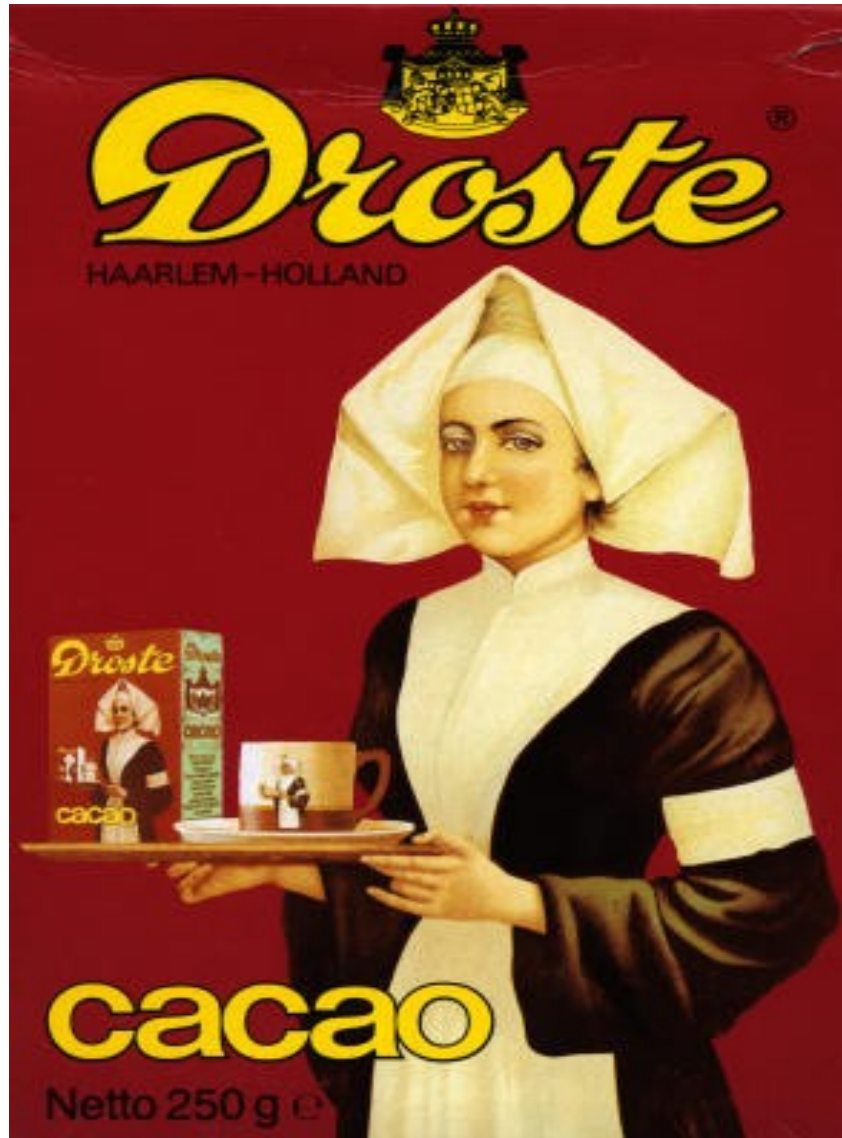
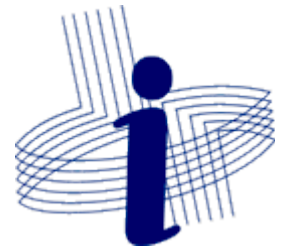




Universidade Federal de Viçosa
Departamento de Informática
Centro de Ciências Exatas e Tecnológicas



INF 112

Programação 2

Aula “6”

Recursividade - 2

Recursividade

Suponha que você tenha um array onde os elementos estão ordenados. Qual a melhor forma de encontrar a posição de um elemento (ou ver que ele não existe) nesse array?

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

Recursividade

Suponha uma busca sequencial: quantas comparações temos que fazer para encontrar a posição do elemento 8? e do elemento 58? e do elemento 984?

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99



Recursividade

- Suponha uma busca sequencial: se o array tem tamanho N , quantas comparações temos que fazer, em média, para encontrar um elemento nele?

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

Recursividade

- Como diminuir o número de comparações necessárias?

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

Recursividade

- Busca binária!
- Imagine que você tenha uma lista telefônica e esteja procurando o telefone de “Maria Silva”: como você aceleraria a busca?



Recursividade

- A ideia da busca binária é sempre dividir o espaço de busca pela metade.
- No pior caso, são realizadas em torno de $\log(N)$ comparações! \rightarrow é um algoritmo $O(\log(n))$
- Qual a complexidade de melhor caso?



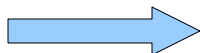
Recursividade

Vamos fazer uma busca binária iterativa!



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 7



```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 7



```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```


0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 7
Meio = $(0+7)/2=3$

```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99


meio



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 7
Meio = 3

```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

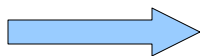
meio



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 7
Meio = 3

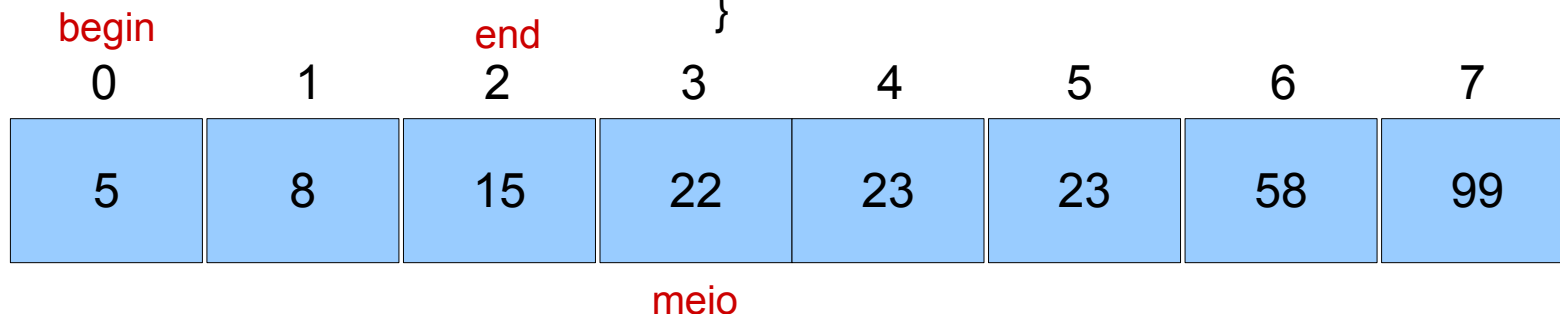
```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 3-1=2
Meio = 3

```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 3-1=2
Meio = 3



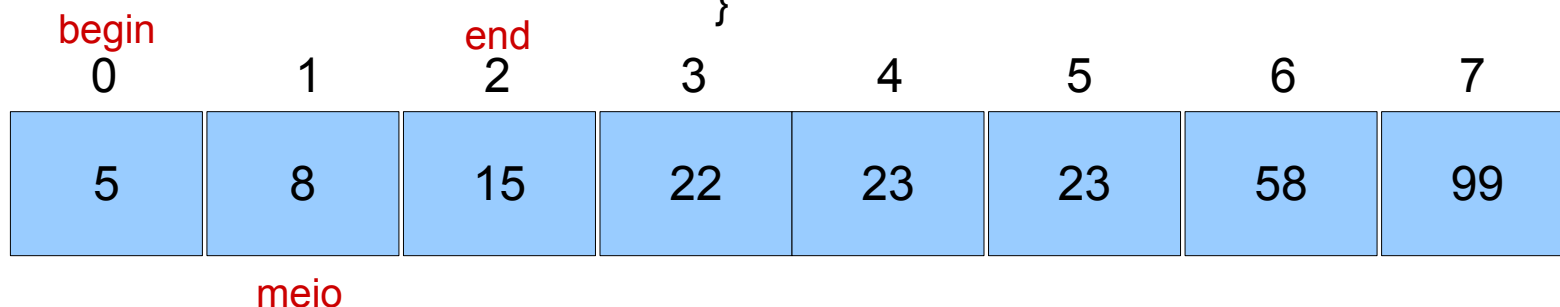
```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 3-1=2
Meio = (0+2)/2=1

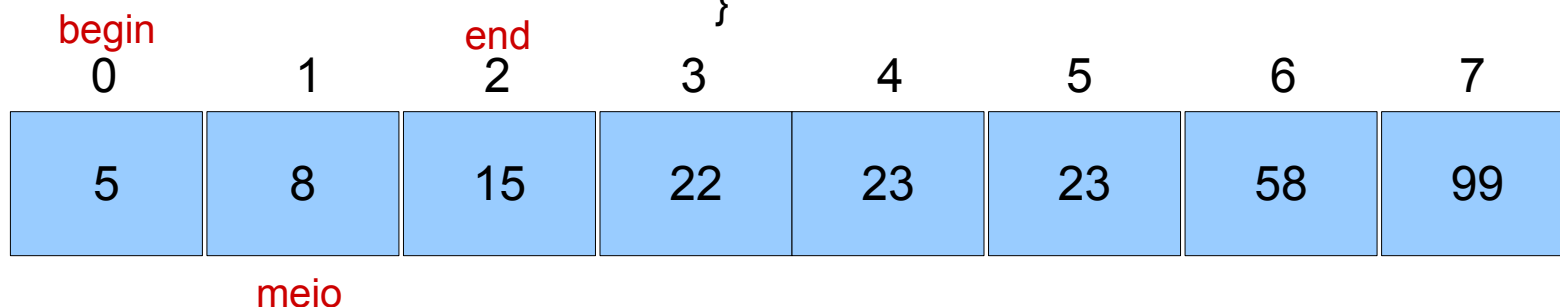
```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 3-1=2
Meio = (0+2)/2=1

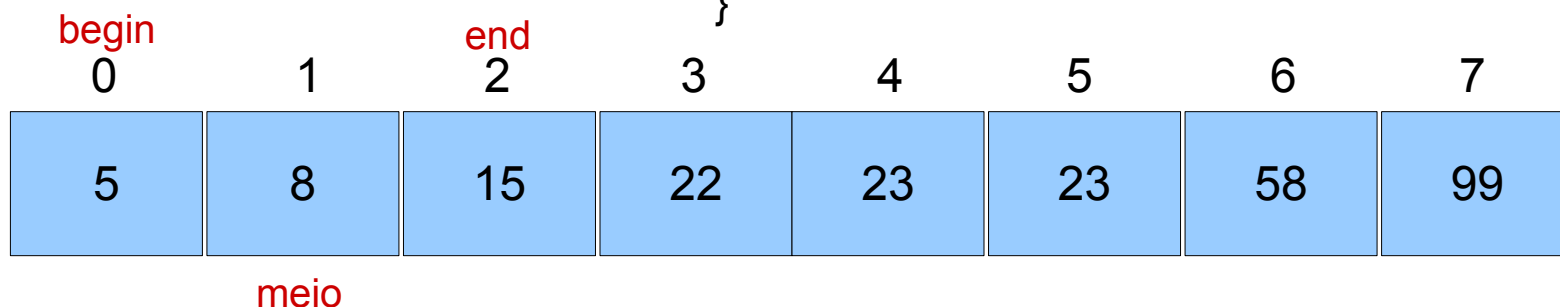
```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 3-1=2
Meio = (0+2)/2=1

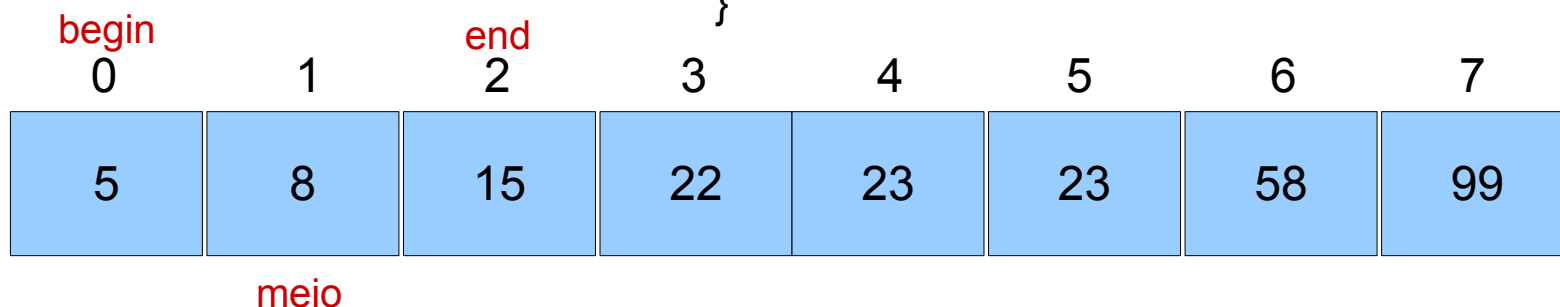
```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 0
End = 3-1=2
Meio = (0+2)/2=1

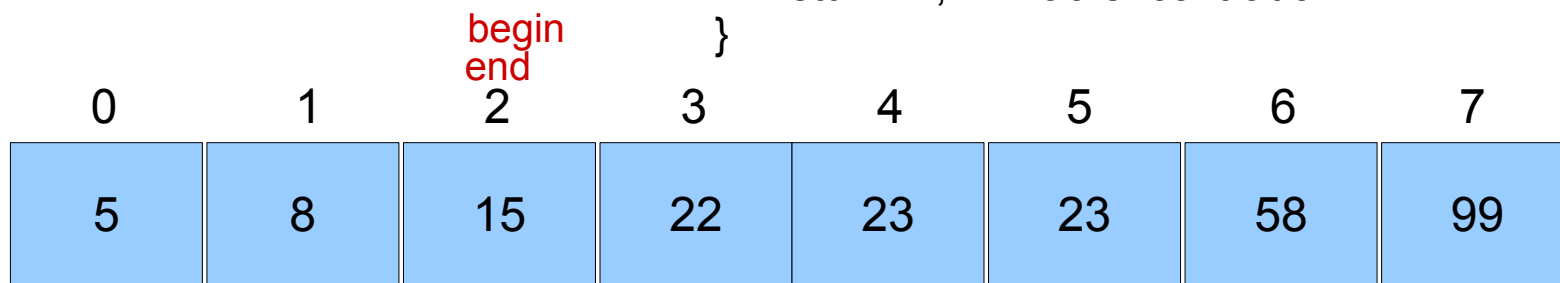
```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 1+1=2
End = 3-1=2
Meio = (0+2)/2=1

```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



meio



Recursividade

Tam = 8
Chave = 15
Begin = 2
End = 2
Meio = 1



```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

begin
end

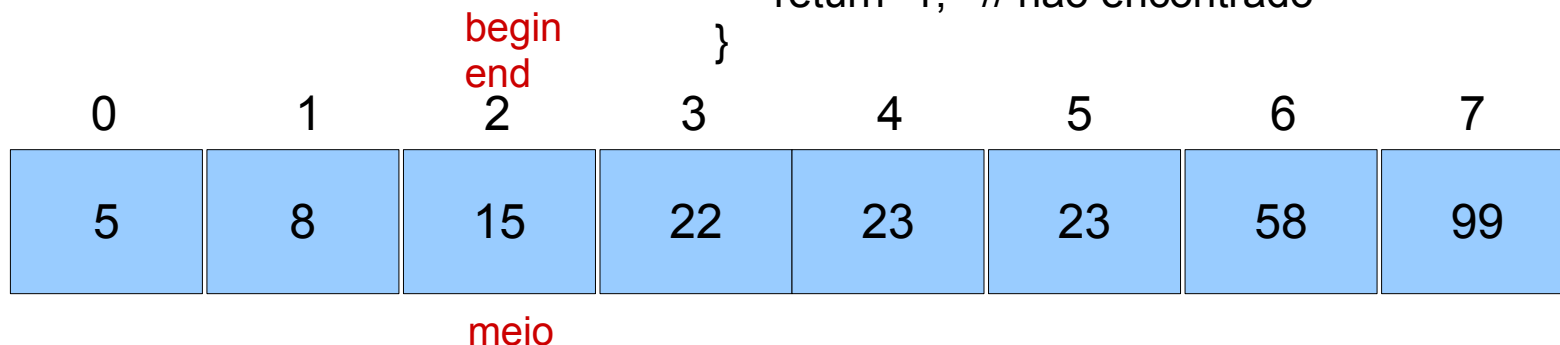
meio



Recursividade

Tam = 8
Chave = 15
Begin = 2
End = 2
Meio = $(2+2)/2=2$

```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```



Recursividade

Tam = 8
Chave = 15
Begin = 2
End = 2
Meio = $(2+2)/2=2$

Vai retornar 2!

Retorna a posição
onde está a chave.

```
int pesquisaBinaria (int vet[], int chave, int Tam){  
    int begin = 0;  
    int end = Tam-1;  
    int meio;  
    while (begin <= end){  
        meio = (begin + end)/2;  
        if (chave == vet[meio])  
            return meio;  
        if (chave < vet[meio])  
            end = meio-1;  
        else  
            begin = meio+1;  
    }  
    return -1; // não encontrado  
}
```

0	1	begin end 2	3	4	5	6	7
5	8	15	22	23	23	58	99
meio							



Recursividade

Voltando à nossa pergunta:

Suponha que você tenha um array onde os elementos estão ordenados. Qual a melhor forma de encontrar a posição de um elemento (ou ver que ele não existe) nesse array?

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

Recursividade

Exercício: agora que já vimos a busca binária iterativa, como seria uma busca binária recursiva?

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99

Recursividade

- Busca binária recursiva opção 1:

```
int buscaBin(int *array,int begin, int end, int chave) {  
    if (begin > end)  
        return -1;  
    int meio = (end-begin)/2 + begin;  
    if (array[meio] == chave)  
        return meio;  
    if (array[meio] > chave)  
        return buscaBin(array,begin, meio-1, chave);  
    return buscaBin(array,meio+1, end, chave);  
}
```



Recursividade

- Busca binária recursiva opção 2:

```
int pesquisaBinariaRec (int vet[], int chave, int begin, int end){
    int meio = (begin + end)/2;
    if (vet[meio] == chave)
        return meio;
    if (begin >= end)
        return -1; // não encontrado
    else{
        if (vet[meio] < chave)
            return pesquisaBinariaRec(vet, chave, meio+1,
end);
        else
            return pesquisaBinariaRec(vet, chave, begin,
meio-1);
    }
}
```



Recursividade

- Busca binária recursiva:

```
int buscaBin(int *array,int begin, int end, int chave) {  
    if (begin > end)          return -1;  
    int meio = (end-begin)/2 + begin;  
    if (array[meio] == chave)  
        return meio;  
    if (array[meio] > chave)  
        return buscaBin(array,begin, meio-1, chave);  
    return buscaBin(array,meio+1, end, chave);  
}
```

Teste: realizar 100.000 buscas em um array ordenado de tamanho 500.000 em um computador com processador Intel E7500.

Gasta 0.01 segundo!

```
int buscaSeq(int *array,int begin,int end, int chave) {  
    for(int i=begin;i<=end;i++)  
        if (array[i] == chave)  
            return i;  
    return -1;  
}
```

Gasta 17.5 segundos !



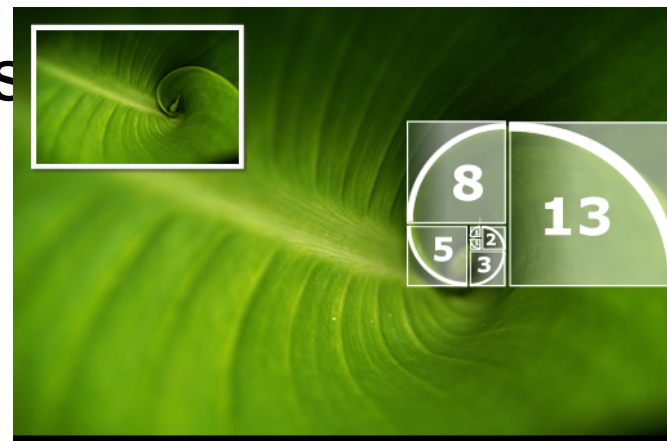
Recursividade

- Busca binária recursiva: tanto a busca sequencial quanto a busca binária podem ser implementadas de forma iterativa ou recursiva.
- Aqui usamos recursão para ilustrar nosso tópico de estudo.



Recursividade

- Fibonacci: sequência muito famosa. Presente em várias construções da natureza.
- $$\text{Fib}(n) = \begin{cases} 0, & \text{se } n = 0 \text{ ou } 1 \\ \text{Fib}(n-1) + \text{Fib}(n-2), & \text{caso contrário} \end{cases}$$
- Isso gera a seguinte sequência: 0 1 1 2 3 5 8 13 ...
- Como podemos calcular o i-ésimo Fibonacci?
- Ideias?



Recursividade

- Fibonacci: código iterativo
- Pergunta: por que utilizamos o tipo de retorno double?

```
double fib(int n) {  
    if (n<=1)  
        return n;  
    double a=0,b=1;  
    for(int i=2;i<=n;i++) {  
        int aux = b;  
        b = a+b;  
        a = aux;  
    }  
    return b;  
}
```

b guarda o último fibonacci calculado e **a** guarda o anterior a b.



Recursividade

- Fibonacci: código iterativo

```
double fib(int n) {  
    if (n<=1)  
        return n;  
    double a=0,b=1;  
    for(int i=2;i<=n;i++) {  
        int aux = b;  
        b = a+b;  
        a = aux;  
    }  
    return b;  
}
```

b guarda o último fibonacci calculado e **a** guarda o anterior a b.



Recursividade

- Fibonacci: iterativo e recursivo

```
double fib(int n) {  
    if (n<=1)  
        return n;  
    double a=0,b=1;  
    for(int i=2;i<=n;i++) {  
        int aux = b;  
        b = a+b;  
        a = aux;  
    }  
    return b;  
}
```

```
double fib(int n) {  
    if (n<=1)  
        return n;  
    return fib(n-1) + fib(n-2);  
}
```

Código elegante!
Parece bom!



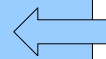
Recursividade

- Fibonacci: iterativo e recursivo

```
double fib(int n) {  
    if (n<=1)  
        return n;  
    double a=0,b=1;  
    for(int i=2;i<=n;i++) {  
        int aux = b;  
        b = a+b;  
        a = aux;  
    }  
    return b;  
}
```

```
double fib(int n) {  
    if (n<=1)  
        return n;  
    return fib(n-1) + fib(n-2);  
}
```

n	Tempo de execução(s)	
	Iterativo	Recursivo
8	0.003	0.002
16	0.003	0.003
47	0.003	15.639
48	0.003	25.787
49	0.003	41.410

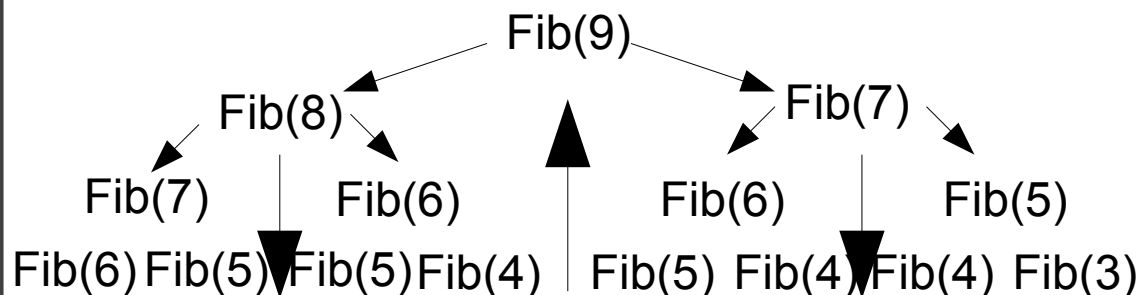


??

O que está
acontecendo?

Recursividade

- Essa versão recursiva do cálculo de um número de Fibonacci é muito ineficiente pois alguns cálculos são repetidos várias vezes.
- Por exemplo, para se calcular $\text{Fib}(9)$ precisa se calcular $\text{Fib}(8)$ e $\text{Fib}(7)$. Note que $\text{Fib}(7)$ é recalculado ao se calcular $\text{Fib}(8)$.

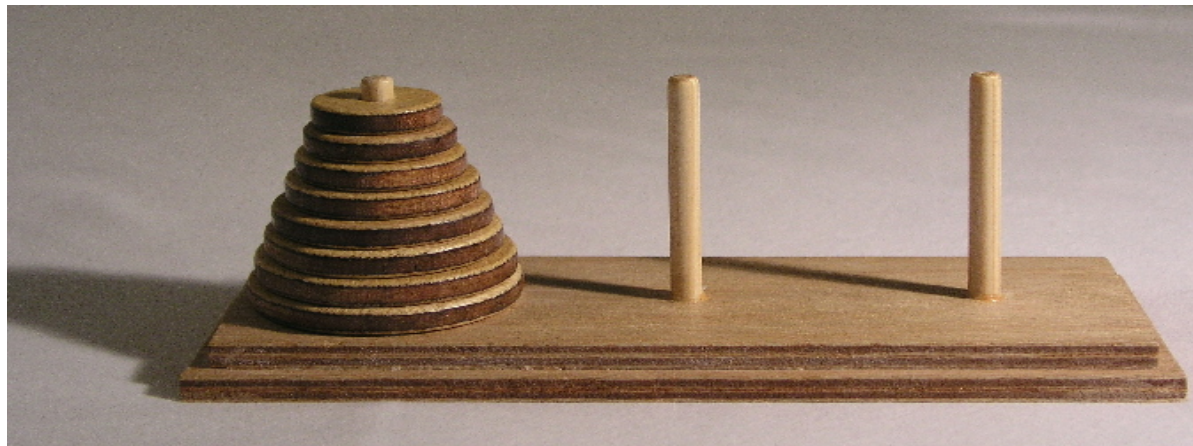


Tempo de execução(s)		
n	Iterativo	Recursivo
8	0.003	0.002
16	0.003	0.003
47	0.003	15.639
48	0.003	25.787
49	0.003	41.410

```
double fib(int n) {
    if (n <= 1)
        return n;
    return fib(n-1) + fib(n-2);
}
```

??

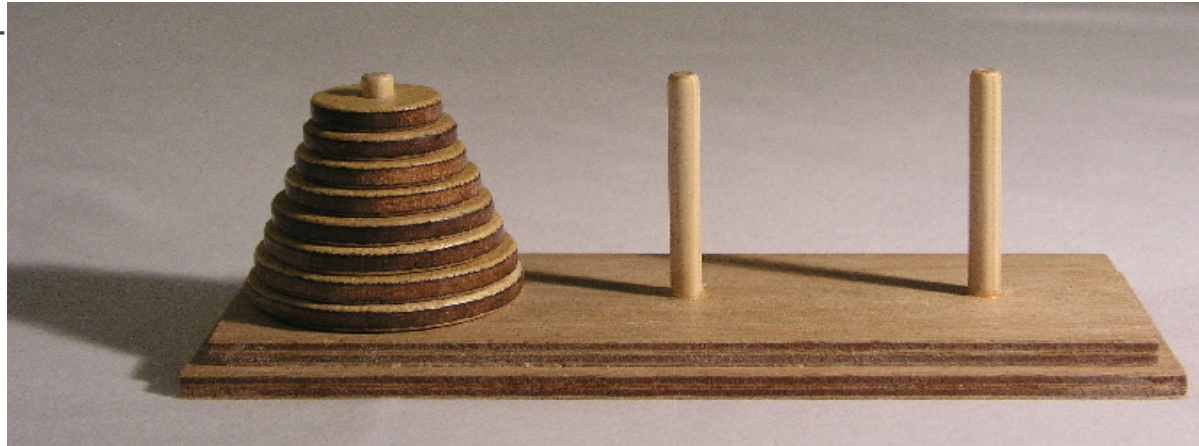
Recursividade



- Torre de Hanoi: jogo matemático muito famoso.
- Consiste em três pinos e um conjunto de discos com diferentes tamanhos.
- Inicialmente, os discos estão empilhados do maior para o menor no primeiro pino.
- O objetivo do jogo é transferir todos os discos para o terceiro pino seguindo as seguintes regras:



Torre de Hanoi



- Consiste em três pinos e um conjunto de discos com diferentes tamanhos.
- Inicialmente, os discos estão empilhados do maior para o menor no primeiro pino.
- O objetivo do jogo é transferir todos os discos para o terceiro pino seguindo as seguintes regras:
 - Apenas um disco pode ser movido por vez.
 - Um disco maior não pode ficar por cima de um menor.
 - Cada movimento realizado consiste em tirar um disco do topo de um dos pinos e passá-lo para o topo de outro pino...

Recursividade

- Torre de Hanoi:
<http://www.somatematica.com.br/jogos/hanoi/>
- Diz a lenda há um templo indiano com uma sala onde monges ficam jogando esse jogo. Nesse jogo, há 64 discos de ouro e, quando os monges terminarem de jogar, o mundo acabará.
- Pode-se provar que se os monges gastarem 1 segundo para mover cada pino e utilizarem o número mínimo possível de movimentos, seriam necessários apenas 600 bilhões de anos para o jogo acabar...



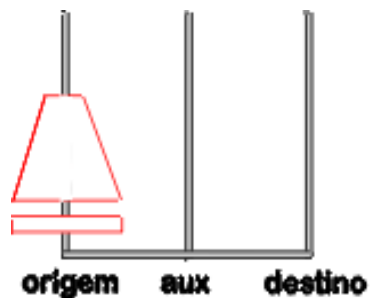
Recursividade

- Torre de Hanoi: algoritmo recursivo para resolver o problema com o número mínimo de movimentos.
- Suponha que se deseja mover n discos do pino A para o pino C utilizando o pino B como “auxiliar”:
 - Mova $n-1$ discos do pino A para o pino B utilizando o C como “auxiliar”. (chamada recursiva)
 - Mova o disco que sobrou no pino A para o pino C.
 - Mova $n-1$ discos do pino B para o pino C utilizando A como auxiliar. (chamada recursiva).
- O caso base do algoritmo acima consiste em mover 1 disco de um pino para outro (o que é trivial).
- São realizados exatamente $2^n - 1$ movimentos.

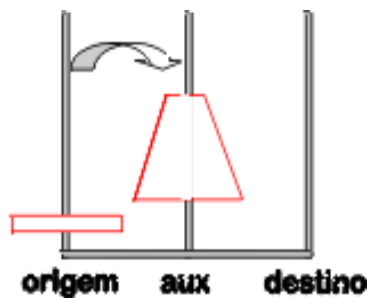


Recursividade

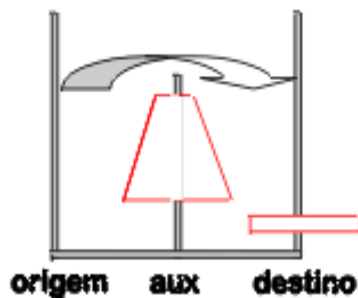
- Mova $n-1$ discos do pino A para o pino B utilizando o C como “auxiliar”. (recursão)
- Mova o disco que sobrou no pino A para o pino C.
- Mova $n-1$ discos do pino B para o pino C utilizando A como auxiliar. (recursão).



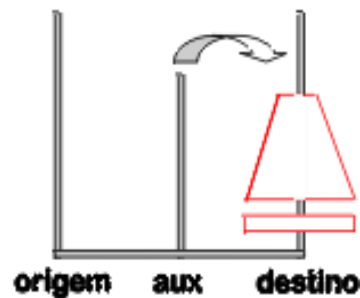
(a)



(b)



(c)



(d)

Fonte: notas de aula do professor Marcus Andrade.



Recursividade

- Implementação do algoritmo recursivo para o problema:

```
void hanoi(int n,int orig, int dest, int aux) {  
    if (n>0) {  
        hanoi(n-1,orig, aux, dest);  
        cout << "Mova: " << orig << " → " << dest << endl;  
        hanoi(n-1,aux, dest, orig);  
    }  
}
```

- Note que cada passo recursivo trata uma instância cada vez menor do problema até chegar em um caso base trivial.

Recursividade

- Exercícios:
- O algoritmo para o problema “torre de hanoi” realiza $2^n - 1$ movimentos (supondo n discos). Suponha que um computador capaz de realizar 1 milhão de “movimentos” por segundo tente destruir o mundo ao resolver o problema com 64 discos. Quanto tempo ele levaria para isso? (dica: $2^{64} = \sim 2 \times 10^{19}$)
- Mostre a sequência de chamadas recursivas para hanoi(3,1,2,3).
- Mostre a sequência de chamadas recursivas para a busca pelos números 31 e 15 no arranjo abaixo:

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99



Recursividade

- Exercícios:
- O algoritmo para o problema “torre de hanoi” realiza $2^n - 1$ movimentos (supondo n discos). Suponha que um computador capaz de realizar 1 milhão de “movimentos” por segundo tente destruir o mundo ao resolver o problema com 64 discos. Quanto tempo ele levaria para isso? (dica: $2^{64} = \sim 2 \times 10^{19}$) =>
- Mostre a sequência de chamadas recursivas para hanoi(3,1,2,3).
- Mostre a sequência de chamadas recursivas para a busca pelos números 31 e 15 no arranjo abaixo:

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99



Mostre a sequência de chamadas recursivas para hanoi(3,1,2,3).

```
void hanoi(int n,int orig, int dest, int aux) {  
    if (n>0) {  
        hanoi(n-1,orig, aux, dest);  
        cout << "Mova: " << orig << " → " << dest << endl;  
        hanoi(n-1,aux, dest, orig);  
    }  
}
```

Mostre a sequência de chamadas recursivas para a busca pelos números 31 e 15 no arranjo abaixo:

```
int pesquisaBinariaRec (int vet[], int chave, int begin, int end){  
    int meio = (begin + end)/2;  
    if (vet[meio] == chave)  
        return meio;  
    if (begin >= end)  
        return -1; // não encontrado  
    else{  
        if (vet[meio] < chave)  
            return pesquisaBinariaRec(vet, chave, meio+1, end);  
        else  
            return pesquisaBinariaRec(vet, chave, begin, meio-1);  
    }  
}
```

0	1	2	3	4	5	6	7
5	8	15	22	23	23	58	99