

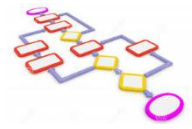


Unidade 10 – Técnicas de Projeto de Algoritmos



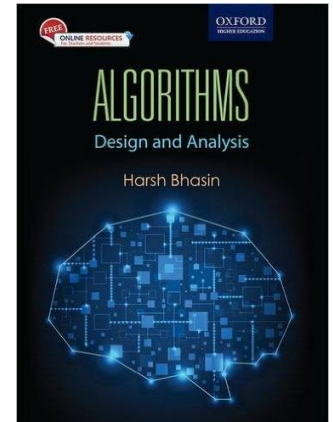
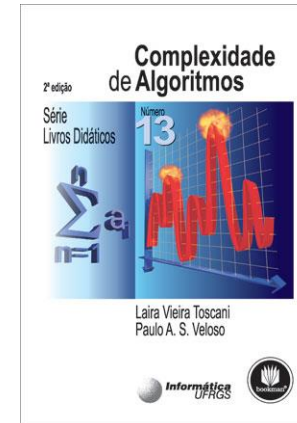
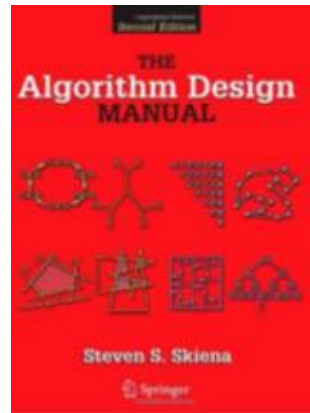
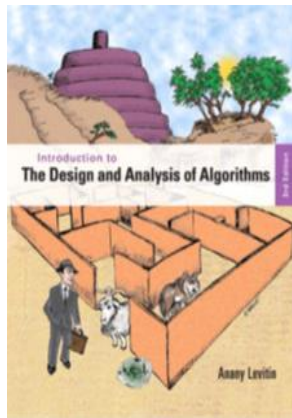
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUVSP
aparecidovfreitas@gmail.com





Bibliografia

- Algorithm Design and Applications – Michael T. Goodrich, Roberto Tamassia, Wiley, 2015
- Introduction to the Design and Analysis of Algorithms – Anany Levitin, Pearson, 2012
- The Algorithm Design Manual – Steven S. Skiena, Springer, 2008
- Complexidade de Algoritmos – Série Livros Didáticos – UFRGS
- Algorithms – Design and Analysis – Harsh Bhasin – Oxford University Press - 2015

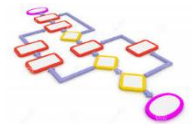




Técnica Força Bruta

- ◆ É uma técnica trivial de soluções de problemas, porém muito geral, que consiste em enumerar todos os possíveis candidatos da solução e checar cada um para saber se ele satisfaz ao enunciado do problema.





Técnica Força Bruta

Exemplo 1 – Máximo valor de uma lista

```
package paa;

public class ForcaBruta_1 {

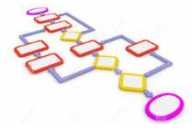
    public static void main(String[] args) {
        int[] lista = { 4,6,1,9,5 } ;
        System.out.println("Maximo valor da lista: " + maximo(lista));
    }

    public static Integer maximo (int[] lista) {

        int maximo = lista[0];

        for (int i = 1; i < lista.length; i++)
            if( maximo < lista[i] )
                maximo = lista[i];
        return maximo;
    }
}
```





Técnica Força Bruta

Exemplo 2 – Soma dos valores de uma lista

```
package paa;
```

```
public class ForcaBruta_2 {
```

```
    public static void main(String[] args) {  
        int[] lista = { 4,6,1,9,5 } ;  
        System.out.println("Soma dos valores da lista: " + soma(lista));  
    }
```

```
    public static Integer soma (int[] lista) {
```

```
        int soma = lista[0];
```

```
        for (int i = 1; i < lista.length; i++)  
            soma = soma + lista[i];  
        return soma;
```

```
    }
```



Técnica Força Bruta

Exemplo 3 – Cálculo do Fatorial

```
package paa;

public class ForcaBruta_3 {

    public static void main(String[] args) {
        int n = 10;
        System.out.println("Fatorial de " + n + ": " + fat(n));
    }

    public static Long fat (int n) {

        Long fat = 1L;

        for (int i = 2; i <= n; i++)
            fat = fat * i;
        return fat;
    }
}
```

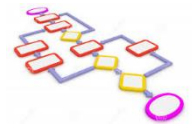




Técnica Divisão e Conquista

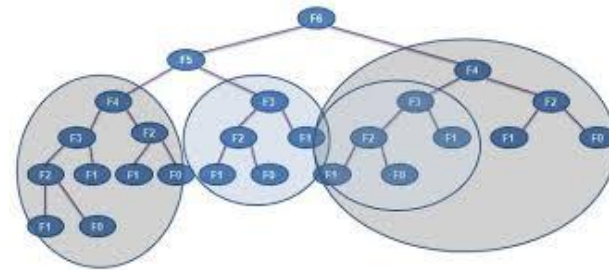
- ◆ É provavelmente uma das mais conhecidas técnicas de projeto de algoritmos;
- ◆ Algoritmos divide-and-conquer funcionam de acordo com um plano geral.





Técnica Divisão e Conquista

- ◆ Dado um problema com uma entrada grande, quebra-se a entrada em porções menores (**DIVISÃO**);
- ◆ Resolve-se cada porção separadamente (**CONQUISTA**);
- ◆ Combinam-se os resultados.





Algoritmos Divisão e Conquista

- ◆ São, em geral, recursivos;
- ◆ Requer um número menor de acessos à memória;
- ◆ São altamente paralelizáveis. Se existirem vários processadores disponíveis, a estratégia propicia eficiência.

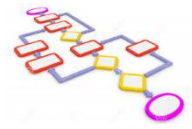




Quando empregar Divisão e Conquista ?

- ◆ Quando for possível decompor-se a instância em sub-instâncias;
- ◆ Problemas onde um grupo de operações são correlacionadas ou repetidas.





Passos do Algoritmo Divisão e Conquista



Dividir

PROBLEMA

Problema

Problema

Problema

Problema

Problema

Conquistar

Solução

Solução

Solução

Solução

Solução

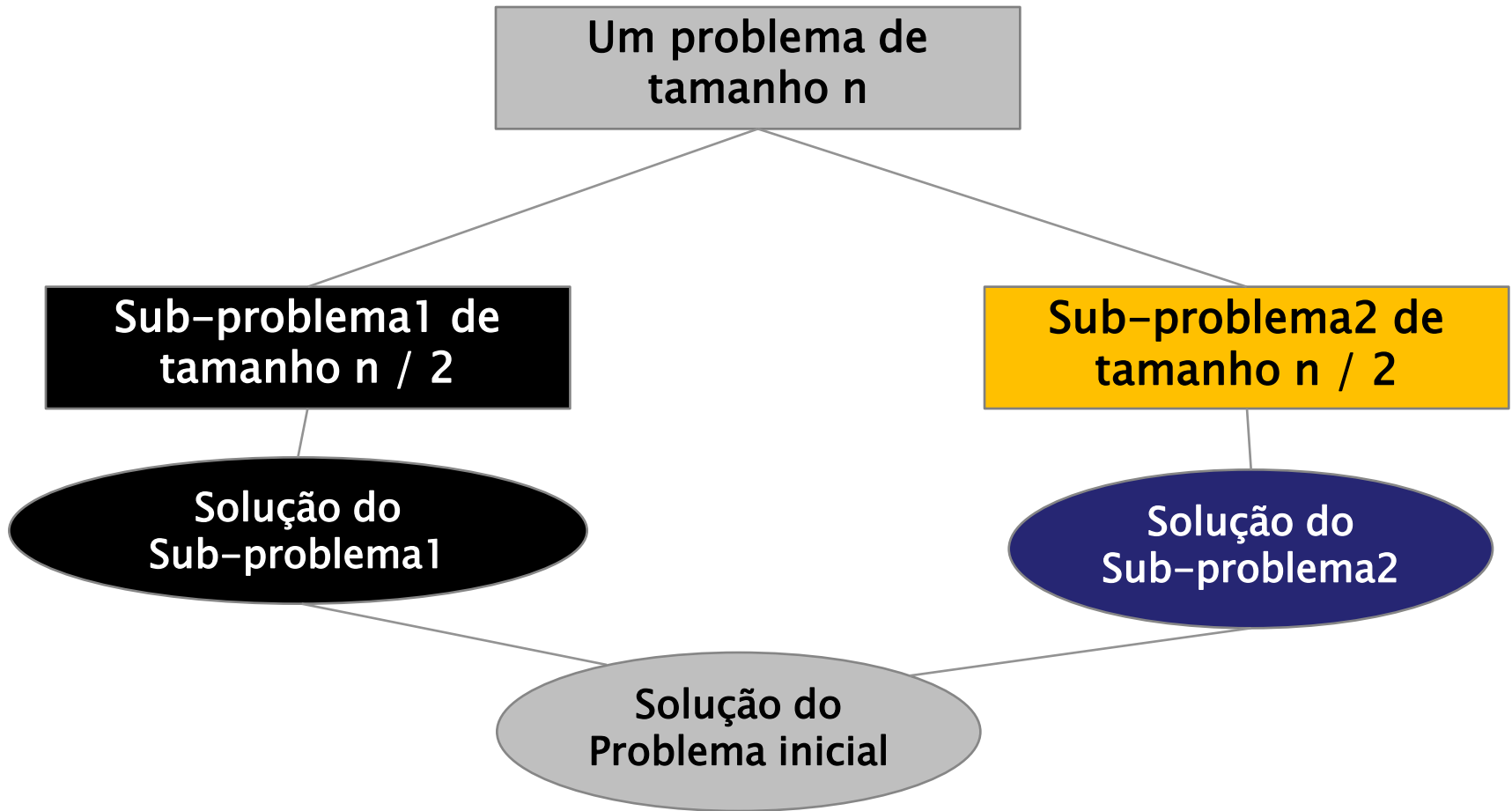
Combinar

Solução





Técnica Divisão e Conquista





Técnica Divisão e Conquista

Algoritmo Genérico

Divisão e Conquista(x)

```
if x é pequeno ou simples do
    return resolver(x)
else
    decompor x em conjuntos menores  $x_0, x_1, \dots, x_n$ 
    for  $i \leftarrow 0$  to  $n$  do
         $y_i \leftarrow \text{Divisão e Conquista}(x_i)$ 
         $i \leftarrow i + 1$ 
    combinar  $y_i$ 's

return y
```





Técnica Divisão e Conquista

Exemplo 1- Máximo elemento de uma lista

O problema consiste em encontrar o maior elemento de um array $a[1..n]$



3	8	1	7	6	2	9	4
---	---	---	---	---	---	---	---

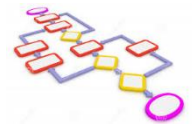


Exemplo 1- Máximo elemento de uma lista

Solução – Força Bruta

```
for (int i = 1; i < lista.length; i++)  
    if( maximo < lista[i] )  
        maximo = lista[i];  
return maximo;
```





Técnica Divisão e Conquista

Exemplo 1 – Máximo elemento de uma lista – Pseudocódigo

```
maximo (lista, int inicio, int fim)
```

```
    if ( inicio = fim )  
        retorna lista(inicio)
```

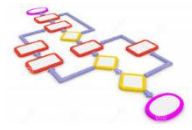
```
    meio = ( inicio + fim ) / 2;
```

```
    m1 = maximo(lista, inicio, meio)
```

```
    m2 = maximo(lista, meio+1, fim);
```

```
    if (m1 >= m2) retorna m1;  
    else retorna m2;
```





Técnica Divisão e Conquista

Exemplo 1 – Máximo elemento de uma lista



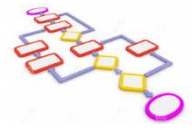
```
package paa;

public class DivConquista_1 {

    public static void main(String[] args) {
        int[] lista = { 4,133,1,9,5,99,2 } ;
        int indiceInicio = 0;
        int indiceFim = lista.length-1;

        System.out.println("Maximo valor da lista: "
            + maximo(lista, indiceInicio, indiceFim));
    }
}
```





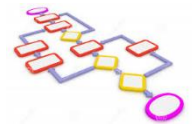
Técnica Divisão e Conquista

Exemplo 1 – Máximo elemento de uma lista



```
public static Integer maximo (int[] lista, int indiceInicio, int indiceFim) {  
  
    if ( indiceInicio == indiceFim )  
        return lista[indiceInicio] ;  
  
    int indiceMetade = ( indiceInicio + indiceFim ) / 2;  
  
    Integer m1 = maximo(lista, indiceInicio, indiceMetade);  
    Integer m2 = maximo(lista, indiceMetade+1, indiceFim);  
  
    if (m1 >= m2) return m1;  
    else return m2;  
}
```





Técnica Divisão e Conquista

Exemplo 2 – Soma dos valores de uma lista



```
package paa;

public class DivConquista_2 {

    public static void main(String[] args) {

        int[] lista = { 4,6,1,9,5,100 } ;

        int indiceInicio = 0;
        int indiceFim = lista.length-1;

        System.out.println("Soma dos valores da lista: " +
            soma(lista, indiceInicio, indiceFim));
    }
}
```





Técnica Divisão e Conquista

Exemplo 2 – Soma dos valores de uma lista



```
public static Integer soma (int[] lista, int indiceInicio, int indiceFim) {  
  
    if (indiceInicio == indiceFim )  
        return lista[indiceInicio];  
  
    int indiceMetade = (indiceInicio + indiceFim)/2;  
  
    int s1 = soma(lista, indiceInicio, indiceMetade);  
  
    int s2 = soma(lista, indiceMetade+1, indiceFim);  
  
    return (s1 + s2);  
}
```





Técnica Divisão e Conquista

Exemplo 3 – Exponenciação

$$\text{base} \leftarrow a^n \leftarrow \text{expoente}$$

O problema consiste em computar x^n .



$$\text{base} \leftarrow x^n \leftarrow \text{expoente}$$



Exemplo 3

Solução Força Bruta – Exponenciação

Pow(a, n)

p ← a

for i ← 2 to n do

p ← p × a

return p



Exemplo 3

Solução Força Bruta – Exponenciação



```
package paa;

public class ForcaBruta_4 {

    public static void main(String[] args) {
        int x = 3;
        int n = 4;
        System.out.println("Potencia de " + x + " elevado a " + n + " : " + pot(x,n));
    }

    public static Long pot (int x , int n) {

        long pot = x;

        for (int i = 2; i <= n; i++)
            pot = pot * x;

        return pot;
    }
}
```



Solução Divisão e Conquista

Exemplo 3 – Exponenciação

Pow(a, n)

```
if n=0 then  
    return 1  
if n é par then  
    return Pow(a, n/2) × Pow(a, n/2)  
else  
    return Pow(a, (n-1)/2) × Pow(a, (n-1)/2) × a
```



Exemplo 3

Solução Divisão e Conquista – Exponenciação



```
package paa;

public class DivConquista_3 {

    public static void main(String[] args) {

        int x = 2;
        int n = 5;

        System.out.println("Potencia de " + x + " elevado a " +
                           n + " : " + pot(x,n));

    }
}
```



Exemplo 3

Solução Divisão e Conquista – Exponenciação



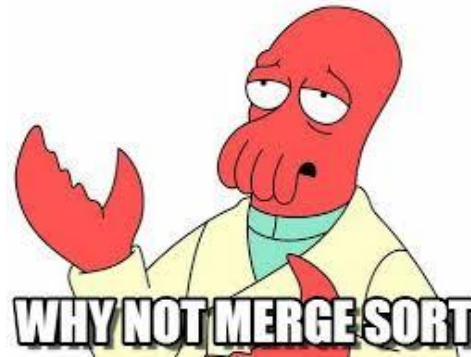
```
public static Long pot (int x , int n) {  
  
    if (n == 0) return 1L;  
  
    if (n%2 == 0)  
  
        return (pot (x, n/2 ) * pot(x,n/2) );  
  
    else  
  
        return (pot (x, (n-1)/2 )  * pot(x, (n-1)/2) * x ) ;  
    }  
}
```

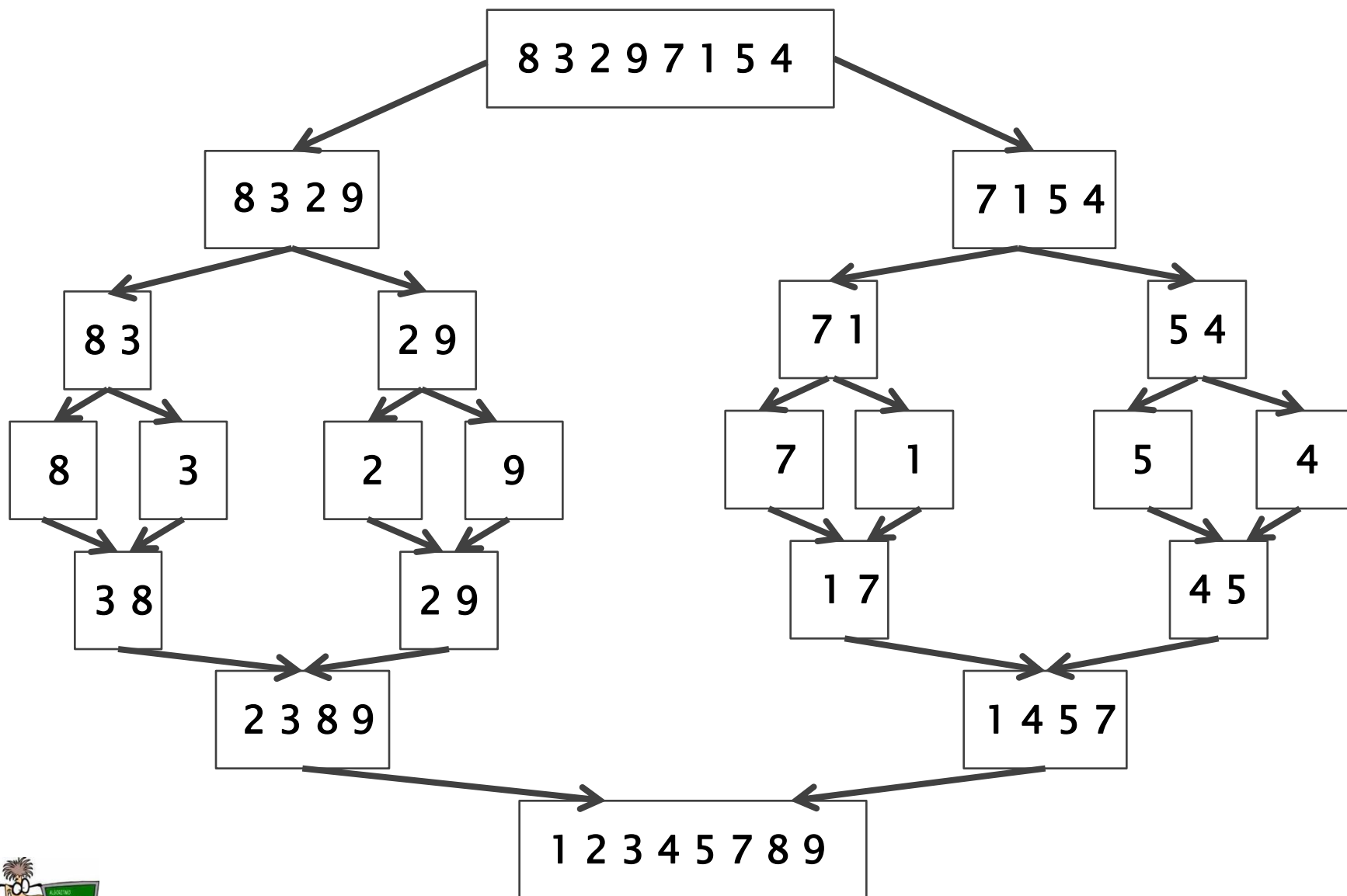




Mergesort

- ✓ É um perfeito exemplo de uma aplicação da técnica Divide-and-Conquer.
- ✓ O algoritmo classifica um dado array $A[0..n-1]$ dividindo-o em duas metades $A[0 \dots \lfloor n/2 \rfloor - 1]$ e $A[\lfloor n/2 \rfloor \dots n-1]$, sorteando cada uma delas de forma recursiva, e em seguida efetua um merge das partes sorteadas.





Mergesort – Implementação

```
package uscs;
```

```
public class MergeSort {
```

```
    public static int[] vetor = {8,3,2,9,7,1,5,4};
```

```
    static int n = vetor.length;
```

```
    public static void main(String[] args) {
```

```
        Sort(0,n-1);
```

```
        for (int i=0;i<n;i++)
```

```
            System.out.println("vetor[" + i + "] = " + vetor[i]);
```

```
    }
```





Mergesort – Implementação

```
public static void Sort(int inicio, int fim) {  
  
    if (inicio < fim) {  
  
        int meio = (inicio + fim) / 2;  
  
        Sort(inicio, meio);  
        Sort(meio + 1, fim);  
        Merge(inicio, meio, fim);  
    }  
}
```





Mergesort – Implementação

```
public static void Merge(int inicio, int meio, int fim) {  
  
    int n = fim - inicio + 1;  
  
    int[] temp = new int[n];  
    int tamanho = temp.length;  
  
    for (int posicao = 0; posicao < tamanho; posicao++) {  
        temp[posicao] = vetor[inicio + posicao];  
    }  
}
```



Mergesort – Implementação

```
int i = 0;
int j = meio - inicio + 1;

for (int posicao = 0; posicao < tamanho; posicao++) {

    if (j <= n- 1 )
        if ( i <= meio - inicio)
            if (temp[i] < temp[j] )
                vetor[inicio + posicao] = temp[i++];
            else vetor[inicio + posicao] = temp[j++];
        else vetor[inicio + posicao] = temp[j++];
        else vetor[inicio + posicao] = temp[i++];
    }

}
```





Mergesort – Eficiência

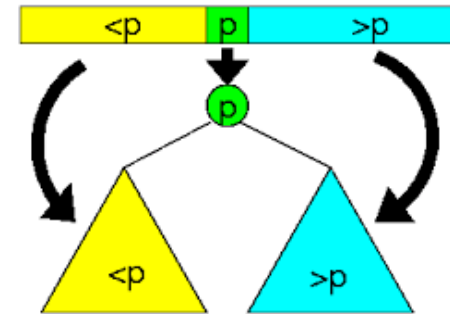
- ◆ Prova-se que a ordem de complexidade do algoritmo **MergeSort** é:

$$O(n) = n * \log n (n)$$





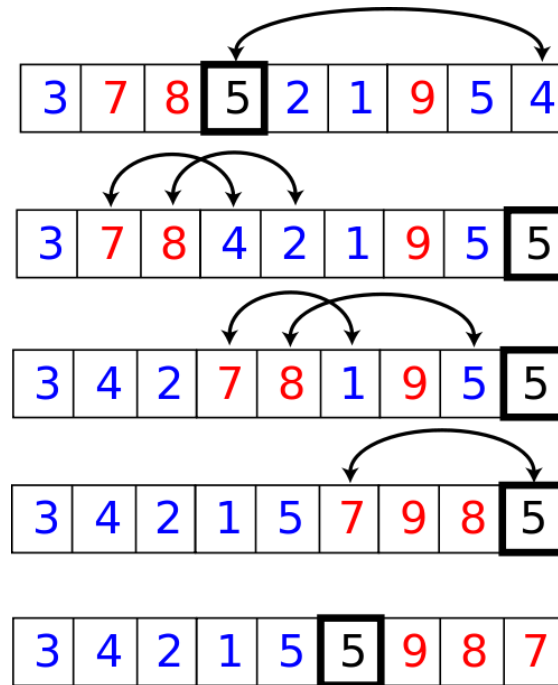
Quicksort





Quicksort

- ◆ É um algoritmo com boa eficiência;
- ◆ Emprega a estratégia **Divisão e Conquista**;
- ◆ Inicialmente divide uma lista em duas menores sub-listas, e recursivamente ordena estas sub-listas;





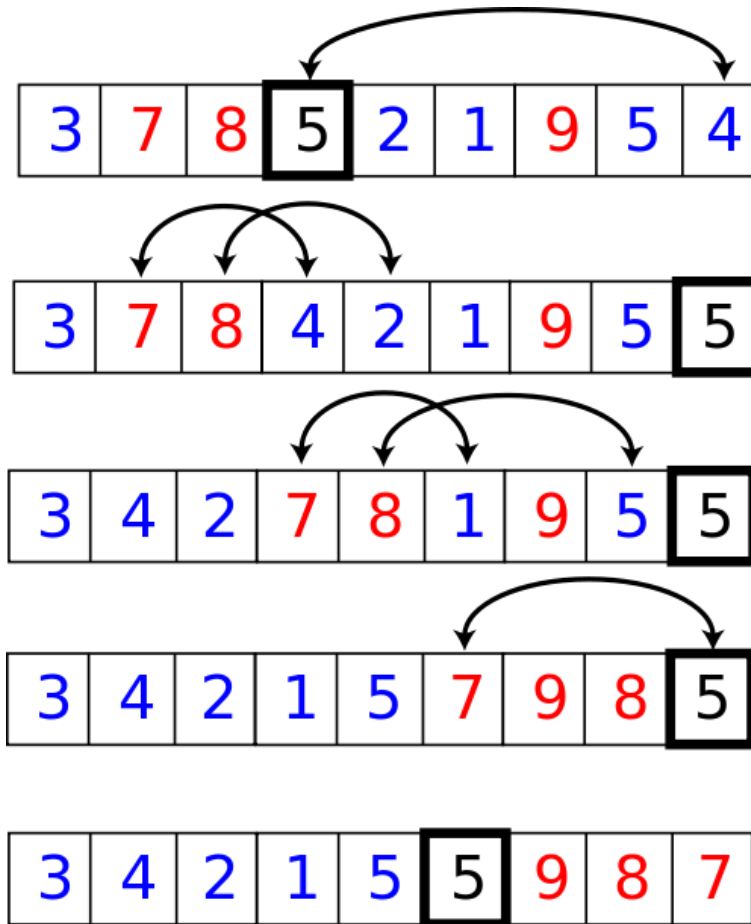
Quicksort – Algoritmo

- ◆ Escolha um elemento da lista, chamado **Pivot**;
- ◆ Geralmente o **Pivot** pode estar no meio da lista;
- ◆ Reordene a lista de modo que todos os elementos menores que o **Pivot** estejam antes dele e todos os elementos maiores que o **Pivot** estejam após ele;
- ◆ Após este particionamento, o **Pivot** está na sua posição final;
- ◆ Este procedimento inicial é chamado Operação de Partição;
- ◆ Recursivamente, aplique os passos acima para as sub-lista com elementos menores e separadamente para a sub-lista com maiores elementos.





Quicksort – Algoritmo



Quicksort – Algoritmo

```
package uscs;

import java.util.Arrays;

public class QuickSort {

    public static int[] lista = {24,2,45,20,56,75,2,56,99,53,12};

    public static void main(String[] args) {

        int n = lista.length;

        Quicksort(0 , n-1);

        System.out.println(Arrays.toString(lista));

    }
```



Quicksort – Algoritmo

```
public static void Quicksort(int inicio, int fim) {  
  
    int i = inicio;  
    int j = fim;  
  
    int pivot = lista[inicio+(fim-inicio)/2];  
  
    while (i <= j) {  
        while (lista[i] < pivot)  
            i++;  
  
        while (lista[j] > pivot)  
            j--;  
  
        if (i <= j) {  
            troca(i, j);  
            i++;  
            j--;  
        }  
    }  
}
```





Quicksort – Algoritmo

```
        if (inicio < j)
            Quicksort(inicio, j);
    if (i < fim)
        Quicksort(i, fim);
}

public static void troca(int i, int j) {
    int temp = lista[i];
    lista[i] = lista[j];
    lista[j] = temp;
}
}
```





Quicksort- Eficiência

◆ Prova-se que a ordem de complexidade do algoritmo **QuickSort** é:

◆ $O(n * \log n)$ (melhor caso)

◆ $O(n^2)$ (no pior caso)





Programação Dinâmica

- ◆ A técnica foi inventada pelo matemático Richard Bellman, na década de 50, como um método geral de otimização de processos de decisão;
- ◆ Assim, a palavra “**programação**” nesta técnica se refere à planejamento ao invés de programação de computadores;
- ◆ Recentemente a técnica foi incorporada na Computação.





Programação Dinâmica – Introdução

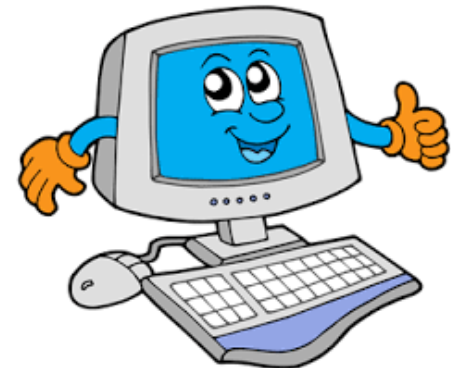
- ◆ Resolve problemas por meio da combinação de subproblemas;
- ◆ A técnica é aplicada quando os subproblemas não são independentes, isto é, quando os subproblemas compartilham subproblemas;
- ◆ Resolve cada subproblema somente uma vez e grava o resultado em uma tabela, evitando-se assim o trabalho de se recalcular a resposta toda vez que o subproblema é encontrado.





Programação Dinâmica – Introdução

- ◆ Em geral, a programação dinâmica é aplicada em problemas de otimização.
- ◆ Problemas de Otimização:
 - ▶ Muitas soluções possíveis
 - ▶ Cada solução tem um valor;
 - ▶ Deseja-se encontrar uma solução com um valor ótimo (min ou máx).





Estrutura de um problema com Programação Dinâmica

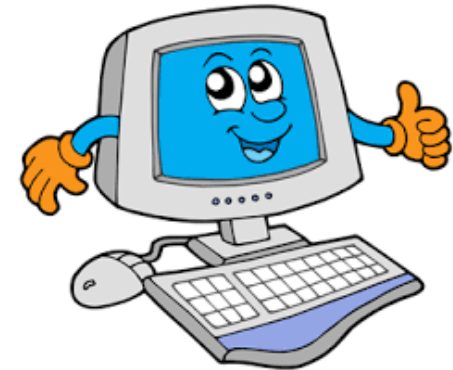
- ◆ Geralmente possuem estruturas recursivas;
- ◆ Sub-problemas devem ser resolvidos para que a solução final possa ser alcançada.





Porque não Recursão ?

- ◆ Se a estrutura do problema é recursiva, por que um algoritmo puramente recursivo não seria uma boa solução ?
- ◆ Algoritmos puramente recursivos trabalharão mais que o necessário resolvendo os sub-problemas comuns mais de uma vez [repetição de cálculos].





Um exemplo simples: **Fibonacci**

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .





Um exemplo simples: **Fibonacci**

- ◆ Os números de Fibonacci são definidos pela função:

$$F(0) = 0, n = 0$$

$$F(1) = 1, n = 1$$

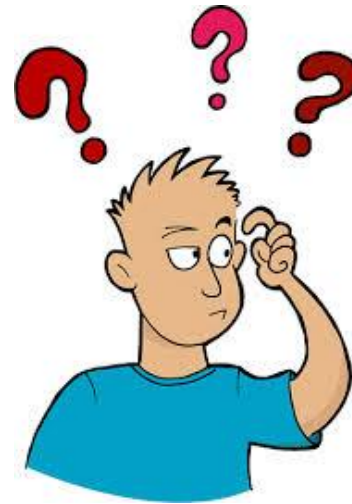
$$F(n) = F(n-1) + F(n-2), n > 1$$

- ◆ Definidos em termos de uma função recursiva.





Como implementar a Série de Fibonacci ?





Código recursivo para os números de Fibonacci

```
public static int Fib(int n) {  
    if (n == 0 || n == 1)  
        return n;  
    else  
        return Fib(n-1) + Fib(n-2);  
    }  
}
```





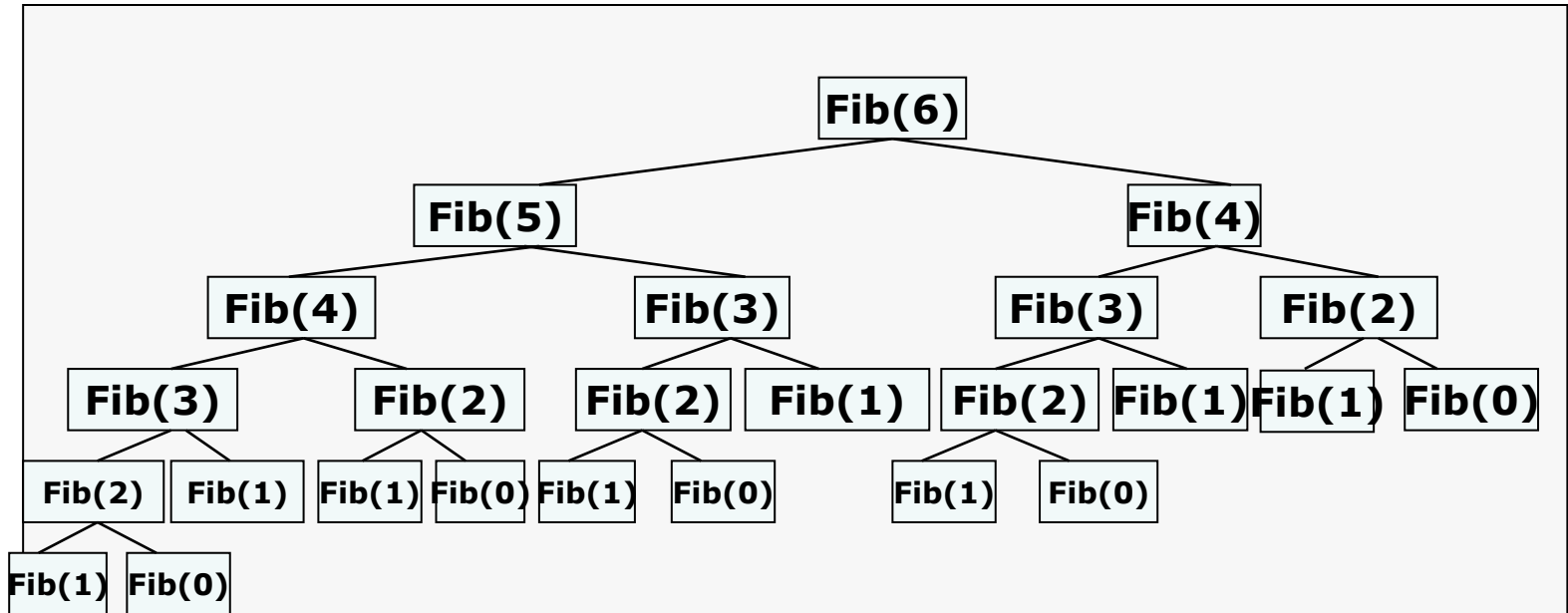
```
public static int Fib(int n) {  
    if (n == 0 || n == 1)  
        return n;  
    else  
        return Fib(n-1) + Fib(n-2);  
    }  
}
```

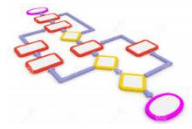
Quais os pontos negativos dessa implementação ?



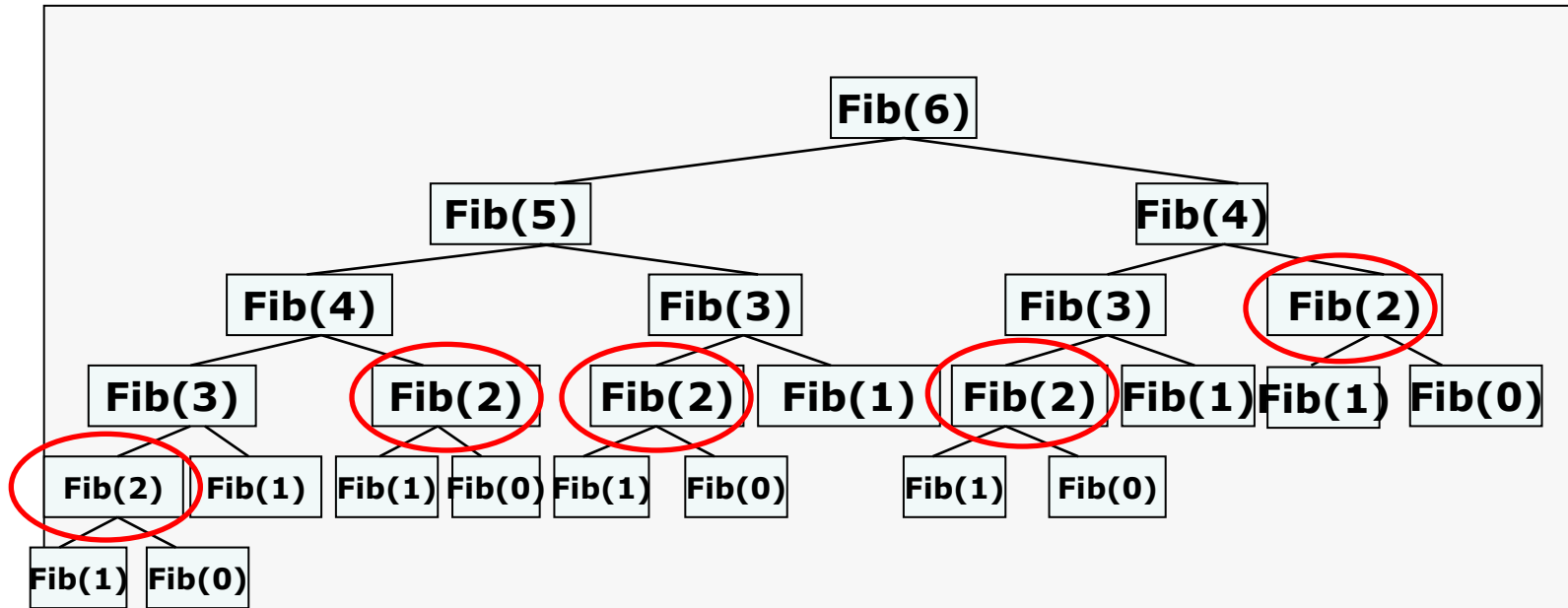


Chamadas recursivas para o cálculo de Fib(6)





Observe que há chamadas redundantes de funções !!!



Quantas vezes Fib(2) é chamada ?





Portanto...

- ✦ Os cálculos redundantes levam o algoritmo a ter maior complexidade computacional;
- ✦ Um algoritmo mais “otimizado” poderia ser feito com **programação dinâmica**;
- ✦ A ideia do algoritmo com programação dinâmica seria guardar numa tabela os sub-resultados;
- ✦ Melhor ainda: Guardaria apenas os dois últimos resultados (afinal só é preciso disso para saber o próximo resultado da sequência)



Com programação dinâmica . . .

```
package uscs;
public class Prog_Dinamica {

    public static void main(String[] args) {
        Integer n = 10;

        System.out.println("Fib de " + n + " = " + Fib(n) );
    }

    public static Integer  Fib(Integer n) {

        Integer u=1,p=0,f=0;
        if ( n == 0 || n == 1) return n;

        for (int i = 2; i <= n+1 ; i++) {
            f = u + p;
            p = u;
            u = f;
        }
        return f;
    }
}
```





Algoritmos Gulosos





Algoritmos Gulosos

- ⊕ Realiza a escolha que parece ser a **melhor** no momento na esperança de que a mesma acarrete em uma solução ou prevenção de futuros problemas a nível global.
- ⊕ É **simples** e de **fácil implementação**;
- ⊕ É **míope**: ele toma decisões com base nas informações disponíveis na iteração corrente.





Algoritmos Gulosos

- ⊕ Jamais se arrepende de uma decisão, as **escolhas** realizadas são **definitivas**;
- ⊕ Não leva em consideração as consequências de suas decisões;
- ⊕ Podem fazer cálculos repetitivos;
- ⊕ Nem sempre produz a melhor solução (depende da quantidade de informação fornecida);
- ⊕ Quanto mais informações, maior a chance de produzir uma solução melhor.



AGENDA
 Introduction to Bioinformatics
 1.1.1. Course - overview
 1.1.2. Course - overview
 1.1.3. Course - overview
 1.1.4. Course - overview
 1.1.5. Course - overview



Algoritmos Gulosos – Exemplo

```
public String calculaNotas(final double troco, String resultado) {  
  
    int nota[] = { 100, 50, 20, 10, 5, 2, 1 };  
  
    int valor;  
    int ct;  
  
    int contadorNota = 0;  
  
    valor = (int) troco;  
    while (valor != 0) {  
        ct = valor / nota[contadorNota]; // calculando a qtde de notas  
        if (ct != 0) {  
            resultado = resultado + (ct + " nota(s) de R$" +  
                                    nota[contadorNota] + "\n");  
            valor = valor % nota[contadorNota]; // sobra  
        }  
        contadorNota++; // próxima nota  
    }  
    return resultado;  
}
```





Algoritmos Gulosos – Exemplo

```
package paa;

public class TesteTrocoGuloso {

    public static void main(String[] args) {

        TrocoGuloso t = new TrocoGuloso();

        double conta = 559.25;

        double pago = 710.0;

        System.out.println(t.calculaTroco(conta, pago)) ;

    }

}
```

