia`
- O problema está em comparar os objetos
  - Não podemos fazer `obj1 == obj2`
- Devemos comparar com base em algum atributo

Método da Bolha (Bubble Sort)

- E se em vez de inteiros quisermos usar objetos?
  - Como nossa `Residencia`
- O problema está em comparar os objetos
  - Não podemos fazer `obj1 == obj2`
- Devemos comparar com base em algum atributo
  - Por exemplo, a área total

Método da Bolha (Bubble Sort)

- E se em vez de inteiros quisermos usar objetos?
  - Como nossa `Residencia`
- O problema está em comparar os objetos
  - Não podemos fazer `obj1 == obj2`
- Devemos comparar com base em algum atributo
  - Por exemplo, a área total
- Alternativamente, podemos definir um método que compare objetos com base nesse atributo

Método da Bolha (Bubble Sort)

Alternativa 1

```
class Residencia {
  ...

  static void bolha(Residencia[] v) {
    for (int ult = v.length-1; ult>0;
                                    ult--)
      for (int i=0; i<ult; i++)
        if (v[i].area() > v[i+1].area()){
          Residencia aux = v[i];
          v[i] = v[i+1];
          v[i+1] = aux;
        }
  }
```

```
  public static void main(String[] args){
    AreaCasa c = new AreaCasa();
    AreaPiscina p = new AreaPiscina();
    Residencia r1 = new Residencia(c,p);

    c = new AreaCasa();
    p = new AreaPiscina(11);
    Residencia r2 = new Residencia(c,p);

    System.out.println(r1.comparaRes(r2));
  }
}
```

Método da Bolha (Bubble Sort)

Alternativa 2 – mais geral

```
class Residencia {
  ...
  double comparaRes(Residencia outra) {
    if (outra == null) return(1);
    return(this.area() - outra.area());
  }

  static void bolha(Residencia[] v) {
    for (int ult = v.length-1; ult>0;
                                    ult--)
      for (int i=0; i<ult; i++)
        if (v[i].comparaRes(v[i+1]) > 0){
          Residencia aux = v[i];
          v[i] = v[i+1];
          v[i+1] = aux;
        }
  }
```

```
  public static void main(String[] args){
    AreaCasa c = new AreaCasa();
    AreaPiscina p = new AreaPiscina();
    Residencia r1 = new Residencia(c,p);

    c = new AreaCasa();
    p = new AreaPiscina(11);
    Residencia r2 = new Residencia(c,p);

    System.out.println(r1.comparaRes(r2));
  }
}
```

Método da Bolha – Exemplo de Uso

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);
  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(
      Math.random()*100,Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                  Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  bolha(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Método da Bolha – Exemplo de Uso

- Math.random()?

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);
  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(
      Math.random()*100,Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                  Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  bolha(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Método da Bolha – Exemplo de Uso

- Math.random()?
  - Gera um número pseudo-aleatório $0 \leq n < 1$

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);
  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(
      Math.random()*100,Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                  Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  bolha(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Método da Bolha – Exemplo de Uso

- Math.random()?
  - Gera um número pseudo-aleatório $0 \leq n < 1$

Saída

```
$ java Projeto
2055.600048644749
949.1972834436008
3316.7903140566305
6698.682789640099
584.1255507074843

584.1255507074843
949.1972834436008
2055.600048644749
3316.7903140566305
6698.682789640099
```

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);
  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(
      Math.random()*100,Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                  Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  bolha(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Ordenação por Seleção (Selection Sort)

Algoritmo:

Ordenação por Seleção (Selection Sort)

Algoritmo:

- Primeiro encontre o menor elemento do arranjo

Ordenação por Seleção (Selection Sort)

Algoritmo:

- Primeiro encontre o menor elemento do arranjo
- Então troque esse elemento de lugar com o que está na primeira posição

Ordenação por Seleção (Selection Sort)

Algoritmo:

- Primeiro encontre o menor elemento do arranjo
- Então troque esse elemento de lugar com o que está na primeira posição
- Encontre o segundo menor do arranjo

Ordenação por Seleção (Selection Sort)

Algoritmo:

- Primeiro encontre o menor elemento do arranjo
- Então troque esse elemento de lugar com o que está na primeira posição
- Encontre o segundo menor do arranjo
- Troque com o da segunda posição

Ordenação por Seleção (Selection Sort)

Algoritmo:

- Primeiro encontre o menor elemento do arranjo
- Então troque esse elemento de lugar com o que está na primeira posição
- Encontre o segundo menor do arranjo
- Troque com o da segunda posição
- E assim por diante, até chegar ao fim do arranjo

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

9 8 4 6 3

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$\begin{array}{ccccc} 9 & 8 & 4 & 6 & 3 \\ \underline{9} & 8 & 4 & 6 & 3 \end{array}$$

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$\begin{array}{ccccc} 9 & 8 & 4 & 6 & 3 \\ \underline{9} & 8 & 4 & 6 & \underline{3} \end{array}$$

- Encontrando o menor elemento

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a primeira passada:

$$\begin{array}{ccccc} 9 & 8 & 4 & 6 & 3 \\ \underline{3} & 8 & 4 & 6 & \underline{9} \end{array}$$

- Trocando com o primeiro elemento, pois $3 < 9$

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Primeira passada completa. Primeiro elemento fixado:

9	8	4	6	3
3	8	4	6	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

9	8	4	6	3
3	8	4	6	9
3	8	4	6	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{ccccc} 9 & 8 & 4 & 6 & 3 \\ \mathbf{3} & 8 & 4 & 6 & 9 \\ \mathbf{3} & \underline{8} & \underline{4} & 6 & 9 \end{array}$$

- Encontrando o segundo menor elemento

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a segunda passada:

$$\begin{array}{ccccc} 9 & 8 & 4 & 6 & 3 \\ \mathbf{3} & 8 & 4 & 6 & 9 \\ \mathbf{3} & 4 & 8 & 6 & 9 \end{array}$$

- Trocando com o segundo elemento, pois $4 < 8$

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Segunda passada completa. Segundo elemento fixado:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	8	6	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	8	6	9

- Encontrando o terceiro menor elemento

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a terceira passada:

$$\begin{array}{ccccc} 9 & 8 & 4 & 6 & 3 \\ \mathbf{3} & 8 & 4 & 6 & 9 \\ \mathbf{3} & \mathbf{4} & 8 & 6 & 9 \\ \mathbf{3} & \mathbf{4} & 6 & 8 & 9 \end{array}$$

- Trocando com o terceiro elemento, pois $6 < 8$

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Terceira passada completa. Terceiro elemento fixado:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a quarta passada:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9
3	**4**	**6**	8	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a quarta passada:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9
3	**4**	**6**	8	9

- Encontrando o quarto menor elemento

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a quarta passada:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9
3	**4**	**6**	8	9

- Não há troca, pois já está na quarta posição

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Quarta passada completa. Quarto elemento fixado:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9
3	**4**	**6**	**8**	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a quinta passada:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9
3	**4**	**6**	**8**	9
3	**4**	**6**	**8**	9

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Executando a quinta passada:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9
3	**4**	**6**	**8**	9
3	**4**	**6**	**8**	9

- Última posição do arranjo. O algoritmo para.

Ordenação por Seleção (Selection Sort)

Ex: ordene em ordem crescente

- Quinta passada completa. Quinto elemento fixado:

9	8	4	6	3
3	8	4	6	9
3	**4**	8	6	9
3	**4**	**6**	8	9
3	**4**	**6**	**8**	9
3	**4**	**6**	**8**	**9**

Ordenação por Seleção (Selection Sort)

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                  && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- Criamos um método que diz a posição do menor elemento em subvetor
  $inicio \leq i < fim$

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                    && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- Criamos um método que diz a posição do menor elemento em subvetor
  $inicio \leq i < fim$
  - Sempre é bom testar a entrada

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- Para cada elemento do arranjo (exceto o último, que sobra já ordenado)

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                  && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- Para cada elemento do arranjo (exceto o último, que sobra já ordenado)
  - Busca o menor elemento a partir deste

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                  && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- Para cada elemento do arranjo (exceto o último, que sobra já ordenado)
  - Busca o menor elemento a partir deste
  - Troca com a posição desse elemento

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                  && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- E como fazer um *Selection Sort* sem método auxiliar?

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                  && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- E como fazer um *Selection Sort* sem método auxiliar?

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                  && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- E como fazer um *Selection Sort* sem método auxiliar?

```
static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor =  i;
    for (int p=i+1; p<v.length; p++)
      if (v[p] < v[posMenor])
        posMenor = p;
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

```
static int posMenorEl(int[] v, int inicio) {
  int posMenor = -1;
  if ((v!=null) && (inicio>=0)
                  && (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i] < v[posMenor]) posMenor = i;
  }
  return(posMenor);
}

static void selecao(int[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    int aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

- E como fica com objetos?

Ordenação por Seleção (Selection Sort)

- E como fica com objetos?

```
static int posMenorEl(Residencia[] v, int inicio)
{
  int posMenor = -1;
  if ((v!=null) && (inicio>=0) &&
                              (inicio < v.length)) {
    posMenor = inicio;
    for (int i=inicio+1; i<v.length; i++)
      if (v[i].comparaRes(v[posMenor]) < 0)
        posMenor = i;
  }
  return(posMenor);
}

static void selecao(Residencia[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = posMenorEl(v,i);
    Residencia aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Ordenação por Seleção (Selection Sort)

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);

  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(Math.random()*100,
                              Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                              Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  selecao(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Ordenação por Seleção (Selection Sort)

Saída

```
$ java Projeto
1886.209373681862
88.04997084519657
10007.673022091289
370.5687127650107
2084.230748531825

88.04997084519657
370.5687127650107
1886.209373681862
2084.230748531825
10007.673022091289
```

```
public static void main(String[] args) {
  Projeto pr = new Projeto(5);

  for (int i=0; i<5; i++) {
    AreaCasa c = new AreaCasa(Math.random()*100,
                              Math.random()*30);
    AreaPiscina p = new AreaPiscina(
                              Math.random()*10);
    Residencia r = new Residencia(c,p);
    pr.adicionaRes(r);
  }

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
  System.out.println();

  selecao(pr.condominio);

  for (Residencia r : pr.condominio)
    System.out.println(r.area());
}
```

Ordenação por Seleção (Selection Sort)

- E sem o método auxiliar...

```
static void selecao(Residencia[] v) {
  for (int i=0; i<v.length-1; i++) {
    int posMenor = i;
    for (int p=i+1; p<v.length; p++)
      if (v[p].comparaRes(v[posMenor]) < 0)
          posMenor = p;
    Residencia aux = v[i];
    v[i] = v[posMenor];
    v[posMenor] = aux;
  }
}
```

Curiosidades

- Bolha:
  - `http://www.youtube.com/watch?v=lyZQPjUT5B4`
- Seleção:
  - `http://www.youtube.com/watch?v=Ns4TPTC8whw` (versão levemente diferente do algoritmo)

Videoaula

https://www.youtube.com/watch?v=8weGr_G3Pqo

https://www.youtube.com/watch?v=Yw_QLLlIsII

e

https://www.youtube.com/watch?v=pnGhdy5B02I