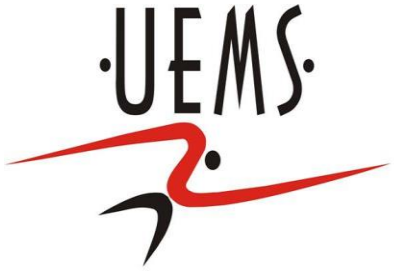


Programação de Computadores I – Recursão

Profa. Mercedes Gonzales Márquez

Recursão

- A solução de um problema é dita recursiva quando ela é escrita em função de si própria para instancias menores do problema.
- A solução recursiva está diretamente ligada ao conceito de indução matemática.
- Na indução matemática define-se a solução do problema para casos básicos e usa-se como hipótese que a solução do problema de tamanho n pode ser obtida a partir da sua solução de tamanho menor, por exemplo, n-1.

Recursão

- Implementação iterativa do cálculo de fatorial de um número.

Implementação não Recursiva

```
long fat(long n){
long r = 1;
for(int i = 1; i <= n; i++)
        r = r * i;
return r;
}
```

Recursão

- A solução do problema também pode ser expressa de forma recursiva

    - Se n = 1 então n! = 1.

    - Se n > 1 então n! = n (n-1)!.

- Aplicamos o princípio da indução assim:

- Sabemos a solução para um caso base: n = 1.

- Definimos a solução do problema geral n! em termos do mesmo problema só que para um caso menor ((n-1)!).

Recursão

Implementação Recursiva

```
long fat(long n){
   if(n <= 1) //Passo Basico
          return 1;
   else //Sabendo o fatorial de (n-1)
//calculamos o fatorial de n
      return (n* fat(n-1));
}
```

Recursão

Toda recursão é composta por:

- **Caso base**

– Uma ou mais instâncias do problema que podem ser solucionadas facilmente (solução trivial)

- **Chamadas Recursivas**

– O objeto define-se em termos de si próprio, procurando convergir para o caso base.

Por exemplo, o fatorial de um número n pode ser calculado a partir do fatorial do número n−1.

Recursão

- Esquematicamente, os algoritmos recursivos tem a seguinte forma:

  Se <condição para o caso base> então

  resolução direta para o caso base

  Senão

  uma ou mais chamadas recursivas

  Fim se

- Sem a condição de parada, expressa no caso base, uma recursão iria se repetir indefinidamente.

Sequência de Fibonacci

fib(1) = 0 fib(2) = 1

fib(3) = fib(2) + fib(1)

fib(4) = fib(3) + fib(2)

Fib(n) = fib(n-1) + fib(n - 2)

Sequência de Fibonacci – Definição Recursiva

- A serie de fibonacci e a seguinte:

  1; 1; 2; 3; 5; 8; 13; 21; ...

- Queremos determinar qual e o n-ésimo (fibo(n)) número da serie.
- Precisamos descrever o n-ésimo número de fibonacci de forma recursiva.

Sequência de Fibonacci – Definição Recursiva

fibo(1) = 0 fibo(2) = 1

fibo(3) = fibo(2) + fibo(1)

fibo(4) = fibo(3) + fibo(2)

fibo(n) = fibo(n-1) + fibo(n - 2)

Assim:

- No caso base temos:

  - Se n = 1 ou n = 2 então fibo(n) = 1.

  - Sabendo casos anteriores podemos computar fibo(n) como: fibo(n) = fibo(n-1) + fibo(n-2).

Sequência de Fibonacci – Definição Recursiva

```
long fibo(long n){
if(n <= 2)
        return 1;
else
        return (fibo(n-1) + fibo(n-2));
}
```

1, 1, 2, 3, 5, 8, 13, 21, 34, ...

Recursão

Exemplo 3. Soma de elementos de um vetor : Faça um algoritmo que preencha por leitura um vetor de 10 elementos inteiros, imprima o seu conteúdo, e o resultado do somatório dos seus elementos, calculado por uma função recursiva.

Vamos denotar por S(n) a soma dos elementos das posições 0 ate n do vetor. Com isso temos:

- Se n = 1 então a soma e igual a v[0].
- Se n > 0 então a soma e igual a v[n-1] + S(n-1).

```
int soma(int n, int v[]){
   if (n == 1)
         return (v[0]);
   else
         return (v[n-1]+soma(n-1, v));
}
```

Recursão

Exemplo 4. Escreva uma função recursiva que recebe como parâmetros um número real **X** e um inteiro **n** e retorna o valor de $\mathbf{X^n}$.

- **Obs.:n** pode ser negativo.

```
float Potencia(float x,int n){
if  ( n ==0 ) return (1);
else
   if ( n <0 ) return (1 / (x* Potencia(x, abs(n)-1)));
   else return (x* Potencia(x, n -1));
}
```

Recursão

- Exemplo 5. Reescreva a função abaixo tornando-a recursiva. Esta função conta o número de algarismos (dígitos) que possui um número inteiro n.

```
inteiro digitos(int n){
int cont
cont=1;
while (abs(n )>9) {
        n = n/10;
        cont ++;
}
return (cont)
}
```

Recursão

```
int digitos(int n){
    if (abs( n )<10)
        return (1);
    else
        return (1+ digitos(n/10));
}
```

Recursão

- Exemplo 6. Escreva uma função recursiva que calcule a soma dos dígitos de um inteiro positivo n. A soma dos dígitos de 132, por exemplo, é 6.

```
int somadigitos(int n){
if (n <10) return (n);
else return (n%10+ somadigitos(n/10));
}
```

Recursão

- Exemplo 7. Escreva uma função recursiva que determine o inverso de um número inteiro positivo n, dado o número de dígitos de n. O inverso é obtido pela inversão dos dígitos do número. O inverso do 132, por exemplo, é 231.

```
int inverso(int n, int ndig){
if (n <10) return (n);
else retorne ((n%10)*pow(10,ndig-1)+ inverso(n/10,ndig-1));
}
```

Recursão

- Exemplo 8. Verificar se um número é capicua utilizando uma função recursiva que considere como entrada o número inteiro positivo n e o número de algarismos de n. Um número capicua é aquele número que coincide com seu inverso, de acordo a definição do exemplo 7.

```
int capicua(int n, int ndig){
if (n <10) return (1);
else
   if ((n%10)!=n/(int)pow(10,ndig-1)) return (0);
   capicua((n%(int)pow(10,ndig-1)/10),ndig-2);
}
```

Recursão

Exemplo 9- Faça um algoritmo que realize uma busca binária em um vetor ordenado de elementos.

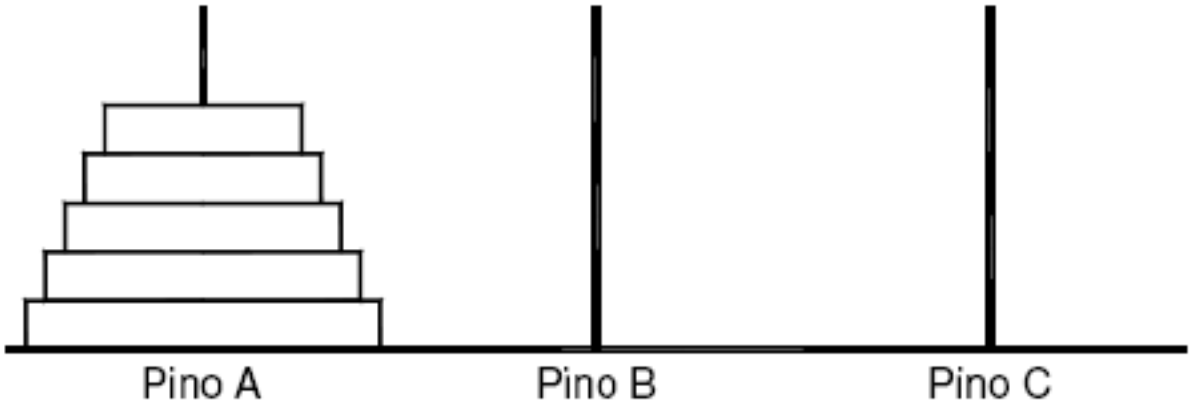
- Divide seu vetor em duas metades
- Três condições
  1. Se o item for igual ao item que está na metade do vetor, o item foi encontrado
  2. Se for menor, procure na primeira metade
  3. Se for maior procure na segunda metade

Busca Binária

```
Void Busca_Binária(int x, int inicio, int fim){
int meio
        meio=(inicio + fim)/ 2;
        if (fim < inicio) printf ("Elemento Não Encontrado");
        else
            if  (v[meio] ==x) printf ("Elemento está na posição %d",meio);
            else
                if (v[meio] < x){
                        inicio=meio +1;
                        Busca_Binaria (x, inicio, fim);
                }else{
                        fim=meio - 1;
                        Busca_Binaria (x, inicio, fim);
                }
}
```

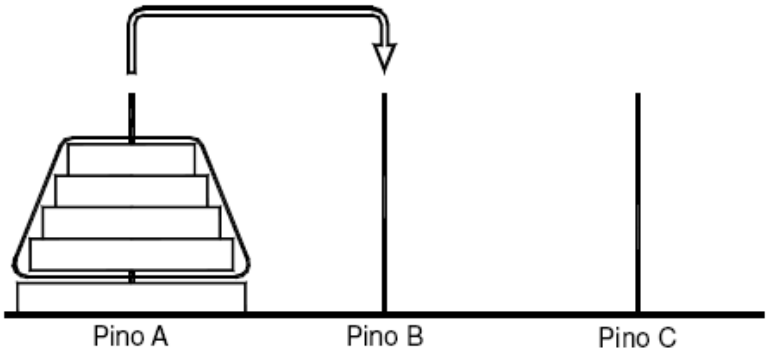
Recursão

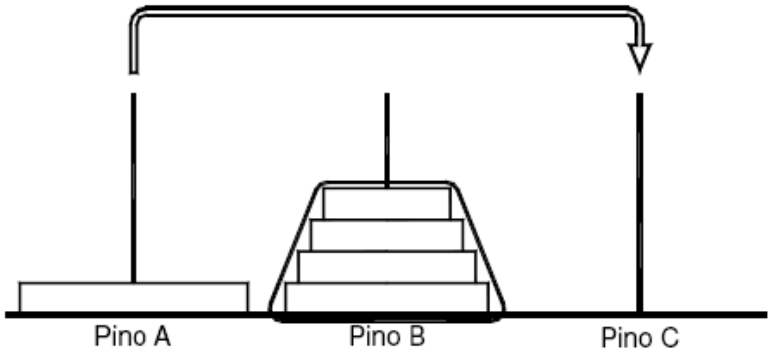
Exemplo 10-Problema da Torre de Hanói
O problema ou quebra-cabeça conhecido como torre de Hanói consiste em transferir, com o menor número de movimentos, a torre composta por *N* discos do pino **A** (origem) para o pino **C** (destino), utilizando o pino **B** como auxiliar. Somente um disco pode ser movimentado de cada vez e um disco não pode ser colocado sobre outro disco de menor diâmetro.

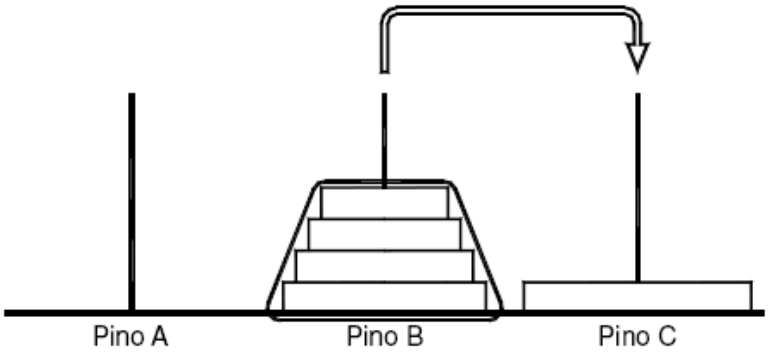


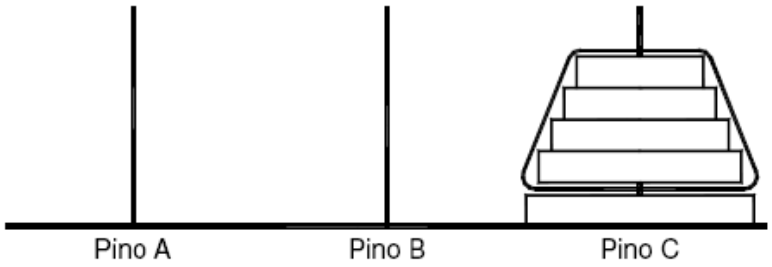
Recursão

Solução: Transferir a torre com *N*-1 discos de **A** para **B**, mover o maior disco de **A** para **C** e transferir a torre com *N*-1 de **B** para **C**. Embora não seja possível transferir a torre com *N*-1 de uma só vez, o problema torna-se mais simples: mover um disco e transferir duas torres com *N*-2 discos. Assim, cada transferência de torre implica em mover um disco e transferir de duas torres com um disco a menos e isso deve ser feito até que torre consista de um único disco.









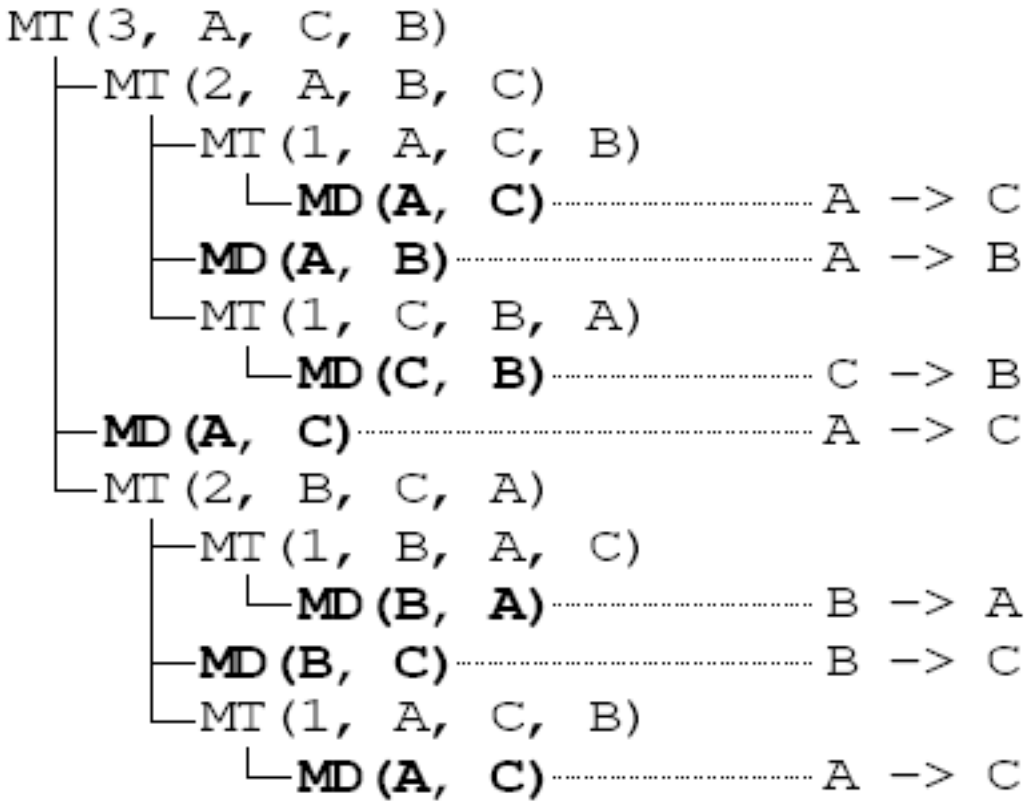
Recursão

```
vazio MoveTorre(inteiro:n, literal: Orig, Dest, Aux)
início
   se n = 1 então
      MoveDisco(Orig, Dest)
   senão
      MoveTorre(n - 1, Orig, Aux, Dest)
      MoveDisco(Orig, Dest)
      MoveTorre(n - 1, Aux, Dest, Orig)
   fim se
fim
vazio MoveDisco(literal:Orig, Dest)
início
Escreva(“Movimento: ”, Orig, “ -> ”, Dest)
fim
```

Recursão

Uma chamada a MoveTorre(3, ‘A’, ‘C’, ‘B’) teria a seguinte saída:

Movimento: A -> C
Movimento: A -> B
Movimento: C -> B
Movimento: A -> C
Movimento: B -> A
Movimento: B -> C
Movimento: A -> C



Recursão

Implemente o procedimento MoveTorre.

Recursão

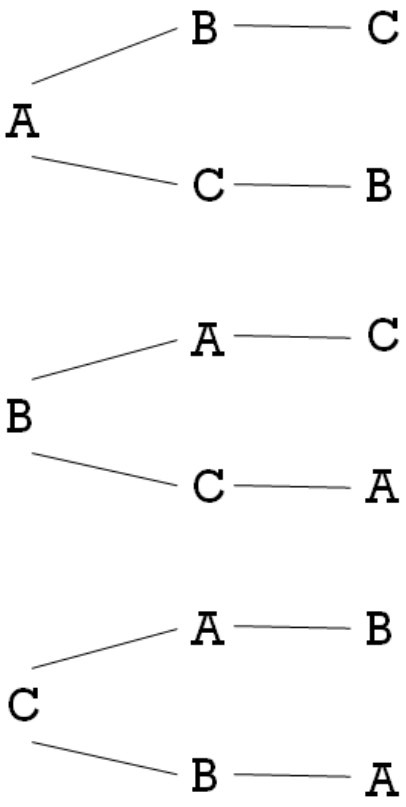
Exemplo 11-

A recursividade pode ser utilizada para gerar todas as possíveis permutações de um conjunto de símbolos. Por exemplo, existem seis permutações no conjunto de símbolos A, B e C: ABC, ACB, BAC, BCA, CBA e CAB**.**

O conjunto de permutações de *N* símbolos é gerado tomando-se cada símbolo por vez e prefixando-o a todas as permutações que resultam dos (*N* - 1) símbolos restantes. Consequentemente, permutações num conjunto de símbolos podem ser especificadas em termos de permutações num conjunto menor de símbolos. Formule um algoritmo recursivo para este problema.

Recursão

Solução: O número total de permutações de n objetos é dado por **P(n) = n!**. Ou seja: para 3 objetos P(3) = 3! = 3 x 2 x 1 = 6 permutações. Para 4 objetos temos: P(4) = 4! = 4 x 3 x 2 x 1 = 24 permutações. O recurso básico para a elaboração do algoritmo consiste em montar uma **árvore de recursão** para uma “instância” de solução para o problema, por exemplo 3 objetos.



Recursão

```
#include <stdio.h>
#include <string.h>
void permutate(char string[], int k, int n){
    int i;
    char temp;

    if(k == (n-1)){
        printf ("%s\n",string);
    }
    else{
        for(i = k;   i < n; i++){
            temp = string[i];
            string[i] = string[k];
            string[k] = temp;
            permutate(string, (k+1), n);
            temp = string[k];
            string[k] = string[i];
            string[i] = temp;
        }
    }
```

```
int main (){
    char str[10];

strcpy(str,"AB");
permutate(str,0,2);
printf ("\n");

strcpy(str,"ABC");
permutate(str,0,3);
printf ("\n");

strcpy(str,"ABCD");
permutate(str,0,4);

return 0;
}
```

Recursão

- Na chamada recursiva :

  Para cada posição i de vetor de caracteres, entre k e o fim do vetor:

  1. Troque os caracteres das posições i e k.
  2. Gere todas as permutações do nível k+1, ou seja, todas as permutações de n-1 caracteres.
  3. Restaure o vetor original trocando as posições i e k.

Recursão

Exemplo 12-

O algoritmo Merge Sort tem como objetivo a ordenação de um vetor A de n elementos reais. A sua estratégia é baseada no paradigma dividir para conquistar.

1. Passo Dividir

Se um dado vetor *A* tem um ou nenhum elemento, simplesmente retorne pois ele já está ordenado. Caso contrário, divida $A[p .. r]$ em dois subvetores $A[p .. q]$ e $A[q + 1 .. r]$, cada um contendo a metade dos elementos de $A[p .. r]$. q é o índice do médio de $A[p .. r]$.

2. Passo Conquistar

Ordene recursivamente os dois subvetores $A[p .. q]$ e $A[q + 1 .. r]$.

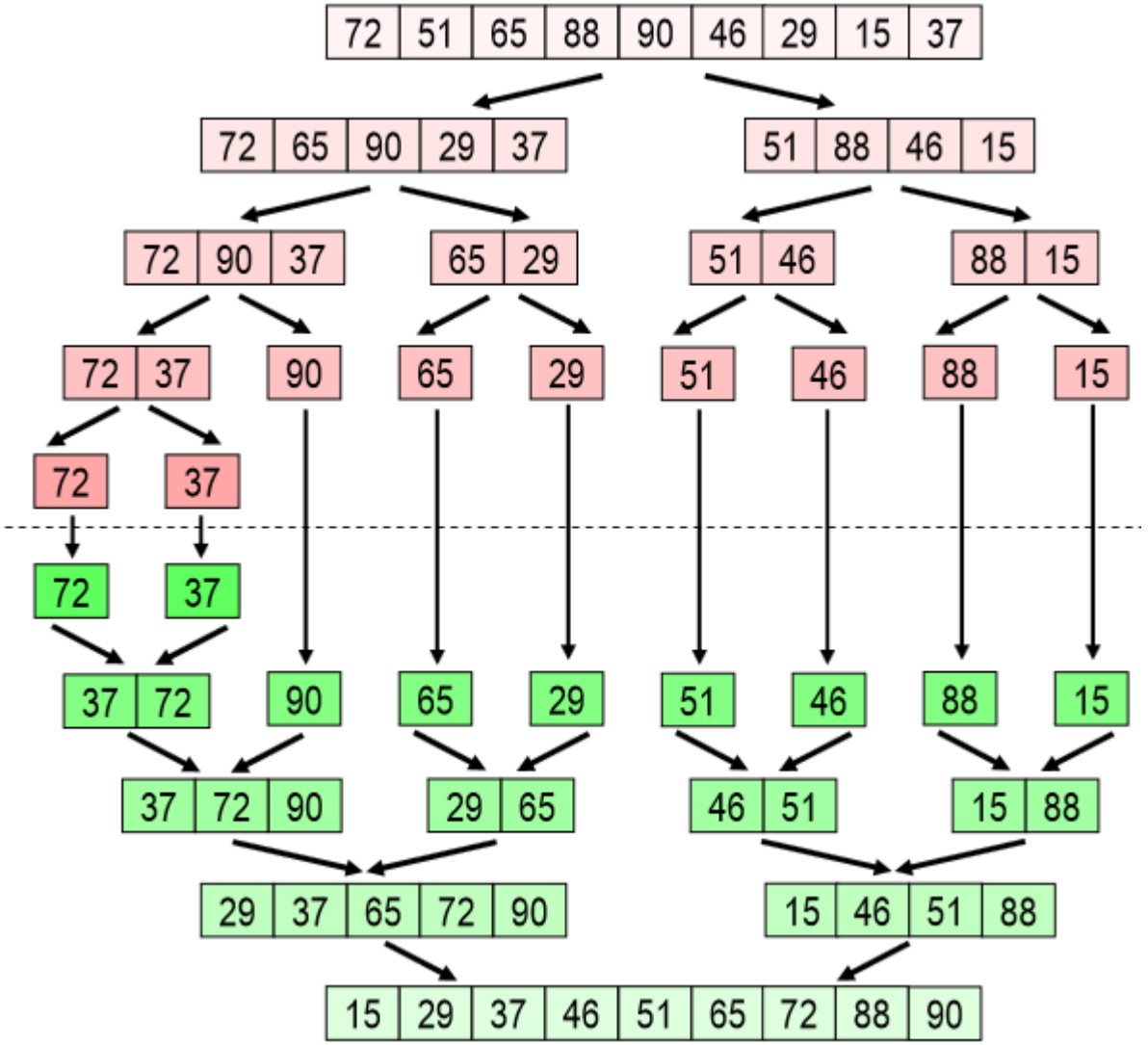
3. Passo combinar ou intercalar

Combine os elementos de volta em $A[p .. r]$ mesclando os subvetores ordenados $A[p .. q]$ e $A[q + 1 .. r]$ em uma sequência ordenada. Para realiza esta etapa vamos definir um procedimento Merge (A, p, q, r).

Recursão

```
void intercala( int p, int q, int r, int v[]) {
  int i, j, k, *w;
  w = malloc( (r-p) * sizeof (int));
  i = p; j = q; k = 0;
  while (i < q && j < r) {
        if (v[i] <= v[j]) w[k++] = v[i++];
        else w[k++] = v[j++]; }
  while (i < q) w[k++] = v[i++];
  while (j < r) w[k++] = v[j++];
  for (i = p; i < r; ++i) v[i] = w[i-p];
  free( w);
  }
 Void mergesort( int p, int r, int v[]) {
  if (p < r-1) {
    int q = (p + r)/2;
    mergesort( p, q, v);
    mergesort( q, r, v);
    intercala( p, q, r, v); }
    }
```

Recursão


72 51 65 88 90 46 29 15 37
72 65 90 29 37
51 88 46 15
72 90 37
65 29
51 46
88 15
72 37
90
65
29
51
46
88
15
72
37
72
37
37 72
90
65
29
51
46
88
15
37 72 90
29 65
46 51
15 88
29 37 65 72 90
15 46 51 88
15 29 37 46 51 65 72 88 90