# Algoritmos e Estrutura de Dados II (AE23CP-3CP)

## Aula #02 - Complexidade de Algoritmos - Parte I

Profª Luciene de Oliveira Marin
lucienemarin@utfpr.edu.br

# Complexidade de Algoritmos - Parte I

## Por que estudar a complexidade dos algoritmos?

Por estamos interessados em algoritmos "eficientes".

Programa $\times$ Algoritmo:

| Programa | Algoritmo |
|---|---|
| Linguagem concreta (C, Java, PHP) | Linguagem Abstrata (pseudo-código) |
| Dependente de SO | Independente de SO |
| Dependente de compilador | Independe de compilador |
| Dependente de máquina | Independe de máquina |
| **Avaliação de tempo real (empírica)** | **Avaliação por estimativa (assintótica)** |

**Por que estudar a complexidade dos algoritmos?**

Por estamos interessados em algoritmos "eficientes".

Programa $\times$ Algoritmo:

| **Programa** | **Algoritmo** |
|---|---|
| Linguagem concreta (C, Java, PHP) | Linguagem Abstrata (pseudo-código) |
| Dependente de SO | Independente de SO |
| Dependente de compilador | Independe de compilador |
| Dependente de máquina | Independe de máquina |
| **Avaliação de tempo real (empírica)** | **Avaliação por estimativa (assintótica)** |

# Algoritmo

## O que é um algoritmo (computacional?)

É uma ferramenta para resolver um **problema computacional**.
Ou seja, é um procedimento computacional bem definido que:

- recebe um conjunto de valores como **entrada**,
- produz um conjunto de valores como **saída**,
- atravé de uma sequência de passos em um **modelo computacional**

# Exemplos de problemas (1/2)

Teste de primalidade:

**Problema:** determinar se um dado número é primo.

Exemplo:

Entrada: 9411461

Saída: É primo.

Exemplo:

Entrada: 8411461

Saída: Não é primo.

# Exemplos de problemas (2/2)

Ordenação:

Definição: um vetor $A[1 \ldots n]$ é crescente se $A[1] \leq \ldots \leq A[n]$.

Problema: rearranjar um vetor $A[1 \ldots n]$ de modo que fique crescente.

Entrada:

| 1 | | | | | | | | | | $n$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 33 | 55 | 33 | 44 | 33 | 22 | 11 | 99 | 22 | 55 | 77 |

Saída:

| 1 | | | | | | | | | | $n$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 11 | 22 | 22 | 33 | 33 | 33 | 44 | 55 | 55 | 77 | 99 |

# Instância de um problema

Uma instância de um problema é um conjunto de valores que serve de entrada para esse.

Exemplo:
Os números 9411461 e 8411461 são instâncias do problema de primalidade.

Exemplo:
O vetor

| 1 | | | | | | | | | | $n$ |
|---|---|---|---|---|---|---|---|---|---|---|
| 33 | 55 | 33 | 44 | 33 | 22 | 11 | 99 | 22 | 55 | 77 |

é uma instância do problema de ordenação.

# A importância dos algoritmos para a computação

**Onde se encontram as aplicações para o uso/desenvolvimento de algoritmos "eficientes"?**

- projeto de genoma de seres vivos
- rede mundial de computadores
- comércio eletrônico
- planejamento de produção de indústrias
- logística de distribuição
- games e filmes
- ...

# Objetivo de estudar complexidade de algoritmos

## Por que analisar a complexidade dos algoritmos?

Para projetar algoritmos, é preferível:

- Se preocupar com o projeto de um algoritmo **eficiente**, desde sua concepção, do que
- Desenvolver um algoritmo e só depois analisar sua complexidade para verificar sua eficiência.

# Dificuldade intrínseca de problemas (1/4)

- Infelizmente, existem certos problemas para os quais **não se conhece** algoritmos eficientes capazes de resolvê-los. Exemplos são os problemas NP-completos.

  Curiosamente, **não foi provado** que tais algoritmos não existem! **Interprete isso como um desafio da inteligência humana.**

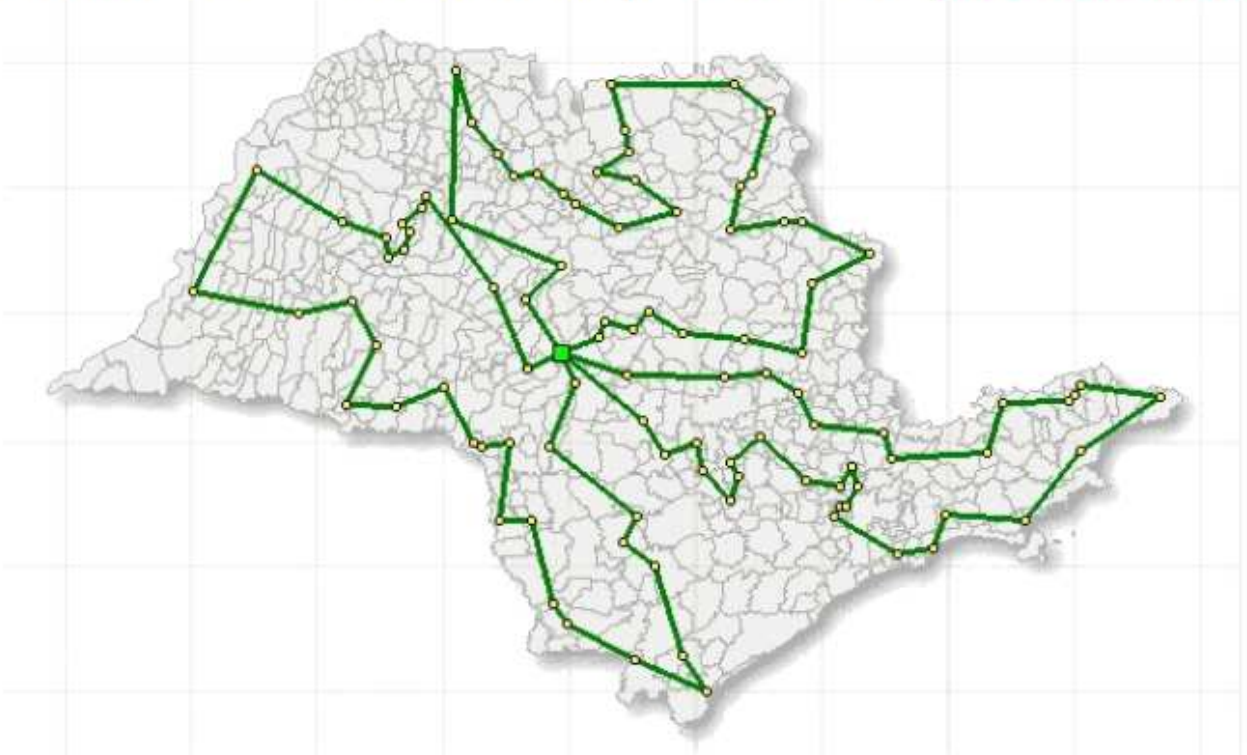
- Esses problemas tem a característica notável de que se um deles admitir um algoritmo "eficiente" então todos admitem algoritmos "eficientes".

**Por que devo me preocupar com problemas NP-completos?**

Problemas dessa classe surgem em inúmeras situações práticas, como problemas NP-difíceis.

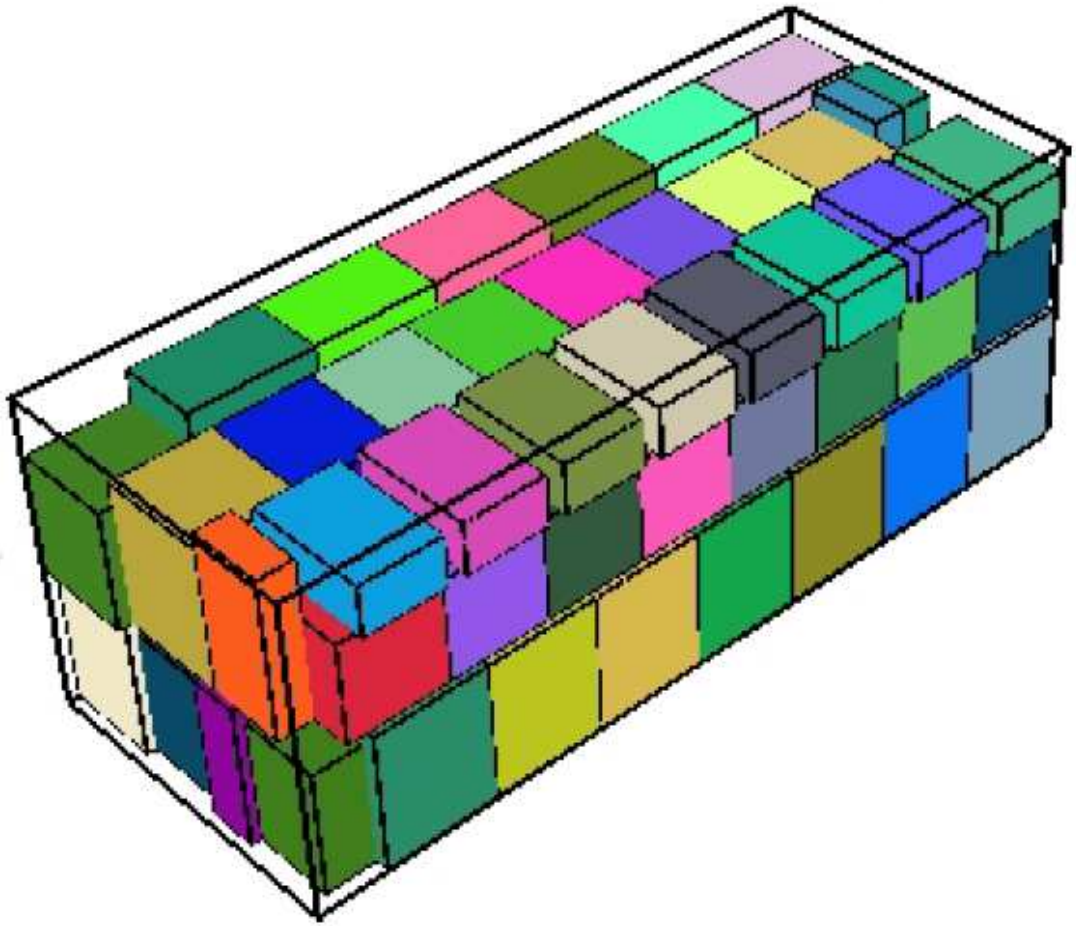
# Dificuldade intrínseca de problemas (2/4)

Exemplo de problema $\mathcal{NP}$-difícil: calcular as rotas dos caminhões de entrega de uma distribuidora de bebidas em São Paulo, minimizando a distância percorrida. (vehicle routing)



# Dificuldade intrínseca de problemas (3/4)

Exemplo de problema $\mathcal{NP}$-difícil: calcular o número mínimo de *containers* para transportar um conjunto de caixas com produtos. (bin packing 3D)



# Dificuldade intrínseca de problemas (4/4)

Exemplo de problema $\mathcal{NP}$-difícil: calcular a localização e o número mínimo de antenas de celulares para garantir a cobertura de uma certa região geográfica. (facility location)

e muito mais...

É importante saber indentificar quando estamos lidando com um problema $\mathcal{NP}$-difícil!

# Algoritmos e tecnologia

**O Éden:**

os computadores têm velocidade de processamento e memória infinita.
Neste caso, qualquer algoritmo é igualmente bom e esta disciplina é inútil!

**O mundo real:**

há computadores com velocidade de processamento na ordem de bilhões por segundo e trilhões de bytes em memória.
Mas ainda sim temos uma limitação na velocidade de processamento e memória dos computadores.

**Conclusão:**

Neste caso faz **muita** diferença ter um **bom** algoritmo.

# Complexidade de algoritmos - eficiência

**Para um dado problema considere dois algoritmos que o resolvem:**

- Seja $n$ um parâmetro que caracteriza o tamanho da entrada do algoritmo.

  Por exemplo, ordenar $n$ números ou multiplicar duas matrizes $n \times n$ (cada uma com $n^2$ elementos).

- **Como comparar os dois algoritmos para escolher o melhor?**

Exemplo: **ordenação de um vetor de $n$ elementos**

- Suponha que os computadores $A$ e $B$ executam $1G$ e $10M$ instruções por segundo, respectivamente. Ou seja, $A$ é **100 vezes mais rápido** que $B$.
- Algoritmo 1: implementado em $A$ por um excelente programador em linguagem de máquina (ultra-rápida). Executa $2n^2$ instruções.
- Algoritmo 2: implementado na máquina $B$ por um programador mediano em linguagem de alto nível dispondo de um compilador "meia-boca". Executa $50n \log n$ instruções.

# Complexidade de algoritmos - eficiência

- O que acontece quando ordenamos um vetor de um milhão de elementos? **Qual algoritmo é mais rápido?**
- Algoritmo 1 na máquina *A*:
$$\frac{2.(10^6)^2 \text{ instruções}}{10^9 \text{ instruções/segundo}} \approx 2000 \text{ segundos}$$
- Algoritmo 2 na máquina *B*:
$$\frac{50.(10^6 \log 10^6) \text{ instruções}}{10^7 \text{ instruções/segundo}} \approx 100 \text{ segundos}$$
- Ou seja, *B* foi **VINTE VEZES** mais rápido do que *A*!
- Se o vetor tiver 10 milhões de elementos, esta razão será de 2.3 dias para 20 minutos!

E se tivermos os tais problemas $\mathcal{NP}$-difíceis ?

| $f(n)$ | $n = 20$ | $n = 40$ | $n = 60$ | $n = 80$ | $n = 100$ |
|---|---|---|---|---|---|
| $n$ | $2{,}0\times10^{-11}$seg | $4{,}0\times10^{-11}$seg | $6{,}0\times10^{-11}$seg | $8{,}0\times10^{-11}$seg | $1{,}0\times10^{-10}$seg |
| $n^2$ | $4{,}0\times10^{-10}$seg | $1{,}6\times10^{-9}$seg | $3{,}6\times10^{-9}$seg | $6{,}4\times10^{-9}$seg | $1{,}0\times10^{-8}$seg |
| $n^3$ | $8{,}0\times10^{-9}$seg | $6{,}4\times10^{-8}$seg | $2{,}2\times10^{-7}$seg | $5{,}1\times10^{-7}$seg | $1{,}0\times10^{-6}$seg |
| $n^5$ | $2{,}2\times10^{-6}$seg | $1{,}0\times10^{-4}$seg | $7{,}8\times10^{-4}$seg | $3{,}3\times10^{-3}$seg | $1{,}0\times10^{-2}$seg |
| $2^n$ | $1{,}0\times10^{-6}$seg | 1,0seg | 13,3dias | $1{,}3\times10^{5}$séc | $1{,}4\times10^{11}$séc |
| $3^n$ | $3{,}4\times10^{-3}$seg | 140,7dias | $1{,}3\times10^{7}$séc | $1{,}7\times10^{19}$séc | $5{,}9\times10^{28}$séc |

Supondo um computador com velocidade de 1 Terahertz (mil vezes mais rápido que um computador de 1 Gigahertz).

E se usarmos um super-computador para resolver os problemas $\mathcal{NP}$-difíceis ?

| $f(n)$ | Computador atual | 100×mais rápido | 1000×mais rápido |
|---|---|---|---|
| $n$ | $N_1$ | $100N_1$ | $1000N_1$ |
| $n^2$ | $N_2$ | $10N_2$ | $31.6N_2$ |
| $n^3$ | $N_3$ | $4.64N_3$ | $10N_3$ |
| $n^5$ | $N_4$ | $2.5N_4$ | $3.98N_4$ |
| $2^n$ | $N_5$ | $N_5 + 6.64$ | $N_5 + 9.97$ |
| $3^n$ | $N_6$ | $N_6 + 4.19$ | $N_6 + 6.29$ |

Fixando o tempo de execução: Não iremos resolver problemas muito maiores.

**Exemplo:** Cálculo do determinante de uma matriz $n \times n$:

- A tabela abaixo mostra o desempenho de dois algoritmos, considerando-se os tempos de operações de um computador real:

| n | Método de Cramer | Método de Gauss |
|---|---|---|
| 2 | 22 µs | 50 µs |
| 3 | 102 µs | 159 µs |
| 4 | 456 µs | 353 µs |
| 5 | 2,35 ms | 666 µs |
| 10 | 1,19 min | 4,95 ms |
| 20 | 15 225 séculos | 38,63 ms |
| 40 | $5 . 10^{33}$ séculos | 0,315 s |

Exemplo: Métodos de Ordenação. Qual implementar?

- Bolha
- Insertsort
- Shellsort
- Quicksort
- Mergesort
- Heapsort

Exemplo: Métodos de Busca. Qual utilizar? Por que?

- Busca Sequencial
- Busca Binária;
- Árvores de Pesquisa

# Algoritmos e tecnologia - Conclusões 1/2

- O uso de um algoritmo adequado pode levar a ganhos extraordinários de desempenho.
- Isso pode ser tão importante quanto o projeto de *hardware*.
- A melhora obtida pode ser tão significativa que não poderia ser obtida simplesmente com o avanço da tecnologia.
- As melhorias nos algoritmos produzem avanços em outras componentes básicas das aplicações (pense nos compiladores, buscadores na internet, etc).

## Complexidade de um algoritmo

Reflete o **esforço computacional** requerido para executá-lo.

## Esforço computacional

Mede a quantidade de trabalho, em termos de **tempo de execução** ou de **quantidade de memória** requerida.

- Complexidade de tempo
  - Exemplo: tempo de execução de um método de ordenação como bolha ou quicksort.
- Complexidade de espaço
  - Exemplo: algoritmos de grafos, data mining, bioinformática

# Complexidade de algoritmos

## Algoritmos

- são o cerne da computação
- um programa codifica um algoritmo a ser executado em um computador para resolver determinado problema.

## Projeto e Análise de Algoritmos

- É extremamente importante pois visa produzir soluções com o menor dispêncio possível de **tempo** e **memória**.

# Critérios de Complexidade

## Operação fundamental:

Operação escolhida para medir a quantidade de trabalho realizado por um algoritmo

- Às vezes, é necessária mais de uma operação fundamental e com pesos diferentes: **custo**

## Exemplo:

Para um **algoritmo de ordenação**, a operação fundamental é a **comparação** entre elementos.

# Critérios de Complexidade

## Tamanho da entrada:

Está associado ao tamanho das estruturas de dados do algoritmo:

- Quantidade de bits para representar um número;
- Quantidade de nós e arestas em um grafo;
- Exemplo:

## Exemplo:

Tamanho do vetor em um problema de ordenação ou pesquisa.

# Medida de complexidade e eficiência de algoritmos

- A complexidade de tempo (= eficiência) de um algoritmo é o número de instruções básicas que ele executa em função do tamanho da entrada.
- Adota-se uma "atitude pessimista" e faz-se uma análise de pior caso.
  Determina-se o tempo máximo necessário para resolver uma instância de um certo tamanho.
- Além disso, a análise concentra-se no comportamento do algoritmo para entradas de tamanho GRANDE = análise assintótica.

Como exemplo, considere o número de operações de cada um dos dois algoritmos que resolvem o mesmo problema, como função de $n$.

- Algoritmo 1: $f_1(n) = 2n^2 + 5n$ operações
- Algoritmo 2: $f_2(n) = 500n + 4000$ operações

Dependendo do valor de $n$, o Algoritmo 1 pode requerer mais ou menos operações que o Algoritmo 2.

(Compare as duas funções para $n = 10$ e $n = 100$.)

# Comportamento assintótico

- Algoritmo 1: $f_1(n) = 2n^2 + 5n$ operações
- Algoritmo 2: $f_2(n) = 500n + 4000$ operações

Um caso de particular interesse é quando $n$ tem valor muito grande ($n \to \infty$), denominado *comportamento assintótico*.

Os termos inferiores e as constantes multiplicativas contribuem pouco na comparação e podem ser descartados.

O importante é observar que $f_1(n)$ cresce com $n^2$ ao passo que $f_2(n)$ cresce com $n$. Um crescimento quadrático é considerado pior que um crescimento linear. Assim, vamos preferir o Algoritmo 2 ao Algoritmo 1.

# Medida de complexidade e eficiência de algoritmos

- Um algoritmo é chamado eficiente se a função que mede sua complexidade de tempo é limitada por um **polinômio** no tamanho da entrada.
  Por exemplo: $n$, $3n-7$, $4n^2$, $143n^2-4n+2$, $n^5$.
- Mas por que polinômios?
  Resposta padrão: (polinômios são funções bem "comportadas").

# Medida de complexidade e eficiência de algoritmos

## Medidas de complexidade:

- Melhor Caso
- Pior Caso
- Caso Médio

# Melhor Caso

- Pouca utilidade prática
- Fornece um cálculo muito otimista sobre o comportamento do algoritmo.
- Exemplo: busca sequencial
- Objetivo: encontrar o elemento 5 no vetor

| 5 | 150 | 31 | 10 | 17 | 18 | 45 | 98 | 101 | 200 |
|---|---|---|---|---|---|---|---|---|---|

- O elemento é encontrado após 1 execução da operação fundamental, ou seja, a melhor situação possível.

# Melhor Caso - exemplo: busca sequencial

```
início {início Pesquisa Sequencial – versão 2}
    tipo v = vetor[1..n] inteiro;
         v: VET;
    inteiro : I, K;
    lógico : ACHOU;
    leia(k);
    leia(VET);
    ACHOU ← falso; i ← 1;
    enquanto ((não ACHOU) e (i < n)) faça
        se (VET[i]  = K) então
            escreva(K, "Esta na posicao ", i);
            ACHOU ← verdadeiro;
        fim_se;
        i ← i +1;
    fim_enquanto;
    se não ACHOU então
        escreva ("A chave", K, "Nao está no vetor"); ;
    fim_se(VET);
fim
```

# Pior Caso

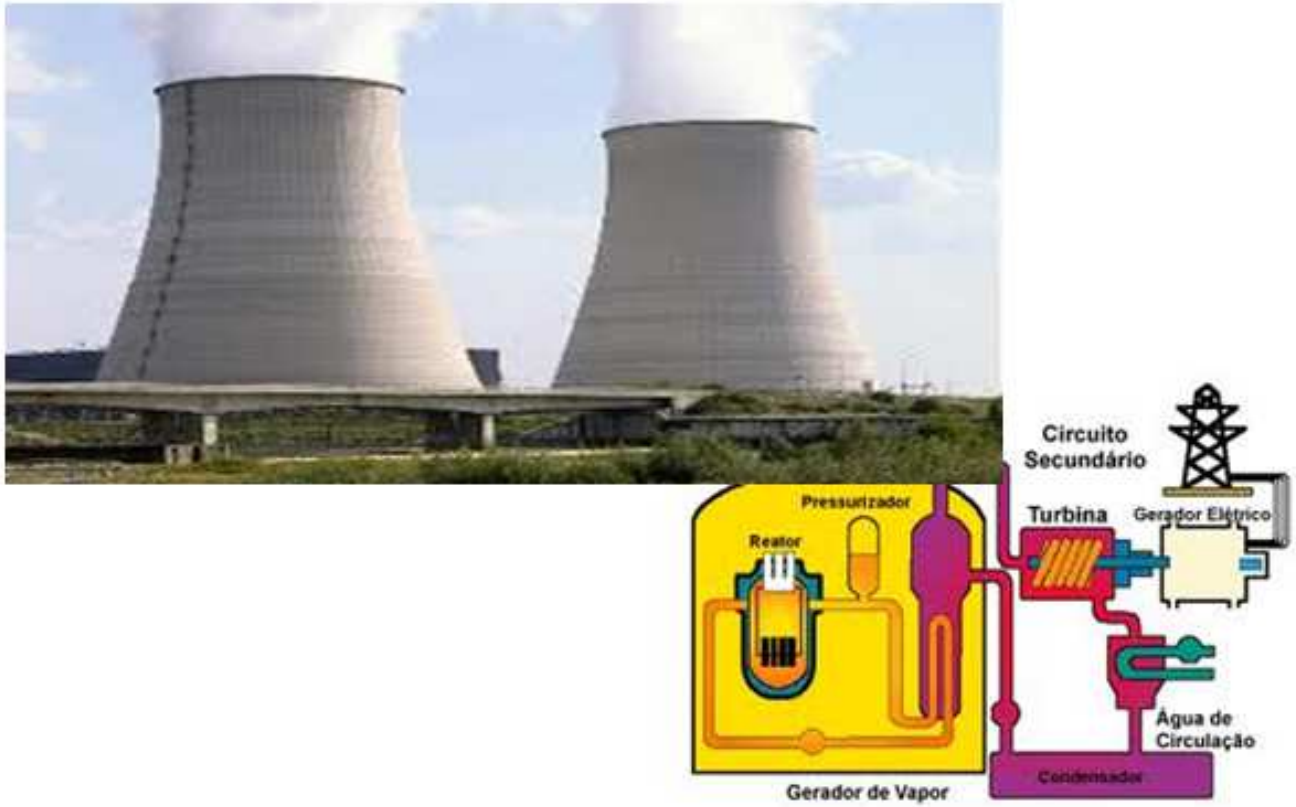
- É a complexidade pessimista, mede o esforço máximo necessário para resolver um problema de tamanho $n$.
- É fácil de calcular
- Exemplo: Encontrar o elemento 200 ou 1.000 no vetor

| 5 | 150 | 31 | 10 | 17 | 18 | 45 | 98 | 101 | 200 |
|---|---|---|---|---|---|---|---|---|---|

- É útil quando precisamos dar garantias sobre o desempenho do algoritmo.

# Pior Caso - Cenário 1: controlador de vôo

# Pior Caso - Cenário 2: controle de processo



# Caso Médio

- Calcula-se a complexidade de todas as entradas possíveis e extrai-se a sua média.
- Exemplo da pesquisa sequencial:
  1. complexidade do elemento estar na $1^a$ posição
  2. complexidade do elemento estar na $2^a$ posição...
  n. complexidade do elemento estar na $n^a$ posição.
- complexidade final é a média de todos os cálculos.
- **Muitas vezes é muito difícil de ser calculada.**

# Método de análise de complexidade proposto: (1/2)

**Vantagens:**

- O modelo é robusto pois permite **prever** o comportamento de um algoritmo para instâncias GRANDES
- O modelo permite **comparar** algoritmos que resolvem um mesmo problema.
- A análise é mais robusta em relação às evoluções tecnológicas.

# Método de análise de complexidade proposto: (2/2)

## Desvantagens:

- Fornece um limite de **complexidade** pessimista sempre considerando o **pior caso**.
- Em uma aplicação real, nem todas as instâncias ocorrem com a mesma frequência e é possível que as "instâncias ruins" ocorram raramente.
- Não fornece nenhuma informação sobre o comportamento do algoritmo no **caso médio**.
- A análise de **complexidade de algoritmos** no **caso médio** é complicada e depende do conhecimento da distribuição das instâncias.

# Lista de Exercícios

Lista de exercícios APS#02 - Métodos de ordenação e desempenho

- Vide moodle da disciplina.