Algoritmos Recursivos Parte 2

Algoritmos e Estrutura de Dados 2 Prof. Dr. Anderson Bessa da Costa Universidade Federal de Mato Grosso do Sul

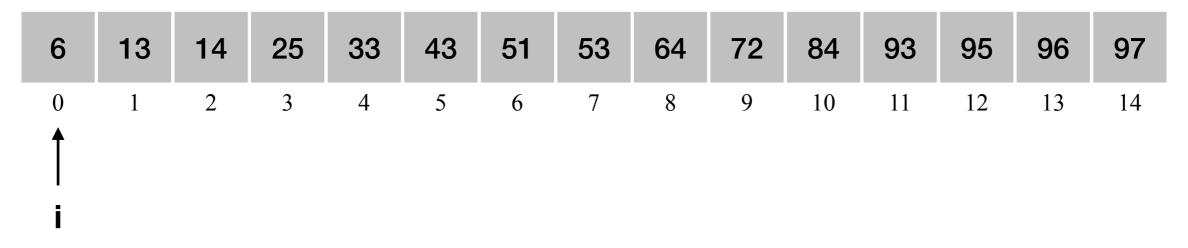
Problema: Busca

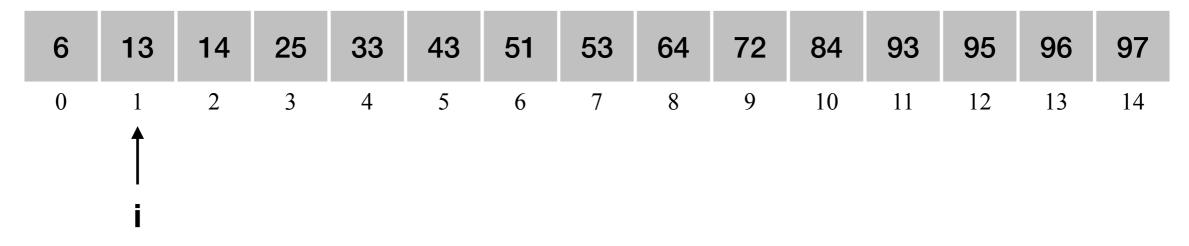
- Considere vetor de registros
- Cada registro tem chave associada
- Forneça algoritmo eficiente para procurar um registro que contenha uma chave específica

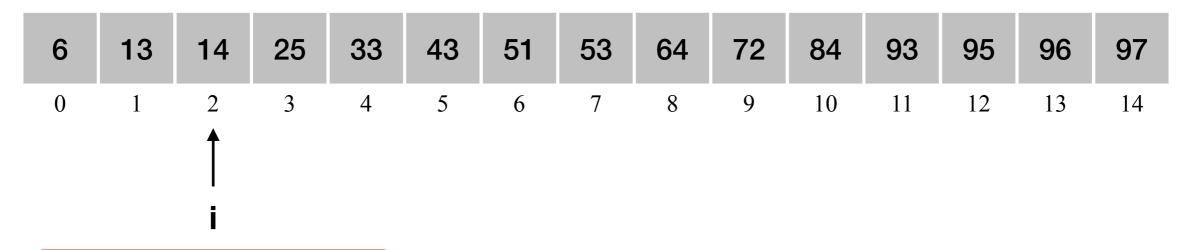
Busca Sequencial

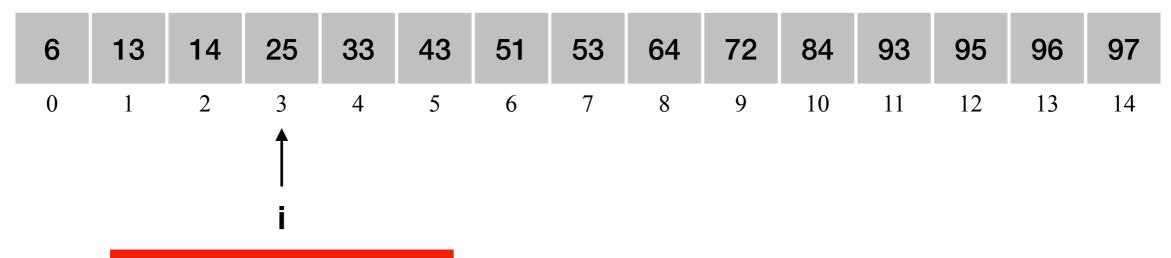
- Percorre elemento a elemento
- Compara cada registro com a chave correspondente
- Busca termina quando:
 - 1. Chave encontrada
 - 2. Ou quando busca examinou todos os registros sem sucesso

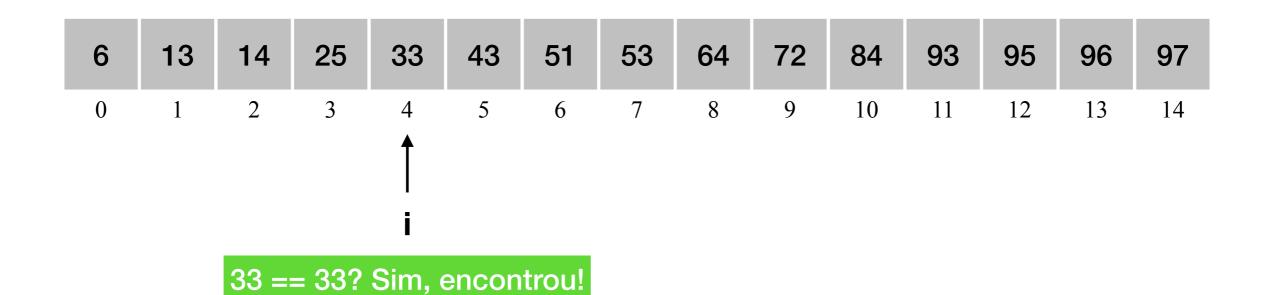
Busca Sequencial: Exemplo para 33











Busca Sequencial: Algoritmo

```
int busca_sequencial(int a[], int n, int x) {
   int i;

// percorra vetor, elemento por elemento
   for (i = 0; i < n; i++) {
        // se chave encontrada
        if (a[i] == x) {
            return i;
        }
   }

// se chegou aqui, chave nao encontrada
   return -1;
}</pre>
```

Busca Sequencial: Análise

- Tempo de execução no pior caso?
 - Determinar a notação ${\cal O}$ para o número de operações necessárias para a busca
 - Depende de n, o número de entradas
 - Vetor de n elementos, pior caso requer n acessos ao vetor: O(n)
 - Registro desejado aparece na última posição
 - Registro desejado n\u00e3o aparece no vetor

Busca Sequencial: Pior caso

- Vetor de n elementos, pior caso requer n acessos ao vetor: O(n)
 - Registro desejado aparece na última posição do vetor
 - Registro desejado não aparece no vetor

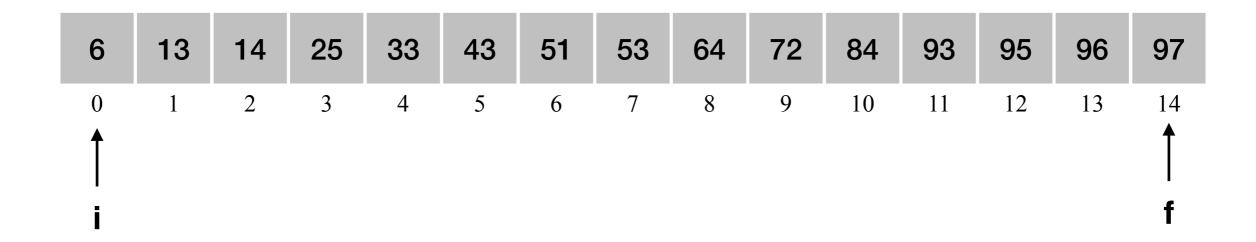
Busca Binária

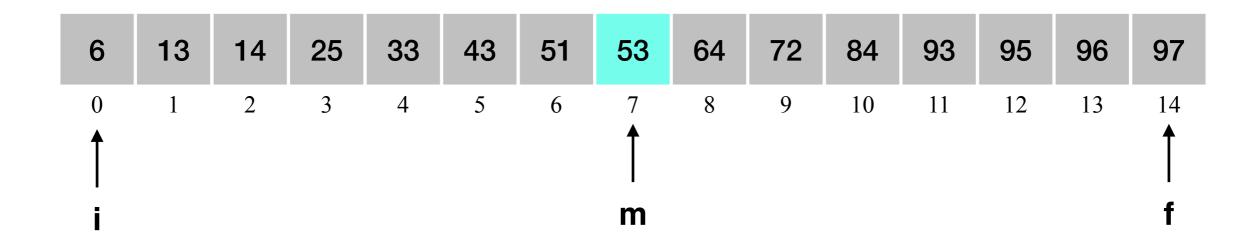
- Parte do pressuposto de que o vetor está ordenado
- Mais eficiente
- Segue o paradigma de divisão e conquista
 - Realiza sucessivas divisões do espaço de busca

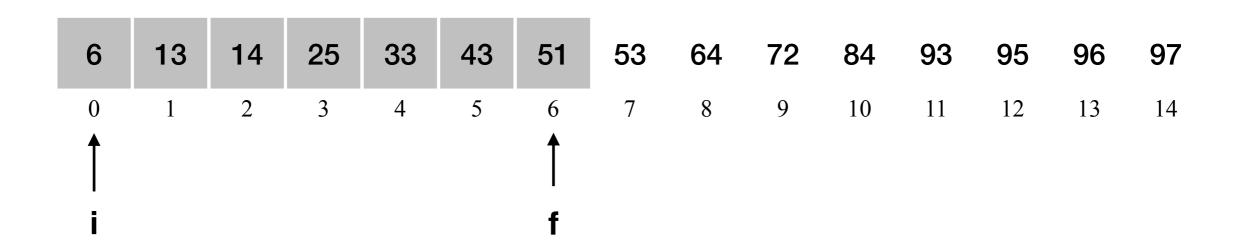
Busca Binária (cont.)

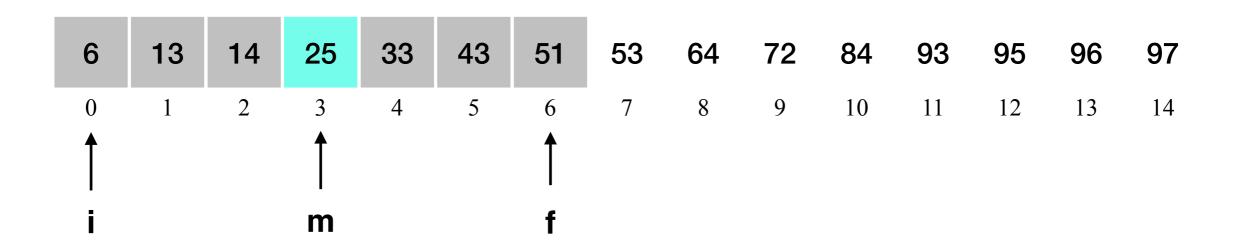
- Se vetor possuir um elemento, problema simples
- Caso contrário, compare item procurado ao item posicionado no meio do vetor
 - Se iguais, busca terminou com sucesso
 - Se elemento do meio for maior que o elemento procurado, o processo de busca será repetido na primeira metade do vetor
 - Caso contrário, o processo será repetido na segunda metade

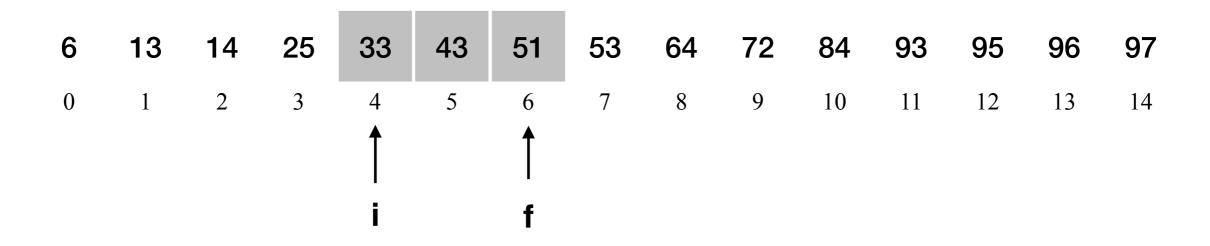
Busca Binária: Exemplo para 33

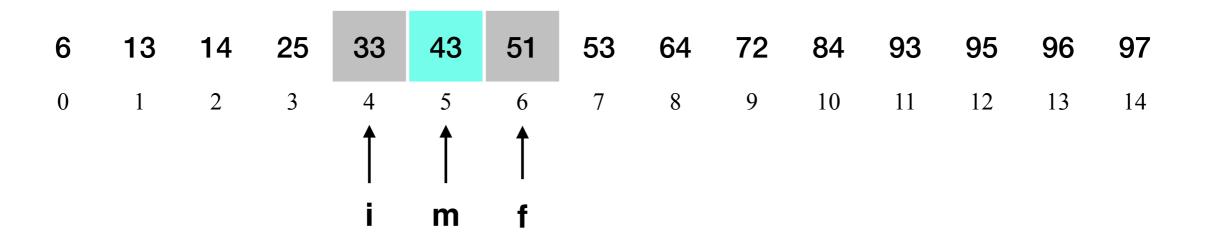


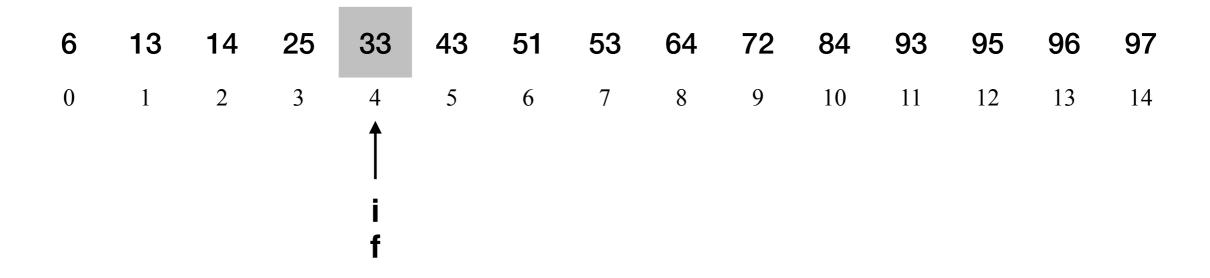


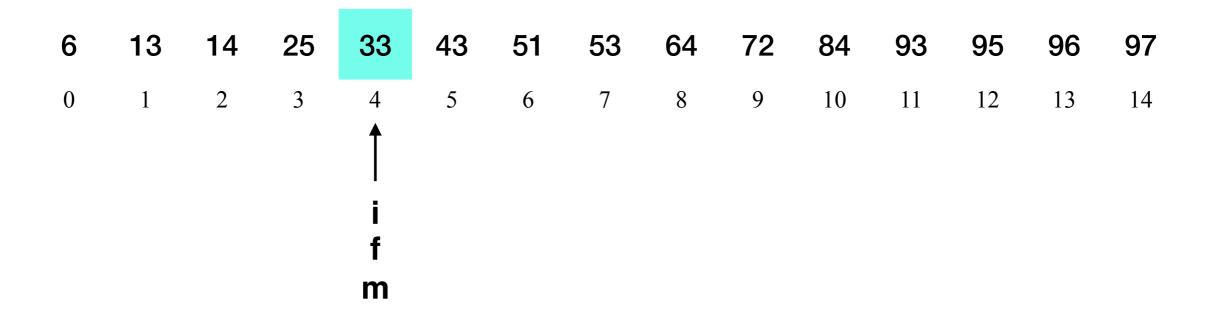


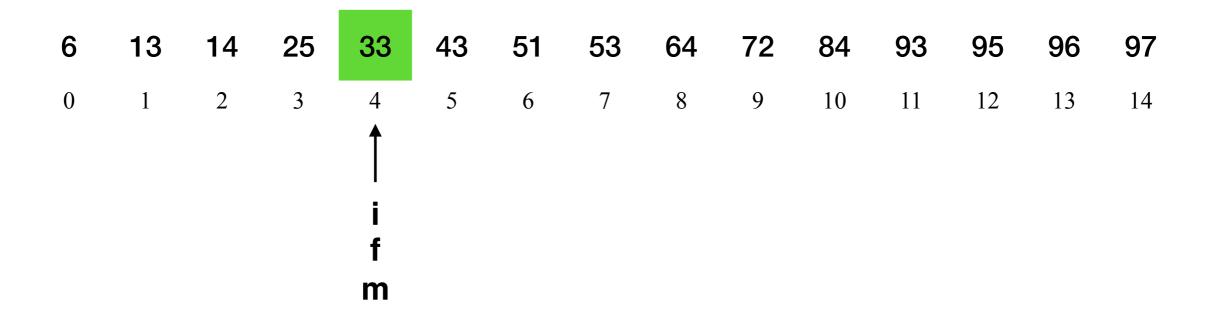












Busca Binária: Implementação

- Uma função para a busca binária:
 - Entrada: vetor a e um elemento x;
 - Retorna: índice i em a, tal que a[i] seja igual a x, ou -1, se esse i não existir;
- Implementação recursiva requer dois parâmetros adicionais
 - Limites entre os quais o vetor deve ser pesquisado precisam ser também especificados

Busca Binária: Algoritmo

```
int busca_binaria(int a[], int x, int inicio, int fim) {
    int meio;
    // calcule a posicao do meio
    meio = (inicio + fim) / 2;
    // se inicio > fim, entao busca acabou sem encontrar o x
    if (inicio > fim)
        return -1;
    // se x esta na posicao do meio
    else if (x == a[meio])
        return meio;
    // se x eh menor que elemento do meio
    else if (x < a[meio])</pre>
        return busca_binaria(a, x, inicio, meio - 1);
    // se x eh maior que elemento do meio
    else
        return busca_binaria(a, x, meio + 1, fim);
```

Busca Binária: Análise

- Complexidade pior caso?
 - Qual a profundidade máxima de chamadas recursivas na pesquisa binária em função de n?
 - A cada nível na recursão, dividimos o vetor ao meio
 - Portanto, a profundidade máxima de recursão é piso $(\log_2 n)$ e o pior caso = $O(\log n)$

Podemos Fazer Melhor?

- Caso pior e médio da pesquisa sequencial = O(n)
- Caso pior e médio da busca binária = $O(log_2n)$
- Podemos fazer melhor do que isso?
 - SIM. Use uma tabela de hash!

Torres de Hanói

O Problema das Torres de Hanoi

 Um problema que não é especificado em termos de recursividade ..

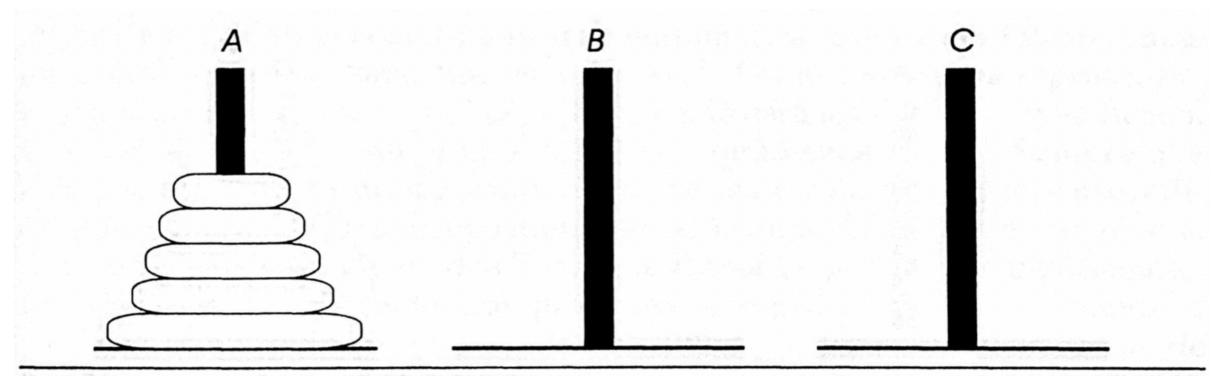


Figura 3.3.1 A configuração inicial das Torres de Hanoi.

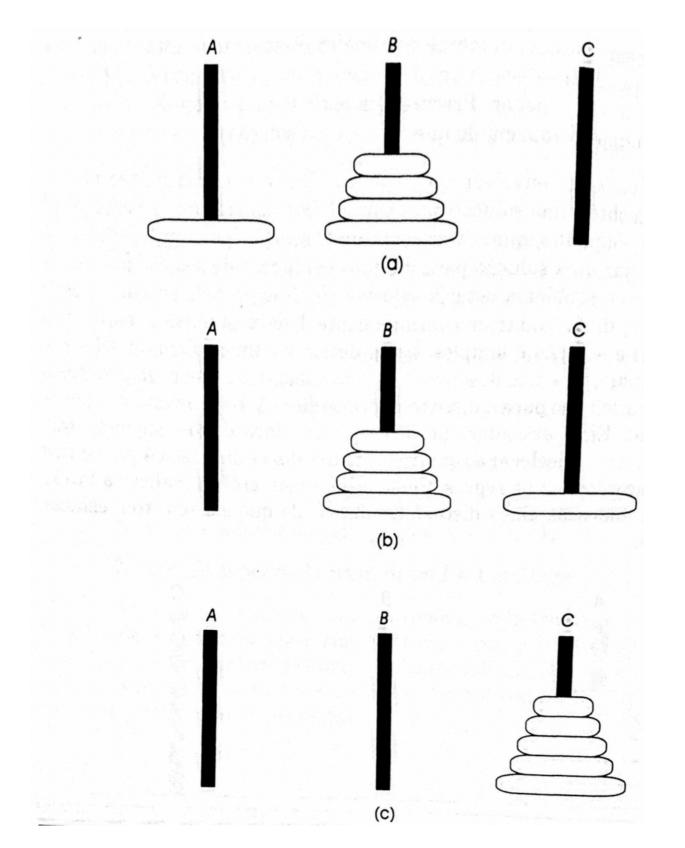
O Problema das Torres de Hanoi (cont.)

https://www.mathsisfun.com/games/towerofhanoi.html

Solução Torres de Hanoi

- No caso trivial de um disco a solução é simples: basta deslocar o único disco da estaca A para a C
- Se pudermos declarar uma solução para n discos em termos de n-1, teríamos desenvolvido uma solução recursiva ...

Solução Recursiva Torres de Hanoi



Solução Recursiva Torres de Hanoi (cont.)

Para mover n discos de A para C, usando B como auxiliar:

- 1. Se n = 1, desloque o único disco de A para C e pare.
- Desloque os n 1 primeiros discos de A para B, usando C como auxiliar.
- 3. Desloque o último disco de A para C.
- 4. Mova os n 1 discos de B para C, usando A como auxiliar.

Solução Recursiva Torres de Hanoi (cont.)

Elaboração das entradas e saídas é importante p/ solução:

- Usuário não verá o método elegante que você incorporou em seu programa, mas "dará duro" para decifrar a saída ou para adaptar os dados de entrada
- Pequena mudança no formato de entrada ou da saída pode tornar o programa muito mais fácil de elaborar

Algoritmo Torres de Hanoi

```
#include <stdio.h>
void towers (int n, char frompeg, char topeg, char auxpeg) {
    // se existe um so disco, faca o movimento e retorne
    if (n == 1) {
        printf("mover disco 1 da estaca %c p/ a estaca %c\n", frompeg, topeg);
    else {
        // mover os primeiros n - 1 discos de A p/ B
        towers(n - 1, frompeg, auxpeg, topeg);
        printf("mover disco %d da estaca %c p/ a estaca %c\n", n, frompeg,
topeq);
        // mover n - 1 discos de B p/ C usando A como auxiliar
        towers(n - 1, auxpeg, topeg, frompeg);
```

Torres de Hanói

- Existem três torres
- 64 discos de ouro, com tamanhos decrescentes, colocados na primeira torre
- Os discos devem ser movidos dentro de uma semana.
 Suponha que um disco possa ser movido em 1 segundo.
 Isso é possível?
- Para criar um algoritmo para resolver este problema, é conveniente generalizar o problema para o problema "Ndisco", onde no nosso caso N=64.

Análise

 Movimentos necessários para resolver, em função de n, o número de discos a serem movidos.

n	# de mov necessários
1	1
2	3
3	7
4	15
5	31
i	2 ⁱ -1
64	2 ⁶⁴ -1 (número grande)

Resumo

- Recursão permite que alguns problemas sejam resolvidos de maneira elegante e eficiente
- Funções podem exigir mais de uma chamada recursiva para realizar sua tarefa
- Existem problemas para os quais podemos projetar uma solução, mas a natureza do problema torna a solução efetivamente incomputável

Referências

- TENENBAUM, Aaron M; ANGENTEIN, Moshe; LANGSAM, Yedidyah. Estruturas de dados usando C. Sao Paulo, SP: Pearson, 1995. 884p.
- FEOFILOFF, Paulo. Algoritmos em linguagem C. Rio de Janeiro: Elsevier, 2009. 208 p.