Linguagem de Programação
Ordenação e Busca em Vetores

Alexandre Mello

Fatec Campinas

2023

Roteiro

Ordenação

- Vamos estudar alguns algoritmos para o seguinte problema:

Dado uma coleção de elementos com uma relação de ordem entre si, devemos gerar uma saída com os elementos ordenados.

- Nos nossos exemplos usaremos um vetor de inteiros para representar tal coleção.
  - É claro que quaisquer inteiros possuem uma relação de ordem entre si.
- Apesar de usarmos inteiros, os algoritmos servem para ordenar qualquer coleção de elementos que possam ser comparados.

Ordenação

- O problema de ordenação é um dos mais básicos em computação.
  - Mas muito provavelmente é um dos problemas com o maior número de aplicações diretas ou indiretas (como parte da solução para um problema maior).
- Exemplos de aplicações diretas:
  - Criação de *rankings*, Definir preferências em atendimentos por prioridade, Criação de Listas etc.
- Exemplos de aplicações indiretas:
  - Otimizar sistemas de busca, manutenção de estruturas de bancos de dados etc.

Selection Sort

- Seja **vet** um vetor contendo números inteiros.
- Devemos deixar **vet** em ordem crescente.
- A idéia do algoritmo é a seguinte:
  - Ache o menor elemento a partir da posição 0. Troque então este elemento com o elemento da posição 0.
  - Ache o menor elemento a partir da posição 1. Troque então este elemento com o elemento da posição 1.
  - Ache o menor elemento a partir da posição 2. Troque então este elemento com o elemento da posição 2.
  - E assim sucessivamente...

Selection-Sort

Exemplo: (5,3,2,1,90,6).

Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).

Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).

Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).

Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).

Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

Exemplo: (5,3,2,1,90,6).
Iteração 0. Acha menor: (5,3,2,1,90,6). Faz troca: (1,3,2,5,90,6).
Iteração 1. Acha menor: (1,3,2,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 2. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 3. Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,90,6).
Iteração 5: Acha menor: (1,2,3,5,90,6). Faz troca: (1,2,3,5,6,90).

Selection-Sort

- Como achar o menor elemento a partir de uma posição inicial?
- Vamos achar **o índice** do menor elemento em um vetor, a partir de uma posição inicial **ini**:

```
int min = ini, j;
for(j=ini+1; j<tam; j++){
   if(vet[min] > vet[j])
      min = j;
}
```

Selection-Sort

- Como achar o menor elemento a partir de uma posição inicial?
- Vamos achar **o índice** do menor elemento em um vetor, a partir de uma posição inicial **ini**:

```
int min = ini, j;
for(j=ini+1; j<tam; j++){
  if(vet[min] > vet[j])
    min = j;
}
```

Selection-Sort

- Criamos então uma função que retorna o índice do elemento mínimo de um vetor, a partir de uma posição **ini** passada por parâmetro:

```
int indiceMenor(int vet[], int tam, int ini){
  int min = ini, j;
  for(j=ini+1; j<tam; j++){
    if(vet[min] > vet[j])
      min = j;
  }
  return min;
}
```

Selection-Sort

- Dada a função anterior para achar o índice do menor elemento, como implementar o algoritmo de ordenação?
- Ache o menor elemento a partir da posição 0, e troque com o elemento da posição 0.
- Ache o menor elemento a partir da posição 1, e troque com o elemento da posição 1.
- Ache o menor elemento a partir da posição 2, e troque com o elemento da posição 2.
- E assim sucessivamente...

Selection-Sort

```
void selectionSort(int vet[], int tam){

  int i, min, aux;

  for(i=0; i<tam; i++){
    min = indiceMenor(vet, tam, i); //Acha posicao do menor elemento
    aux = vet[i];                   //a partir de i
    vet[i] = vet[min];
    vet[min] = aux;
  }
}
```

Selection-Sort

```
void selectionSort(int vet[], int tam){

  int i, min, aux;

  for(i=0; i<tam; i++){
    min = indiceMenor(vet, tam, i); //Acha posicao do menor elemento
    aux = vet[i];                   //a partir de i
    vet[i] = vet[min];
    vet[min] = aux;
  }
}
```

Selection-Sort

```
void selectionSort(int vet[], int tam){

   int i, min, aux;

   for(i=0; i<tam; i++){
      min = indiceMenor(vet, tam, i); //Acha posicao do menor elemento
      aux = vet[i];                   //a partir de i
      vet[i] = vet[min];
      vet[min] = aux;
   }
}
```

Selection-Sort

Com as funções anteriores implementadas podemos executar o exemplo:

```
int main(){
  int vetor[10]={14,7,8,34,56,4,0,9,-8,100};
  int i;
  printf("\nVetor Antes: (");
  for(i=0;i<10;i++)
    printf("%d, ",vetor[i]);
  printf(")");

  selectionSort(vetor,10);

  printf("\n\nVetor Depois: (");
  for(i=0;i<10;i++)
    printf("%d, ",vetor[i]);
  printf(")\n");

  return 0;
}
```

Selection-Sort

- Uma operação muito comum nos algoritmos de ordenação é a troca do conteúdo entre duas variáveis do vetor.
- No algoritmo Selection Sort temos:

```
aux = vet[i];
vet[i] = vet[min];
vet[min] = aux;
```

- Vamos criar uma função que faz a troca do conteúdo de duas variáveis inteiras:

```
void troca(int *a, int *b){
  int aux;
  aux = *a;
  *a = *b;
  *b = aux;
}
```

Selection-Sort

Podemos então alterar o código do Selection Sort de

```
void selectionSort(int vet[], int tam){
  int i, min, aux;
  for(i=0; i<tam; i++){
    min = indiceMenor(vet, tam, i); //Acha posicao do menor elemento
    aux = vet[i];                   //a partir de i
    vet[i] = vet[min];
    vet[min] = aux;
  }
}
```

para

```
void selectionSort2_1(int vet[], int tam){
  int i, min;
  for(i=0; i<tam; i++){
    min = indiceMenor(vet, tam, i); //Acha posicao do menor elemento
    troca(&vet[i], &vet[min]);      //a partir de i
  }
}
```

Selection-Sort

- O uso da função para achar o índice do menor elemento não é estritamente necessária.
- Podemos refazer a função selectionSort como segue:

```
for(i=0; i<tam; i++){
    min = i;
    for(j = i+1; j<tam; j++){
        if(vet[min] > vet[j])
            min = j;
    }
    troca(&vet[i], &vet[min]);
}
```

Selection-Sort

- O uso da função para achar o índice do menor elemento não é estritamente necessária.
- Podemos refazer a função selectionSort como segue:

```
for(i=0; i<tam; i++){
    min = i;
    for(j = i+1; j<tam; j++){
       if(vet[min] > vet[j])
           min = j;
    }
    troca(&vet[i], &vet[min]);
}
```

Selection-Sort

Antes:

```
void selectionSort(int vet[], int tam){
  int i, min;
  for(i=0; i<tam; i++){
    min = indiceMenor(vet, tam, i);
    troca(&vet[i], &vet[min]);
  }
}
```

Depois:

```
void selectionSort(int vet[], int tam){
  int i, j, min;
  for(i=0; i<tam; i++){
    min = i;
    for(j = i+1; j<tam; j++){//Acha posicao do menor elemento
      if(vet[min] > vet[j])          //a partir de i
        min = j;
    }
    troca(&vet[i], &vet[min]);
  }
}
```

Bubble-Sort

- Seja **vet** um vetor contendo números inteiros.
- Devemos deixar **vet** em ordem crescente.
- O algoritmo faz algumas iterações repetindo o seguinte:
  - Compare $vet[0]$ com $vet[1]$ e troque-os se $vet[0] > vet[1]$.
  - Compare $vet[1]$ com $vet[2]$ e troque-os se $vet[1] > vet[2]$.
  - .....
  - Compare $vet[tam-2]$ com $vet[tam-1]$ e troque-os se $vet[tam-2] > vet[tam-1]$.

  Após uma iteração repetindo estes passos o que podemos garantir???
  - O maior elemento estará na posição correta!!!

Bubble-Sort

- Seja **vet** um vetor contendo números inteiros.
- Devemos deixar **vet** em ordem crescente.
- O algoritmo faz algumas iterações repetindo o seguinte:
  - Compare $vet[0]$ com $vet[1]$ e troque-os se $vet[0] > vet[1]$.
  - Compare $vet[1]$ com $vet[2]$ e troque-os se $vet[1] > vet[2]$.
  - .....
  - Compare $vet[tam-2]$ com $vet[tam-1]$ e troque-os se $vet[tam-2] > vet[tam-1]$.

  Após uma iteração repetindo estes passos o que podemos garantir???
  - O maior elemento estará na posição correta!!!

Bubble-Sort

- Após uma iteração de trocas, o maior elemento estará na última posição.
- Após outra iteração de trocas, o segundo maior elemento estará na posição correta.
- E assim sucessivamente.
- Quantas iterações repetindo estas trocas precisamos para deixar o vetor ordenado?

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:

$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

Exemplo: (5,3,2,1,90,6).
Valores sublinhados estão sendo comparados:
$(\underline{5}, \underline{3}, 2, 1, 90, 6)$
$(3, \underline{5}, \underline{2}, 1, 90, 6)$
$(3, 2, \underline{5}, \underline{1}, 90, 6)$
$(3, 2, 1, \underline{5}, \underline{90}, 6)$
$(3, 2, 1, 5, \underline{90}, \underline{6})$
$(3, 2, 1, 5, 6, 90)$

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!!!
- Mas notem que não precisamos mais avaliar a última posição!

Bubble-Sort

- O código abaixo realiza as trocas de uma iteração.
- São comparados e trocados, os elementos das posições: 0 e 1; 1 e 2; ...; $i-1$ e i.
- Assumimos que de $(i+1)$ até $(tam-1)$, o vetor já tem os maiores elementos ordenados.

```
for(j=0; j < i; j++)
    if( vet[j] > vet[j+1] )
        troca(&vet[j], &vet[j+1])
```

Bubble-Sort

- O código abaixo realiza as trocas de uma iteração.
- São comparados e trocados, os elementos das posições: 0 e 1; 1 e 2; ...; $i-1$ e i.
- Assumimos que de $(i+1)$ até $(tam-1)$, o vetor já tem os maiores elementos ordenados.

```
for(j=0; j < i; j++)
    if( vet[j] > vet[j+1] )
        troca(&vet[j], &vet[j+1])
```

Bubble-Sort

```
void bubbleSort(int vet[], int tam){
    int i,j;

    for(i=tam-1; i>0; i--){
      for(j=0; j < i; j++)  //Faz trocas até posição i
        if( vet[j] > vet[j+1] )
          troca(&vet[j], &vet[j+1]);
    }
}
```

Bubble-Sort

- Note que as trocas na primeira iteração ocorrem até a última posição.
- Na segunda iteração ocorrem até a penúltima posição.
- E assim sucessivamente.
- Por que?

Exercício

Altere os algoritmos vistos nesta aula para que estes ordenem um vetor de inteiros em ordem decrescente ao invés de ordem crescente.

Exercício

No algoritmo SelectionSort, o laço principal é executado de i=0 até i=tam-2 e não i=tam-1. Por que?

```
void selectionSort2(int vet[], int tam){
 int i, j, min;
 for(i=0; i<tam-1; i++){condição não precisa ser i < tam. Por que?
    min = i;
    for(j = i+1; j<tam; j++){
      if(vet[min] > vet[j])
        min = j;
    }
    troca(&vet[i], &vet[min]);
 }
}
```

Insertion Sort

- Seja **vet** um vetor contendo números inteiros, que devemos deixar ordenado.
- A idéia do algoritmo é a seguinte:
  - A cada passo, uma porção de 0 até $i - 1$ do vetor já está ordenada.
  - Devemos inserir o item da posição i na posição correta para deixar o vetor ordenado até a posição i.
  - No passo seguinte consideramos que o vetor está ordenado até i.

Insertion Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice *i*

$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion Sort

Exemplo: (5,3,2,1,90,6).
O valor sublinhado representa onde está o índice i
$(5, \underline{3}, 2, 1, 90, 6)$: vetor ordenado de $0 - 0$.
$(3, 5, \underline{2}, 1, 90, 6)$: vetor ordenado de $0 - 1$.
$(2, 3, 5, \underline{1}, 90, 6)$: vetor ordenado de $0 - 2$.
$(1, 2, 3, 5, \underline{90}, 6)$: vetor ordenado de $0 - 3$.
$(1, 2, 3, 5, 90, \underline{6})$: vetor ordenado de $0 - 4$.
$(1, 2, 3, 5, 6, 90)$: vetor ordenado de $0 - 5$.

Insertion Sort

- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
j=i;
while(j>0){ //trocar v[i] com elementos anteriores
            //até achar sua posicao correta
  if(vet[j-1] > vet[j]){
    troca(&vet[j-1], &vet[j]);
    j--;
  } else
    break;
}
```

Insertion Sort

- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
j=i;
while(j>0){ //trocar v[i] com elementos anteriores
            //até achar sua posicao correta
  if(vet[j-1] > vet[j]){
     troca(&vet[j-1], &vet[j]);
     j--;
  } else
     break;
}
```

Insertion Sort

Código completo:

```
void insertionSort(int vet[], int tam){
  int i, j;

  for(i=1; i<tam; i++){
    j = i; //Colocar elemento v[i] na pos. correta

    while(j>0){ //trocar v[i] com elementos anteriores
                //até achar sua posicao correta
      if(vet[j-1] > vet[j]){
        troca(&vet[j-1], &vet[j]);
        j--;
      } else
        break;
    }
  }
}
```

Insertion Sort

Código completo:

```
void insertionSort(int vet[], int tam){
  int i, j;

  for(i=1; i<tam; i++){
    j = i; //Colocar elemento v[i] na pos. correta

    while(j>0){ //trocar v[i] com elementos anteriores
                //até achar sua posicao correta
      if(vet[j-1] > vet[j]){
        troca(&vet[j-1], &vet[j]);
        j--;
      } else
        break;
    }
  }
}
```

Insertion Sort

Código completo:

```
void insertionSort(int vet[], int tam){
  int i, j;

  for(i=1; i<tam; i++){
    j = i; //Colocar elemento v[i] na pos. correta

    while(j>0){ //trocar v[i] com elementos anteriores
                //até achar sua posicao correta
      if(vet[j-1] > vet[j]){
         troca(&vet[j-1], &vet[j]);
         j--;
      } else
        break;
    }
  }
}
```

Insertion Sort

- Vamos apresentar uma forma alternativa de colocar **v[i]** na posição correta.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];   //inserir aux na posição correta
j = i - 1; //analisar elementos das posições j anteriores


while( j >=0  && vet[j] > aux ){
     vet[j+1] = vet[j];      // enquanto vet[j] > aux empurra
      j--;                   // vet[j] para frente
}


//Quando terminar o laço:
//    OU j == -1, significando que você empurrou v[0] para frente
//    OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
 vet[j+1] = aux;
}
```

Insertion Sort

- Vamos apresentar uma forma alternativa de colocar **v[i]** na posição correta.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];   //inserir aux na posição correta
j = i - 1; //analisar elementos das posições j anteriores

while( j >=0  && vet[j] > aux ){
    vet[j+1] = vet[j];      // enquanto vet[j] > aux empurra
     j--;                   // vet[j] para frente
}


//Quando terminar o laço:
//    OU j == -1, significando que você empurrou v[0] para frente
//    OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
 vet[j+1] = aux;
}
```

Insertion Sort

- Vamos apresentar uma forma alternativa de colocar **v[i]** na posição correta.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];   //inserir aux na posição correta
j = i - 1; //analisar elementos das posições j anteriores

while( j >=0  && vet[j] > aux ){
     vet[j+1] = vet[j];      // enquanto vet[j] > aux empurra
      j--;                   // vet[j] para frente
}

//Quando terminar o laço:
//    OU j == -1, significando que você empurrou v[0] para frente
//    OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
 vet[j+1] = aux;
}
```

Insertion Sort

- Vamos apresentar uma forma alternativa de colocar **v[i]** na posição correta.
- Vamos supor que o vetor está ordenado de 0 até $i - 1$.
- Vamos inserir o elemento da posição i no lugar correto.

```
aux = vet[i];  //inserir aux na posição correta
j = i - 1; //analisar elementos das posições j anteriores

while( j >=0  && vet[j] > aux ){
     vet[j+1] = vet[j];     // enquanto vet[j] > aux empurra
      j--;                  // vet[j] para frente
}

//Quando terminar o laço:
//    OU j == -1, significando que você empurrou v[0] para frente
//    OU vet[j] <= aux.
// De qualquer forma (j+1) é a posição correta para v[i]
 vet[j+1] = aux;
}
```

Insertion Sort

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4) : aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4) : aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4) : aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4) : aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Insertion Sort

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4) : aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4) : aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4) : aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4) : aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Insertion Sort

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.
$(1, 3, 5, 10, \underline{20}, 2, 4) : aux = 2; j = 4;$
$(1, 3, 5, \underline{10}, 20, 20, 4) : aux = 2; j = 3;$
$(1, 3, \underline{5}, 10, 10, 20, 4) : aux = 2; j = 2;$
$(1, \underline{3}, 5, 5, 10, 20, 4) : aux = 2; j = 1;$
$(\underline{1}, 3, 3, 5, 10, 20, 4) : aux = 2; j = 0;$
Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$
$(1, 2, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Insertion Sort

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.
$(1, 3, 5, 10, \underline{20}, 2, 4) : aux = 2; j = 4;$
$(1, 3, 5, \underline{10}, 20, 20, 4) : aux = 2; j = 3;$
$(1, 3, \underline{5}, 10, 10, 20, 4) : aux = 2; j = 2;$
$(1, \underline{3}, 5, 5, 10, 20, 4) : aux = 2; j = 1;$
$(\underline{1}, 3, 3, 5, 10, 20, 4) : aux = 2; j = 0;$
Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$
$(1, 2, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Insertion Sort

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.

$(1, 3, 5, 10, \underline{20}, 2, 4)$: $aux = 2; j = 4;$

$(1, 3, 5, \underline{10}, 20, 20, 4)$: $aux = 2; j = 3;$

$(1, 3, \underline{5}, 10, 10, 20, 4)$: $aux = 2; j = 2;$

$(1, \underline{3}, 5, 5, 10, 20, 4)$: $aux = 2; j = 1;$

$(\underline{1}, 3, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$

$(1, 2, 3, 5, 10, 20, 4)$: $aux = 2; j = 0;$

Insertion Sort

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.
$(1, 3, 5, 10, \underline{20}, 2, 4) : aux = 2; j = 4;$
$(1, 3, 5, \underline{10}, 20, 20, 4) : aux = 2; j = 3;$
$(1, 3, \underline{5}, 10, 10, 20, 4) : aux = 2; j = 2;$
$(1, \underline{3}, 5, 5, 10, 20, 4) : aux = 2; j = 1;$
$(\underline{1}, 3, 3, 5, 10, 20, 4) : aux = 2; j = 0;$
Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$
$(1, 2, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Insertion Sort

Exemplo $(1, 3, 5, 10, 20, 2^*, 4)$ com $i = 5$.
$(1, 3, 5, 10, \underline{20}, 2, 4) : aux = 2; j = 4;$
$(1, 3, 5, \underline{10}, 20, 20, 4) : aux = 2; j = 3;$
$(1, 3, \underline{5}, 10, 10, 20, 4) : aux = 2; j = 2;$
$(1, \underline{3}, 5, 5, 10, 20, 4) : aux = 2; j = 1;$
$(\underline{1}, 3, 3, 5, 10, 20, 4) : aux = 2; j = 0;$
Aqui temos que $vet[j] < aux$ logo fazemos $vet[j + 1] = aux$
$(1, 2, 3, 5, 10, 20, 4) : aux = 2; j = 0;$

Insertion Sort

Código completo.

```
void insertionSort(int vet[], int tam){
    int i,j, aux;

    for(i=1; i<tam; i++){ //Assume vetor ordenado de 0 ate i-1

        aux = vet[i];
        j=i-1;

        while(j>=0 && vet[j] > aux){ //Poe elementos v[j]>v[i]
            vet[j+1] = vet[j];       //para frente
            j--;
        }
        vet[j+1] = aux; //poe v[i] na pos. correta
    }
}
```

Insertion Sort

Código completo.

```
void insertionSort(int vet[], int tam){
    int i,j, aux;

    for(i=1; i<tam; i++){ //Assume vetor ordenado de 0 ate i-1

        aux = vet[i];
        j=i-1;

        while(j>=0 && vet[j] > aux){ //Poe elementos v[j]>v[i]
            vet[j+1] = vet[j];       //para frente
            j--;
        }
        vet[j+1] = aux; //poe v[i] na pos. correta
    }
}
```

Insertion Sort

Código completo.

```
void insertionSort(int vet[], int tam){
    int i,j, aux;

    for(i=1; i<tam; i++){ //Assume vetor ordenado de 0 ate i-1

      aux = vet[i];
      j=i-1;

      while(j>=0 && vet[j] > aux){ //Poe elementos v[j]>v[i]
        vet[j+1] = vet[j];         //para frente
        j--;
      }
      vet[j+1] = aux; //poe v[i] na pos. correta
    }
}
```

Insertion Sort

Código completo.

```
void insertionSort(int vet[], int tam){
    int i,j, aux;

    for(i=1; i<tam; i++){ //Assume vetor ordenado de 0 ate i-1

      aux = vet[i];
      j=i-1;

      while(j>=0 && vet[j] > aux){ //Poe elementos v[j]>v[i]
        vet[j+1] = vet[j];         //para frente
        j--;
      }
      vet[j+1] = aux; //poe v[i] na pos. correta
    }
}
```

O Problema da Busca

- Vamos estudar alguns algoritmos para o seguinte problema:

Temos uma coleção de elementos, onde cada elemento possui um identificador/chave único, e recebemos uma chave para busca. Devemos encontrar o elemento da coleção que possui a mesma chave ou identificar que não existe nenhum elemento com a chave dada.

- Nos nossos exemplos usaremos um vetor de inteiros como a coleção.
  - O valor da chave será o próprio valor de cada número.
- Apesar de usarmos inteiros, os algoritmos servem para buscar elementos em qualquer coleção de elementos que possuam chaves que possam ser comparadas, como registros com algum campo de identificação único (RA, ou RG, ou CPF, etc.).

O Problema da Busca

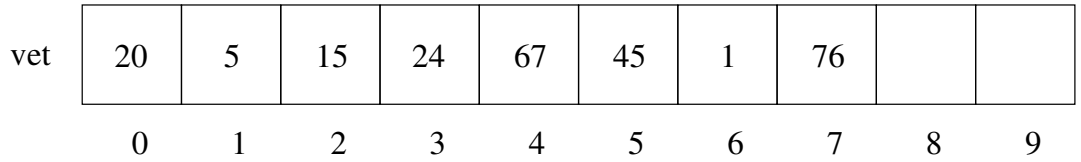
- O problema da busca é um dos mais básicos em Computação e também possui diversas aplicações.
  - Suponha que temos um cadastro com registros de motoristas.
  - Um vetor de registros é usado para armazenar as informações dos motoristas. Podemos usar como chave o número da carteira de motorista, ou o RG, ou o CPF.
- Veremos algoritmos simples para realizar a busca assumindo que dados estão em um vetor.
- Em cursos mais avançados são estudados outros algoritmos e estruturas (que não um vetor) para armazenar e buscar elementos.

O Problema da Busca

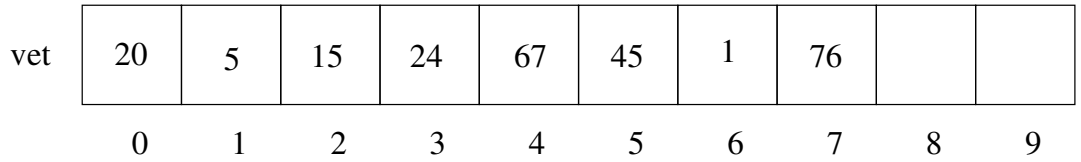
- Nos nossos exemplos vamos criar a função:
  - **int busca(int vet[], int tam, int chave)**, que recebe um vetor com um determinado tamanho, e uma chave para busca.
  - A função deve retornar o índice do vetor que contém a chave ou -1 caso a chave não esteja no vetor.

O Problema da Busca

chave = 45 tam = 8



chave = 100 tam = 8



No exemplo mais acima, a função deve retornar 5, enquanto no exemplo mais abaixo a função deve retornar -1.

Busca Sequencial

- A busca sequencial é o algoritmo mais simples de busca:
  - Percorra todo o vetor comparando a chave com o valor de cada posição.
  - Se for igual para alguma posição, então devolva esta posição.
  - Se o vetor todo foi percorrido então devolva -1.

Busca Sequencial

```
int buscaSequencial(int vet[], int tam, int chave){
  int i;
  for(i=0; i<tam; i++){
    if(vet[i] == chave)
      return i;
  }
  return -1;
}
```

Busca Sequencial

```c
#include <stdio.h>

int buscaSequencial(int vet[], int tam, int chave);

int main(){
  int pos, vet[] = {20, 5, 15, 24, 67, 45, 1, 76, -1, -1}; //-1 indica
                                                           //posição não usada
  pos = buscaSequencial(vet, 8, 45);
  if(pos != -1)
    printf("A posicao da chave 45 no vetor é: %d\n", pos);
  else
    printf("A chave 45 não está no vetor! \n");

  pos = buscaSequencial(vet, 8, 100);
  if(pos != -1)
    printf("A posicao da chave 100 no vetor é: %d\n", pos);
  else
    printf("A chave 100 não está no vetor! \n");
}

int buscaSequencial(int vet[], int tam, int chave){
  int i;
  for(i=0; i<tam; i++){
    if(vet[i] == chave)
      return i;
  }
  return -1;
}
```

Busca Binária

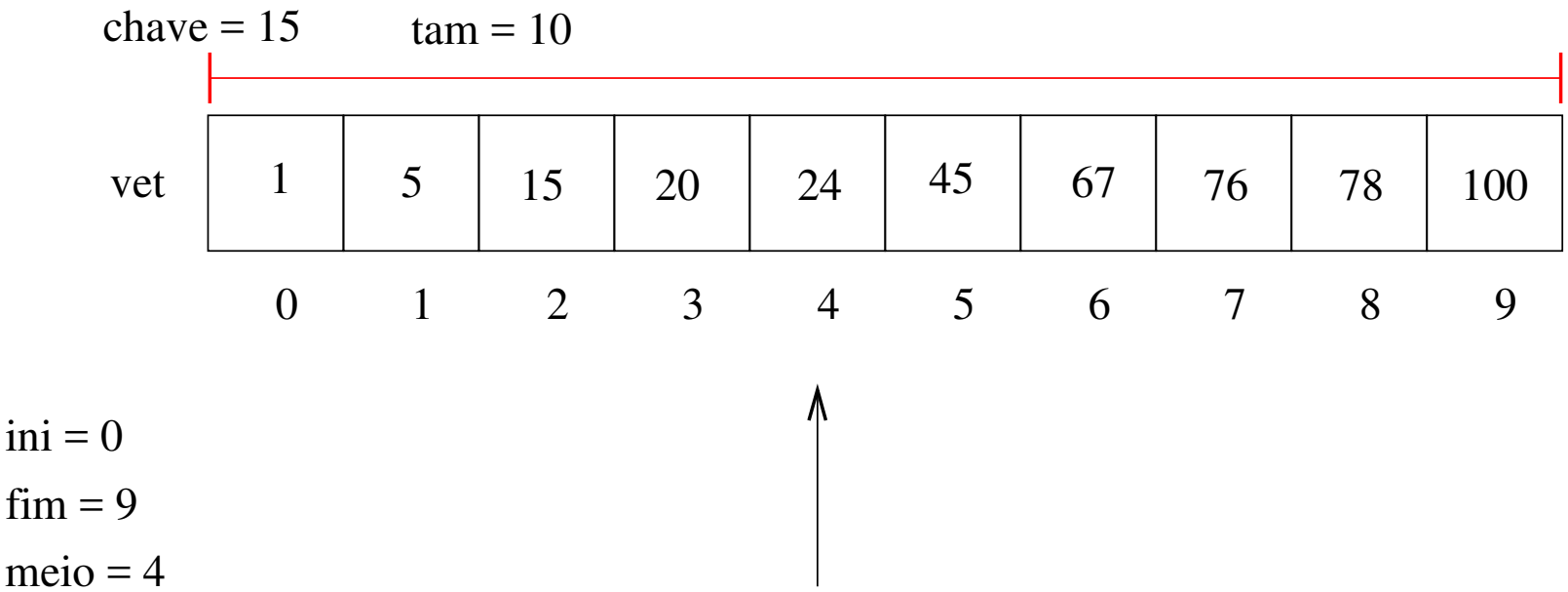
- A busca binária é um algoritmo um pouco mais sofisticado.
- É mais eficiente, mas requer que o vetor esteja ordenado pelos valores da chave de busca.
- A idéia do algoritmo é a seguinte (assuma que o vetor está ordenado):
  - Verifique se a chave de busca é igual ao valor da posição do meio do vetor.
  - Caso seja igual, devolva esta posição.
  - Caso o valor desta posição seja maior, então repita o processo mas considere que o vetor tem metade do tamanho, indo até a posição anterior a do meio.
  - Caso o valor desta posição seja menor, então repita o processo mas considere que o vetor tem metade do tamanho e inicia na posição seguinte a do meio.

Busca Binária

Pseudo-Código:

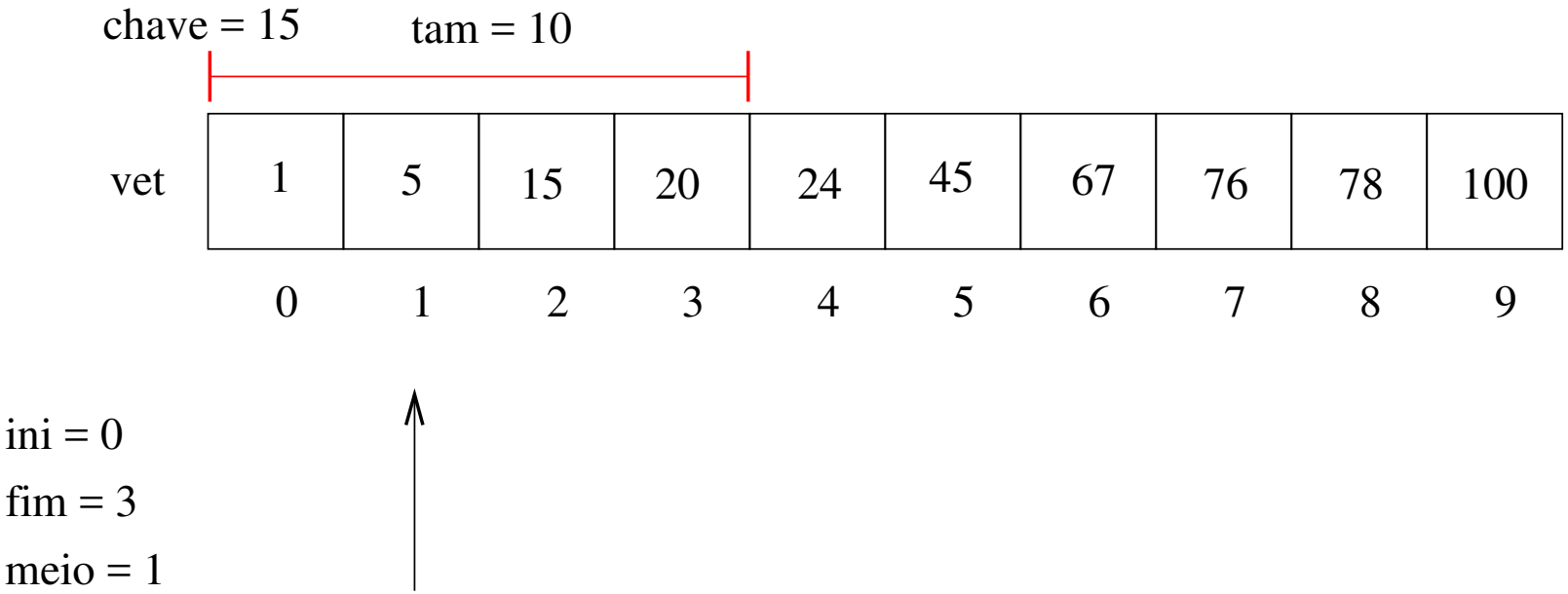
```
//vetor começa em ini e termina em fim
ini = 0
fim = tam-1

Repita enquanto tamanho do vetor considerado for >= 1
        meio = (ini + fim)/2

        Se vet[meio] == chave Então
                devolva meio

        Se vet[meio] > chave Então
                fim = meio - 1

        Se vet[meio] < chave Então
                ini = meio + 1
```

Busca Binária



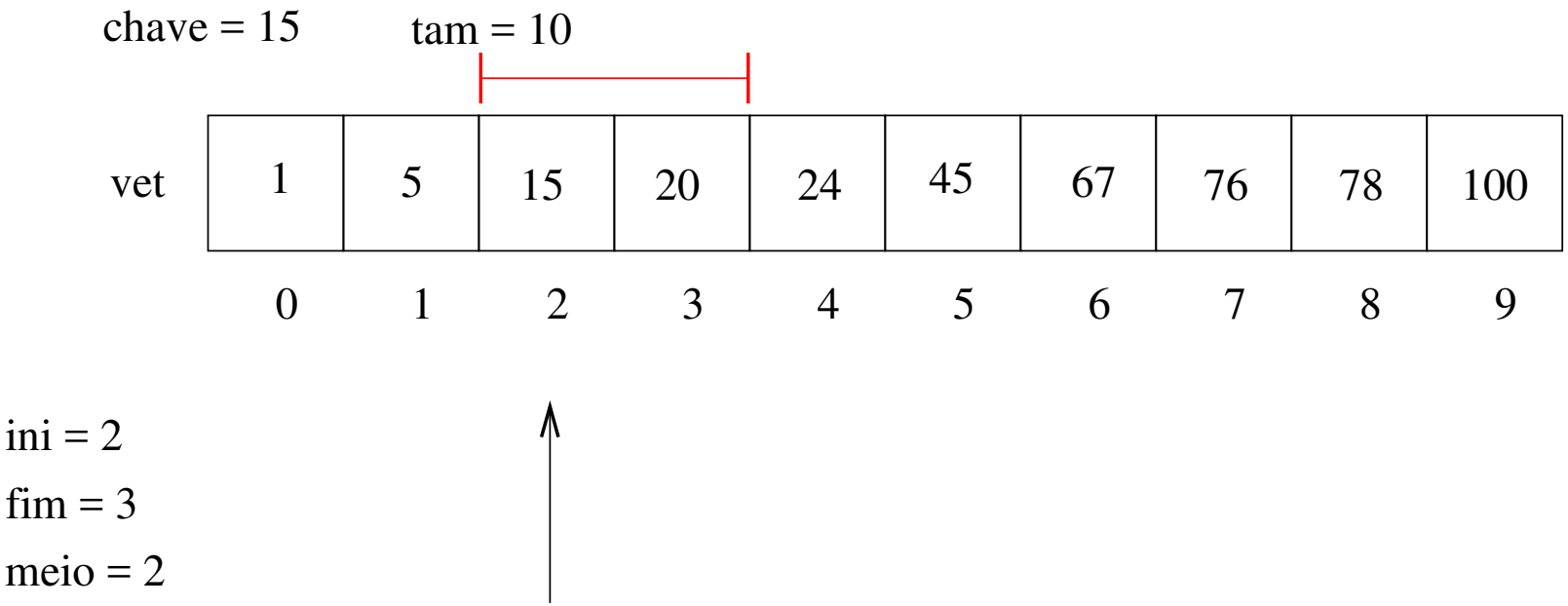
Como o valor da posição do meio é maior que a chave, atualizamos **fim** do vetor considerado.

Busca Binária



Como o valor da posição do meio é menor que a chave, atualizamos **ini** do vetor considerado.

Busca Binária



Finalmente encontramos a chave e podemos devolver sua posição 2.

Busca Binária

Código completo:

```
int buscaBinaria(int vet[], int tam, int chave){
  int ini=0, fim=tam-1, meio;

  while(ini <= fim){ //enquanto o vetor tiver pelo menos 1 elemento
    meio = (ini+fim)/2;

    if(vet[meio] == chave)
      return meio;
    else if(vet[meio] > chave)
      fim = meio - 1;
    else
      ini = meio + 1;
  }

  return -1;
}
```

Busca Binária

Exemplo de uso:

```
int main(){
   int vet[] = {20, 5, 15, 24, 67, 45, 1, 76, 78, 100};
   int pos, i;

   //antes de usar a busca devemos ordenar o vetor
   insertionSort(vet,10);
   printf("Vetor Ordenado:");
   for(i =0; i<10; i++){
      printf("%d, ", vet[i]);
   }
   printf("\n");
   pos = buscaBinaria(vet, 10, 15);
   if(pos != -1)
      printf("A posicao da chave 15 no vetor é: %d\n", pos);
   else
      printf("A chave 15 não está no vetor! \n");
}
```

Eficiência dos Algoritmos

Podemos medir a eficiência de qualquer algoritmo analisando a quantidade de recursos (tempo, memória, banda de rede, etc.) que o algoritmo usa para resolver o problema para o qual foi proposto.

- A forma mais simples é medir a eficiência em relação ao tempo. Para isso, analisamos quantas instruções um algoritmo usa para resolver o problema.
- Podemos fazer uma análise simplificada dos algoritmos de busca analisando a quantidade de vezes que os algoritmos **acessam** uma posição do vetor.

Eficiência dos Algoritmos

No caso da busca sequencial existem três possibilidades:

- Na melhor das hipóteses a chave de busca estará na posição 0. Portanto teremos um único acesso em **vet[0]**.
- Na pior das hipóteses, a chave é o último elemento ou não pertence ao vetor, e portanto acessaremos todas as **tam** posições do vetor.
- É possível mostrar que se uma chave qualquer pode ser requisitada com a mesma probabilidade, então o número de acessos será

$$(\text{tam} + 1)/2$$

na média.

Eficiência dos Algoritmos

No caso da busca binária temos as três possibilidades:

- Na melhor das hipóteses a chave de busca estará na posição do meio. Portanto teremos um único acesso.
- Na pior das hipóteses, teremos $(\log_2 \textbf{tam})$ acessos.
  - Para ver isso note que a cada verificação de uma posição do vetor, o tamanho do vetor considerado é dividido pela metade. No pior caso repetimos a busca até o vetor considerado ter tamanho 1. Se você pensar um pouco, o número de acessos x pode ser encontrado resolvendo-se a equação:

    $$\frac{\text{tam}}{2^x} = 1$$

    cuja solução é $x = (\log_2 \textbf{tam})$.
- É possível mostrar que se uma chave qualquer pode ser requisitada com a mesma probabilidade, então o número de acessos será

  $$(\log_2 \textbf{tam}) - 1$$

  na média.

Eficiência dos Algoritmos

Para se ter uma idéia da diferença de eficiência dos dois algoritmos, considere que temos um cadastro com 10^6 (um milhão) de itens.

- Com a busca sequencial, a procura de um item qualquer gastará na média

$$(10^6 + 1)/2 \approx 500000 \text{ acessos.}$$

- Com a busca binária teremos

$$(\log_2 10^6) - 1 \approx 20 \text{ acessos.}$$

Eficiência dos Algoritmos

Mas uma ressalve deve ser feita: para utilizar a busca binária, o vetor precisa estar ordenado!

- Se você tiver um cadastro onde vários itens são removidos e inseridos com frequência, e a busca deve ser feita intercalada com estas operações, então a busca binária pode não ser a melhor opção, já que você precisará ficar mantendo o vetor ordenado.
- Caso o número de buscas feitas seja muito maior, quando comparado com outras operações, então a busca binária é uma boa opção.

Exercícios

- Altere o código do algoritmo insertionSort para que este ordene um vetor de inteiros em ordem decrescente.

Exercícios

- Refaça as funções de busca sequencial e busca binária assumindo que o vetor possui chaves que podem aparecer repetidas. Neste caso, você deve retornar em um outro vetor todas as posições onde a chave foi encontrada.
  Protótipo: **int busca(int vet[], int tam, int chave, int posicoes[])**
- Você deve devolver em **posicoes[]** as posições de **vet** que possuem a **chave**, e o retorno da função é o número de ocorrências da chave.
  - **OBS:** Na chamada desta função, o vetor **posições** deve ter espaço suficiente (tam) para guardar todas as possíveis ocorrências.