Recursão

Profa. Graziela Araújo
graziela.araujo@ufms.br

Algoritmos e Programação II

Motivação

- Conceito fundamental em computação
- Programas elegantes e mais curtos
- Equivalência entre programas recursivos – não-recursivos
- Memória

Recursividade

A definição de recursividade (recursão) aplica-se a funções e procedimentos. Por isso, vale (re)lembrar os seus conceitos:

- Função: é um módulo que produz um único valor de saída. Ela pode ser vista como uma expressão que é avaliada para um único valor, sua saída, assim como uma função em Matemática.
- Procedimento: é um tipo de módulo usado para várias tarefas, não produzindo valores de saída.

Como a diferença entre função e procedimento é sutil, utilizaremos os termos funções e procedimentos de forma indiscriminada durante o curso.

Recursividade

Até agora, foram vistos exemplos de funções chamadas genericamente de iterativas. Recebem este nome pois a repetição de processos neles inclusos fica explícita, através do uso de laços.

Um exemplo de função iterativa, para cálculo do fatorial de um número *n*, pode ser visto a seguir:

```
01    int Fatorial (int n){
02        int fat =1;    int i;
03         for (i =1; i <= n; i++)
04            fat = fat * i;
05
06        return fat;
07    }
```

Recursividade

- Alguns problemas têm uma estrutura recursiva: cada entrada do problema contém uma entrada menor do mesmo problema
- Estratégia:

  se a entrada do problema é pequena então
    resolva-a diretamente;
  senão,
    reduza-a a uma entrada menor do mesmo problema,
    aplique este método à entrada menor
    e volte à entrada original.

- Algoritmo recursivo, programa recursivo, função recursiva
- Uma função recursiva é aquela que possui uma ou mais chamadas a si mesma (chamada recursiva)

Recursividade

- Toda função deve possuir ao menos uma chamada externa a ela.
- Se todas as chamadas à função são externas, então a função é dita **não-recursiva**
- Em geral, a toda função recursiva corresponde uma outra não-recursiva equivalente
- A implementação de uma função recursiva pode acarretar gasto maior de memória, já que durante o processo de execução da função muitas informações devem ser guardadas na pilha de execução

FUNÇÃO ITERATIVA

```
int Fatorial (int n)
{
        int fat =1;         int i;
        for (i =1; i <= n; i++)
             fat = fat * i;

        return fat;
}

int main()
{
     int res, n = 5;
     // chamada externa
     res = Fatorial(n);
     printf("%d ", res);
}
```

CHAMADAS

Fatorial(3)

Fatorial(i)

Fatorial(3*n - 1)

Exemplos

- **Problema:**

> **Dado um número inteiro $n \geq 0$, computar o fatorial $n!$.**

Estratégia

Exemplos

- **Problema:**

> Dado um número inteiro n ≥ 0, computar o fatorial n!.

Usamos uma fórmula que nos permite naturalmente escrever uma função recursiva para calcular n! :

$$n! = \begin{cases} 1 & , \text{se } n \leq 1 \\ n \times (n-1)! & , \text{caso contrário} \end{cases}$$

Exemplos – Solução 1

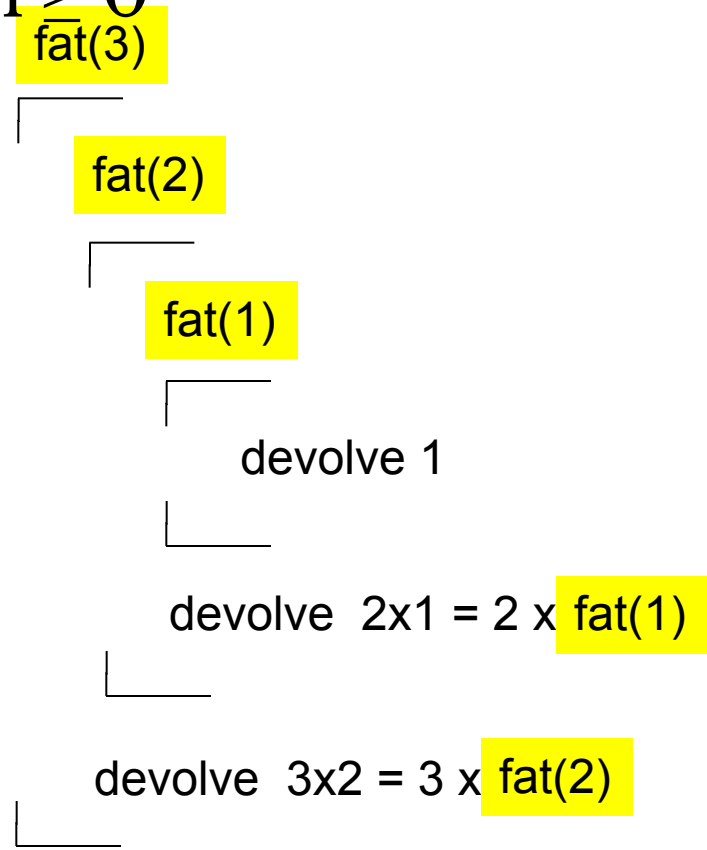
Suponha a chamada externa **fat(3)**

```
/* Recebe um número inteiro n >= 0 e devolve o fatorial de n */
int fat(int n)
{
   int result;

   if (n <= 1)
      result = 1;
   else
      result = n * fat(n-1);

   return result;
}
```

Exemplos – Solução 2

/* Recebe um número inteiro n ≥ 0
e devolve o fatorial de n */

```
/* Recebe um número inteiro n >= 0
int fat(int n)
{
   if (n <= 1)
      return 1;
   else
      return n * fat(n-1);
}
```



Exemplos

- Os procedimentos recursivos possuem uma descrição mais clara e concisa, fazendo com que o código do programa que o implemente seja “enxuto”.

- Por outro lado, os programas que implementam procedimentos recursivos tendem a ser mais custosos (em termos de memória e tempo) que aqueles que implementam a versão iterativa correspondente.

- Além disso, a depuração de programas recursivos é um pouco mais complicada que a de programas iterativos.

Exemplos

- **Problema:**

Dado um número inteiro *n* > 0 e uma sequência de *n* números inteiros armazenados em um vetor *v*, determinar um valor máximo em *v*.

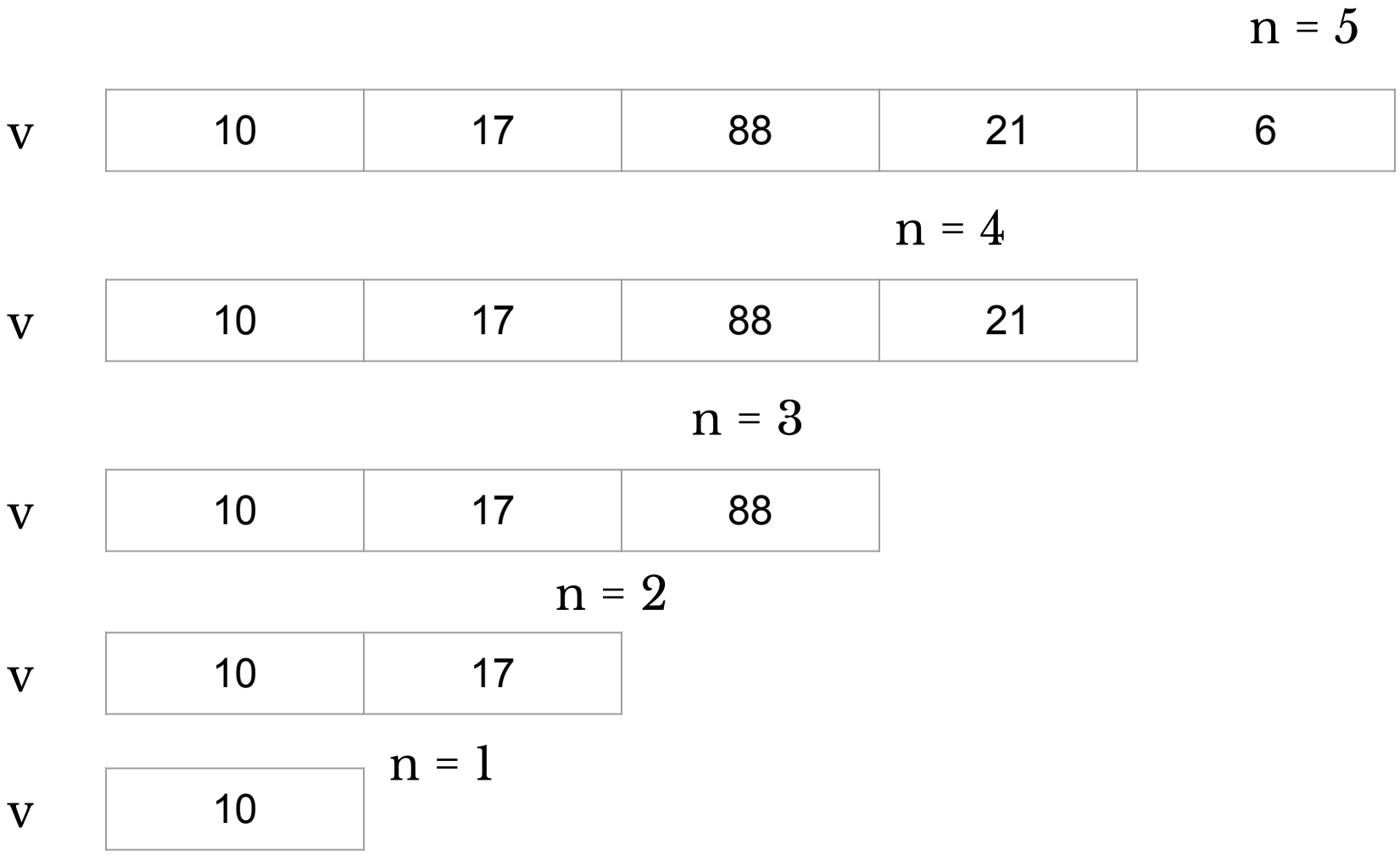
PROTÓTIPO DA FUNÇÃO:

```
int maximo(int n, int v[MAX]);
```

n -- ?
v -- ?

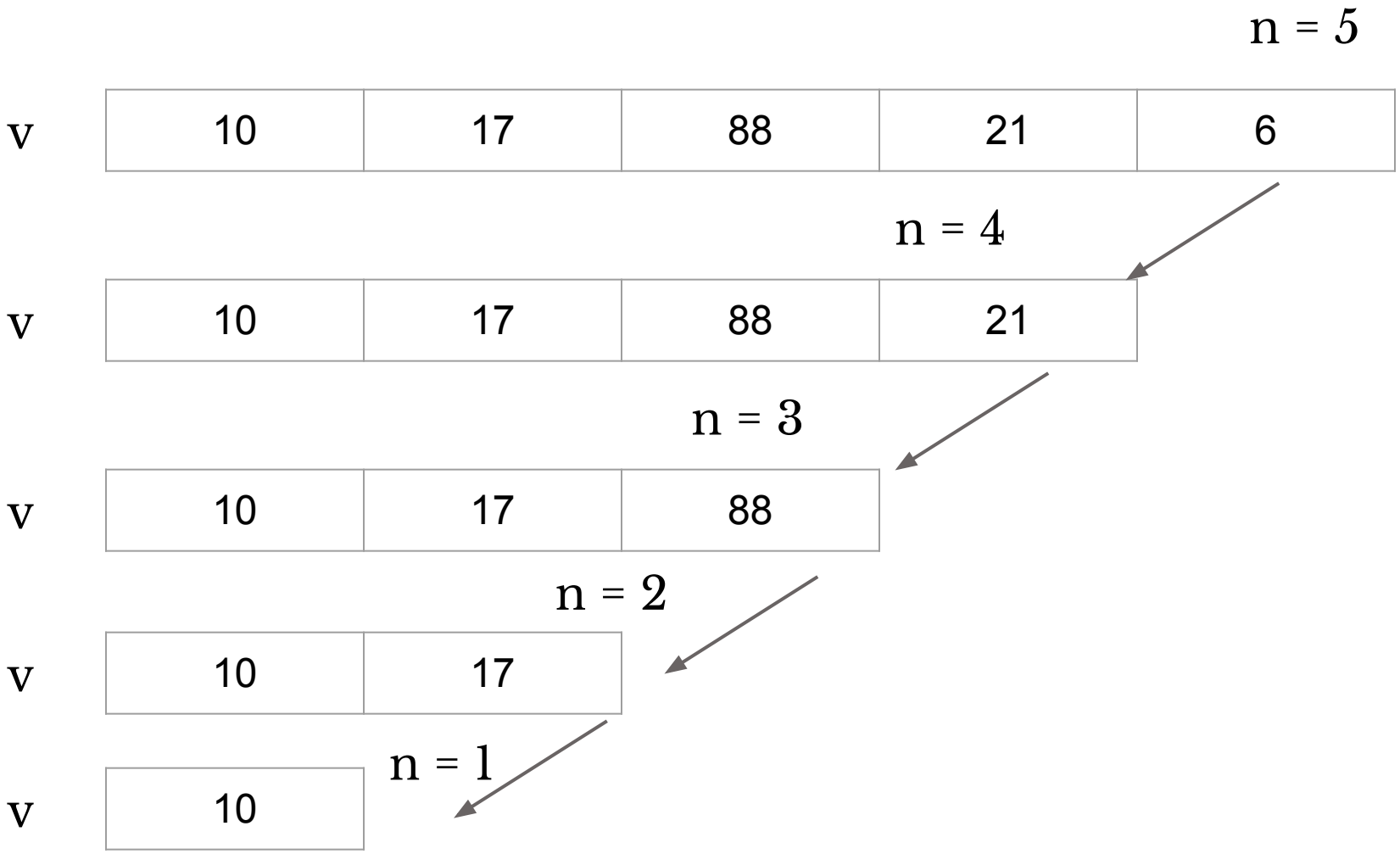
Função e chamada da função

int maximo(int n, int v[MAX])



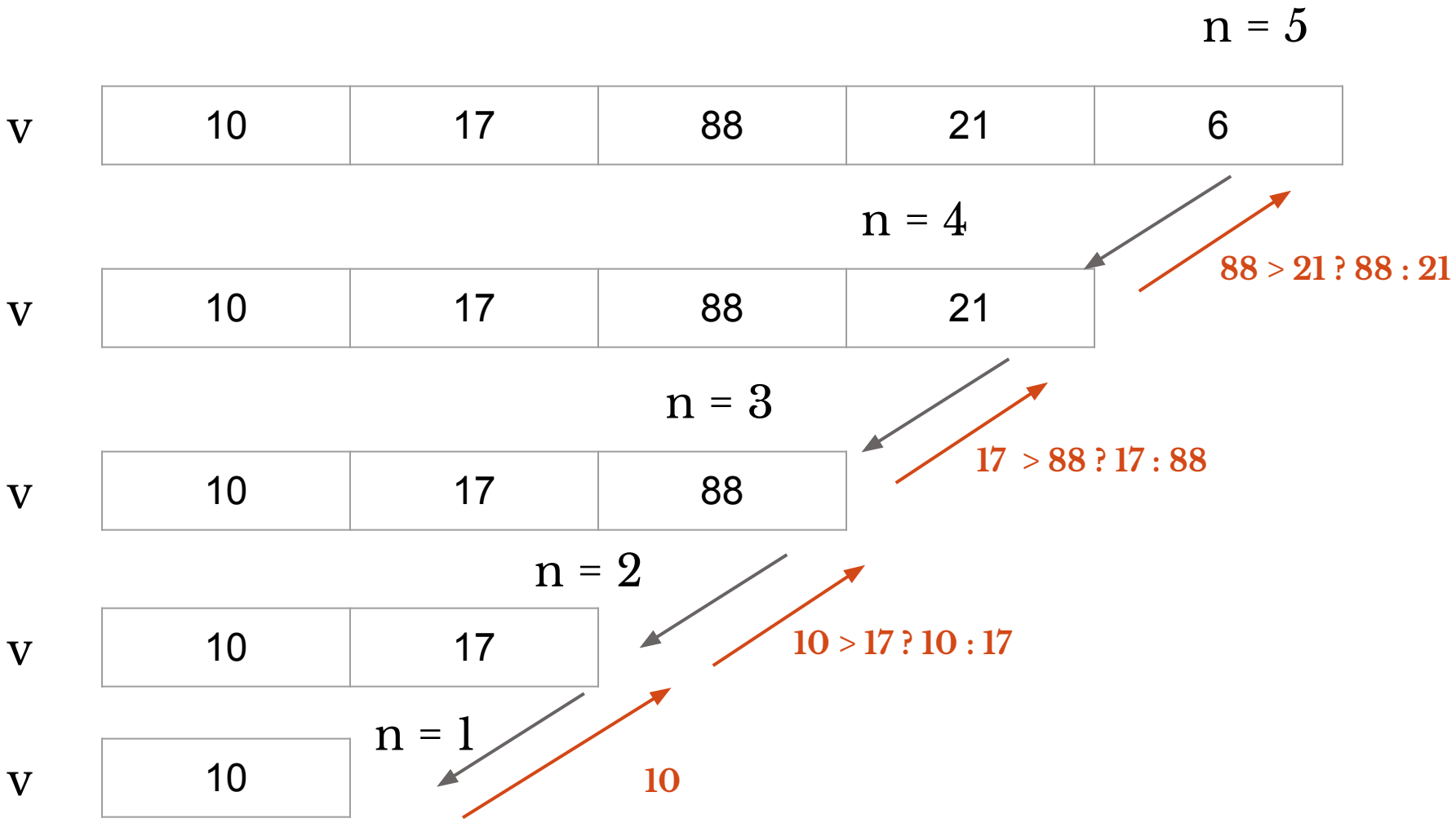
ESTRATÉGIA

int maximo(int n, int v[MAX])



ESTRATÉGIA

int maximo(int n, int v[MAX])



Exemplos

```
/* Recebe um número inteiro n > 0 e um vetor v de números in-
   teiros com n elementos e devolve um elemento máximo de v */
int maximo(int n, int v[MAX])
{
   int aux;

   if (n == 1)
      return v[0];
   else {
      aux = maximo(n-1, v);
      if (aux > v[n-1])
         return aux;
      else
         return v[n-1];
   }
}
```

Exemplos

- **Problema:**

Dado um número inteiro n, calcular e imprimir a soma dos dígitos de n.

Escreva uma função recursiva que dado **n**, retorna a soma dos dígitos de **n**.

```
int somadigitos(int n);
```