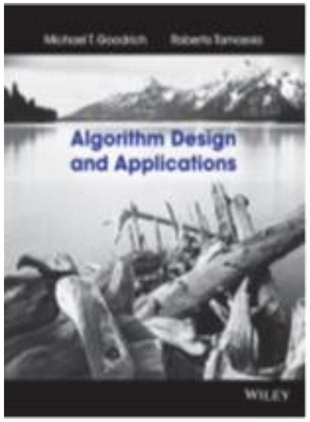
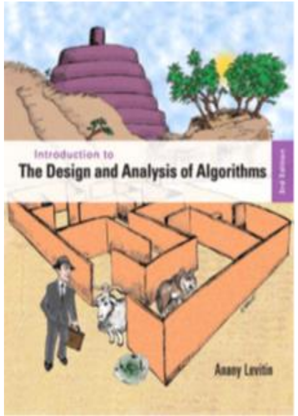
Unidade 3 - Função de Complexidade de Algoritmos e notação Big(O)

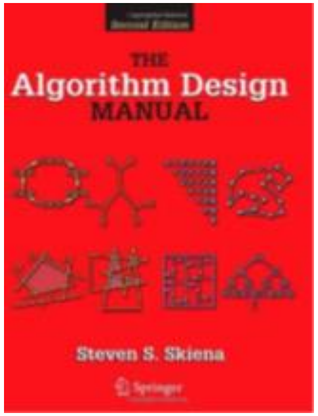
Prof. Aparecido V. de Freitas
Doutor em Engenharia
da Computação pela EPUSP
aparecidovfreitas@gmail.com

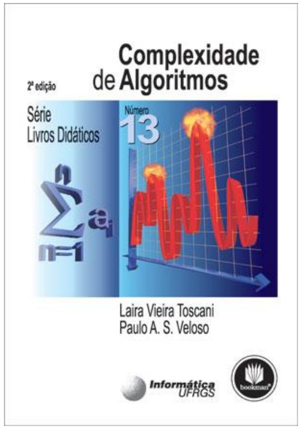
Bibliografia

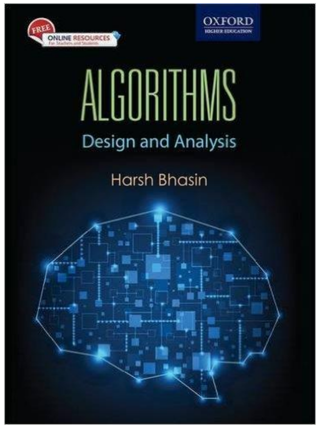
- Algorithm Design and Applications – Michael T. Goodrich, Roberto Tamassia, Wiley, 2015
- Introduction to the Design and Analysis of Algorithms – Anany Levitin, Pearson, 2012
- The Algorithm Design Manual – Steven S. Skiena, Springer, 2008
- Complexidade de Algoritmos – Série Livros Didáticos – UFRGS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











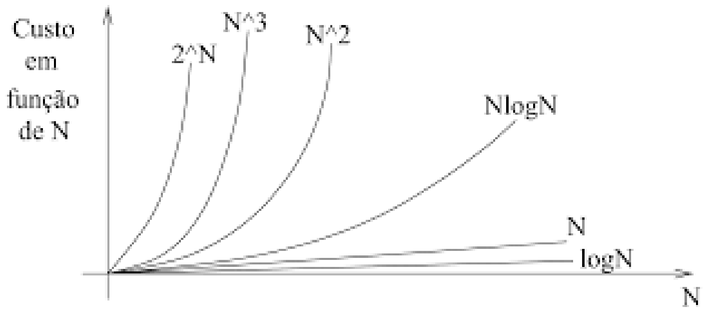
Função de Complexidade

- Para medir o custo de execução de um algoritmo é comum definir uma função de **custo** ou função de **complexidade** f.

- Função de complexidade de tempo: f(n) mede o tempo necessário para executar um algoritmo em um problema de tamanho n .

- Função de complexidade de espaço: f(n) mede a memória necessária para executar um algoritmo em um problema de tamanho n.

- A complexidade de tempo na realidade não representa tempo diretamente, mas o número de vezes que determinada operação considerada relevante é executada.



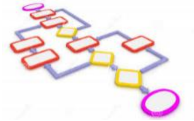
Exemplo: Maior elemento de um array de inteiros

Principal.java

```java
package br.com.qualitsys;

import java.util.Arrays;

public class Principal {

    public static void main(String[] args) {

        int n = 100;

        int[] array = new int[n];

        for (int i = 0; i <array.length; i++)
            array[i] = (int) (1000.0 * Math.random());

        System.out.println("Maximo valor do array: " + Max(array));

    }
```

Exemplo: Maior elemento de um array de inteiros

```java
public static int Max(int[] array) {
    System.out.println(Arrays.toString(array));
    int contador =0;
    int n = array.length;

    int Max = array[0];
    for (int i = 1; i<n ; i++) {
        if (array[i] >= Max)
            Max = array[i];
        contador++;
    }
    System.out.println("Contador:" + contador);
    return Max;
}
```



Exemplo: Maior elemento de um array de inteiros

Trace de Execução

```
Problems  @ Javadoc  Declaration  Console
<terminated> Principal (1) [Java Application] C:\Program Files\Java\jre1.8.0_181\bin\javaw.exe (Mar 6, 2019, 6:20:10 PM)
[587, 616, 841, 177, 162, 347, 412, 478, 657, 355, 485, 690, 675, 234, 310, 194, 814, 76
Contador:99
Maximo valor do array: 990
```

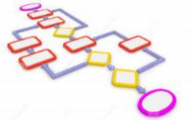
- ✓ O array tem **100** elementos e foram executadas **99** comparações
- ✓ Assim, se o array tiver **n** elementos, serão executadas **n-1** comparações . . .

Exemplo: Maior elemento de um array de inteiros

- Seja f um função de complexidade tal que f(n) corresponda ao número de comparações entre os elementos do array, considerando que o array tenha n elementos;

- f(**n**) = **n – 1**, para n > 0;

- Logo, **n – 1** comparações são necessárias.

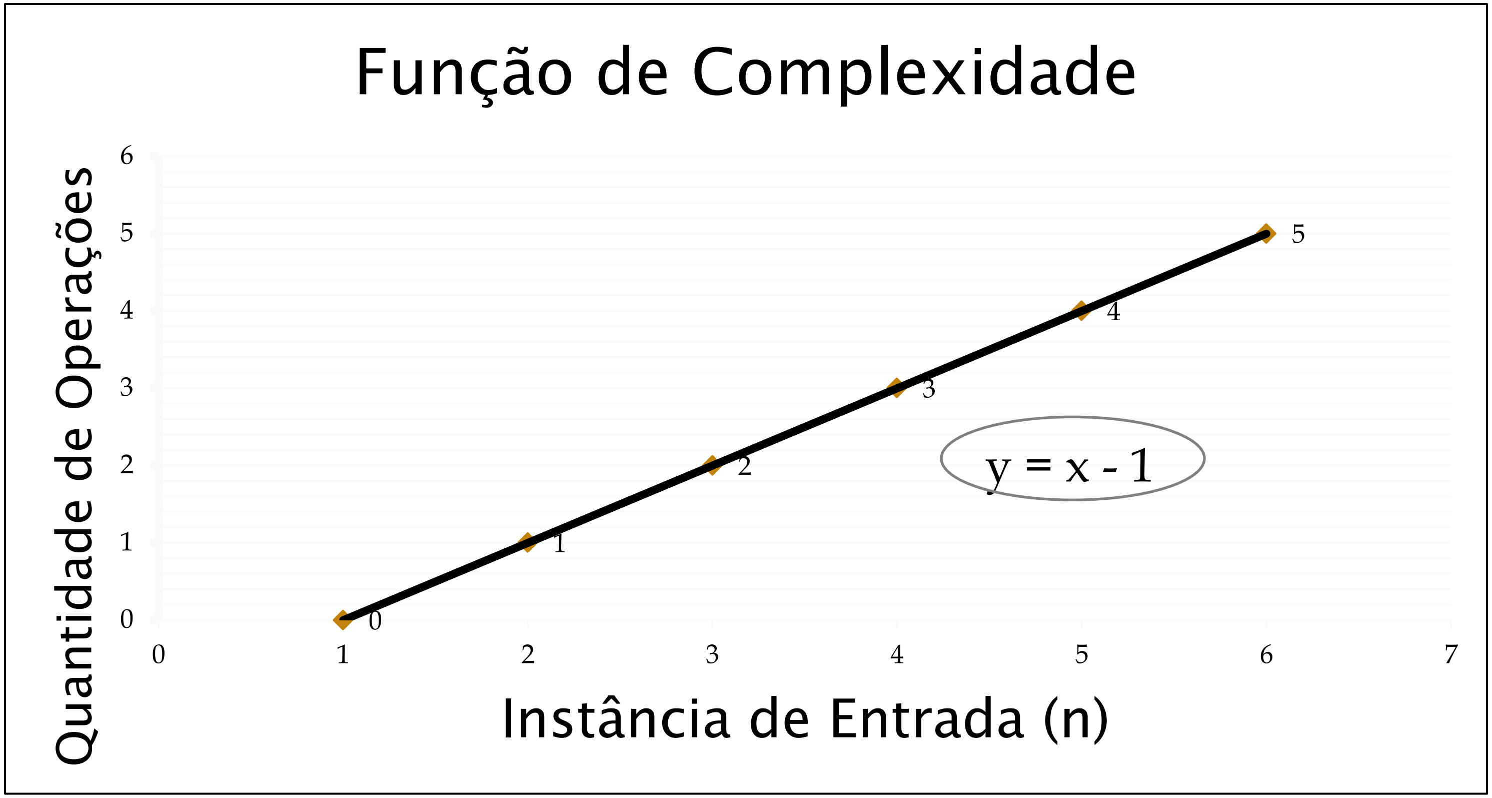


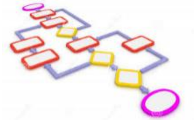
Exemplo: Maior elemento de um array de inteiros

Função de Complexidade

n	f(n) = n- 1
1	0
2	1
3	2
4	3
10	9
20	19

Exemplo: Maior elemento de um vetor de inteiros

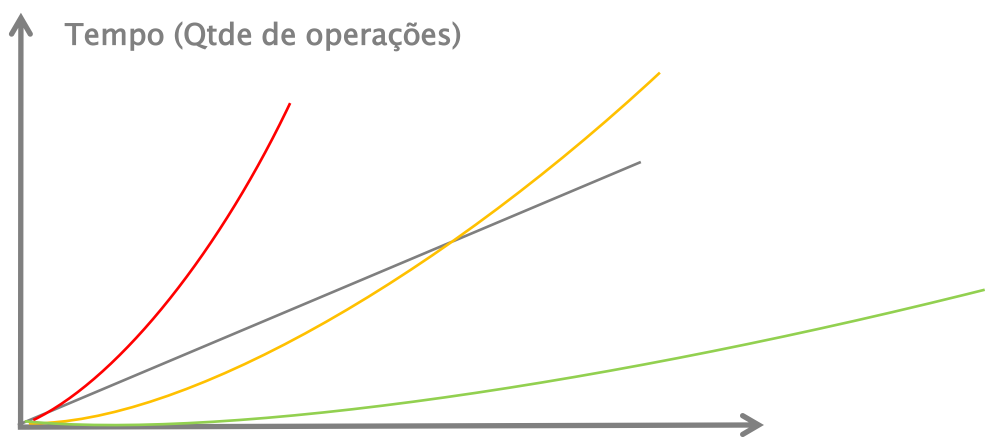




Tamanho da entrada de dados (Instância)

- A medida do custo de execução de um algoritmo depende principalmente do tamanho da entrada dos dados;

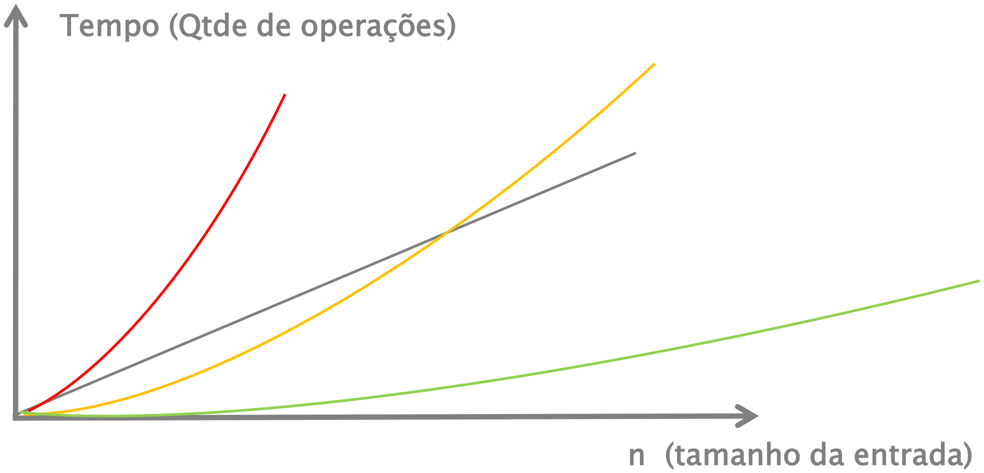
- É comum considerar o tempo de execução de um programa como uma função do tamanho da entrada.

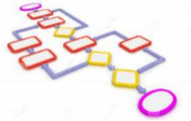


Tamanho da entrada de dados (Instância)

- A medida do custo de execução de um algoritmo depende principalmente do tamanho da entrada dos dados;

- É comum considerar o tempo de execução de um programa como uma função do tamanho da entrada.





Tamanho da entrada de dados

- No caso do método **max** do programa do exemplo, o custo é proporcional à entrada de dados submetida ao algoritmo;

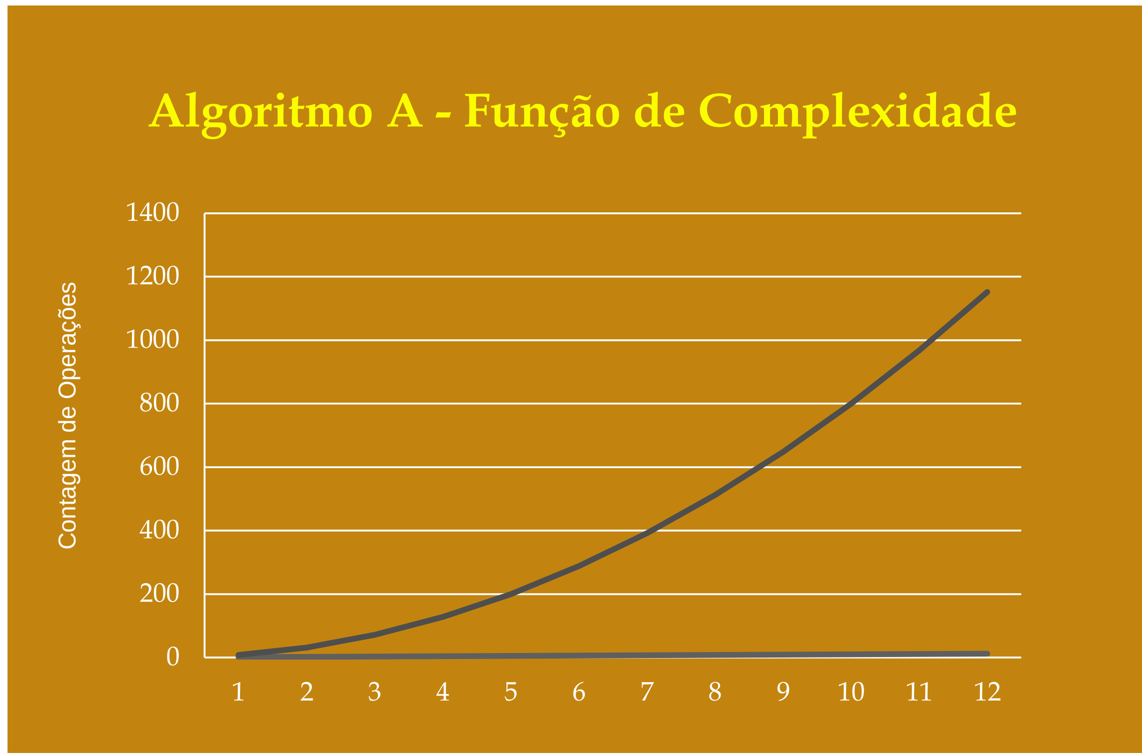
- Já para um algoritmo de ordenação isso não ocorre: se os dados de entrada já estiverem quase ordenados, então o algoritmo irá trabalhar menos.

Atividade 1

Considere dois algoritmos **A** e **B** com complexidades respectivamente iguais a $8n^2$ e n^3. Qual o maior valor de n, para o qual o algoritmo B é mais eficiente que o algoritmo A ?

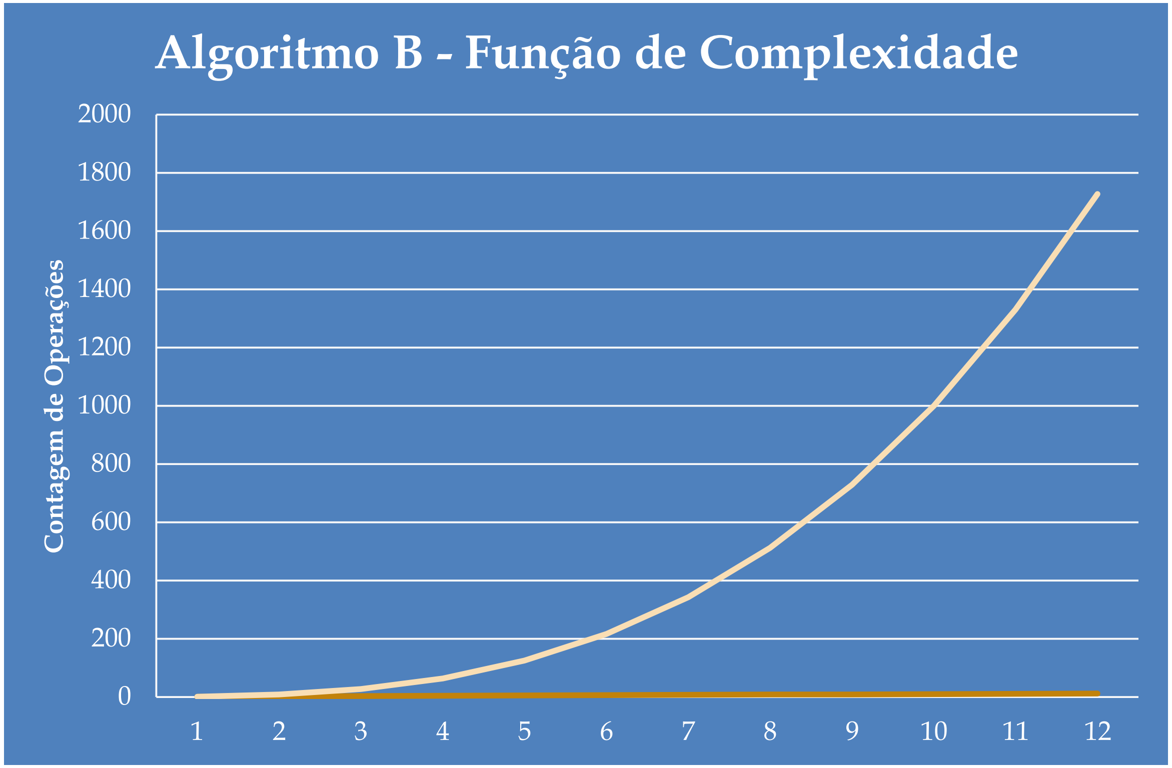
Função de Complexidade $8n^2$

n	$8n^2$
1	8
2	32
3	72
4	128
5	200
6	288
7	392
8	512
9	648
10	800
11	968
12	1152

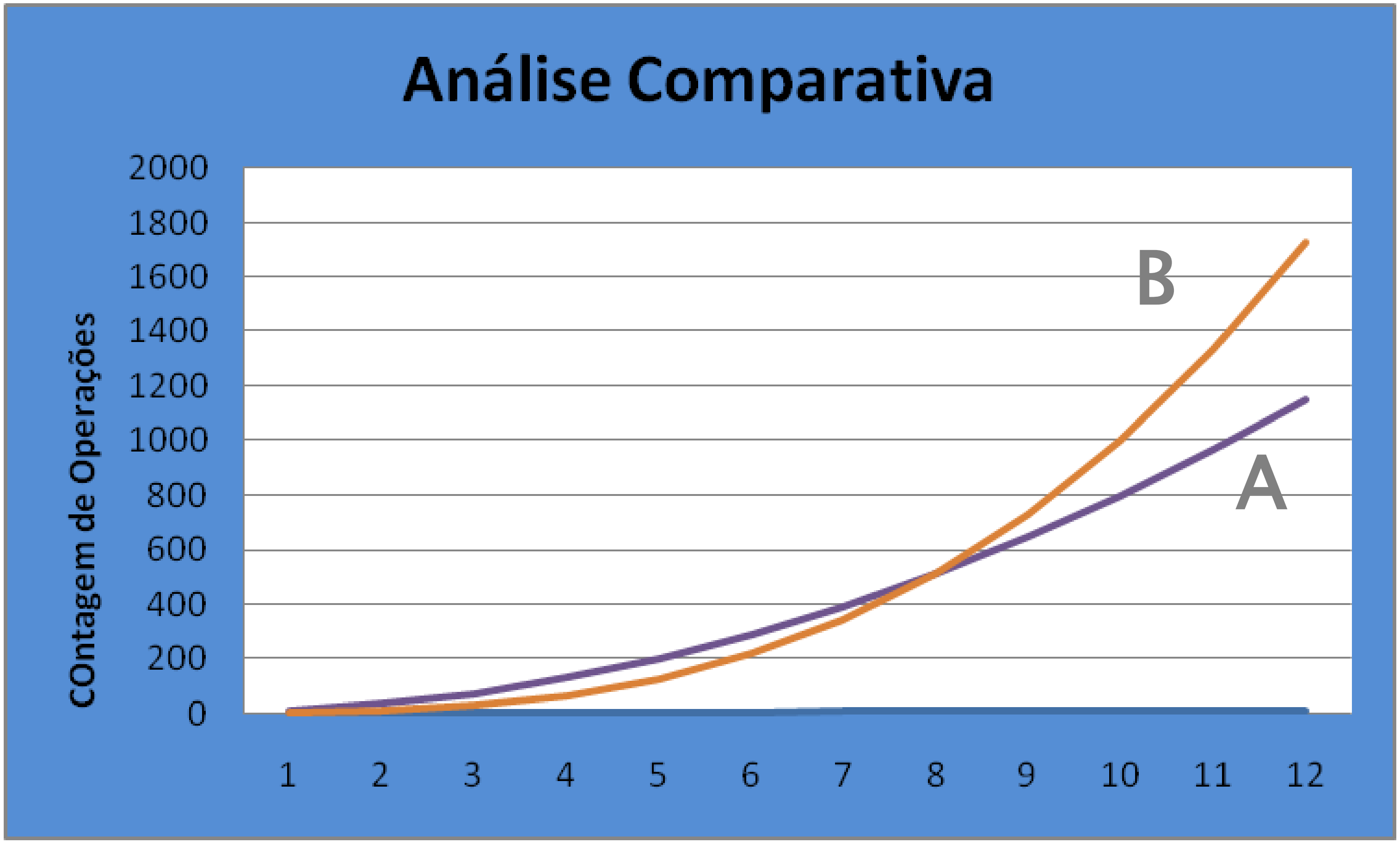


Função de Complexidade n^3

n	n^3
1	1
2	8
3	27
4	64
5	125
6	216
7	343
8	512
9	729
10	1000
11	1331
12	1728

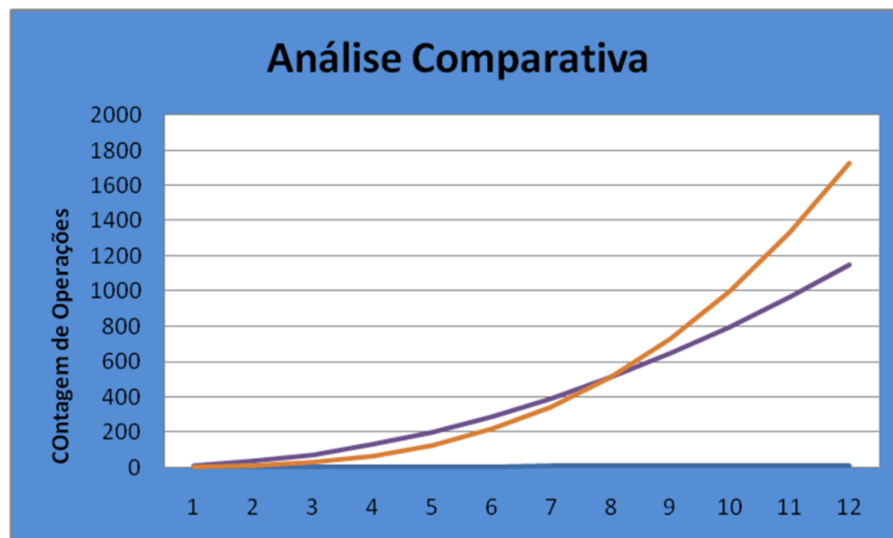


Análise Gráfica



Atividade 1

- ✓ Até um determinado valor de instância de entrada, o algoritmo B é melhor.
- ✓ A partir de um certo valor de n, o algoritmo A passa a ser melhor.
- ✓ O ponto de equilíbrio ocorre quando: $\mathbf{n^3 = 8n^2 \Rightarrow n^3 - 8n^2 = 0 \Rightarrow n^2 (n - 8) = 0}$
- ✓ Assim, n = 0 ou n-8 = 0. Portanto, o ponto de equilíbrio é **n = 8**
- ✓ Assim, o algoritmo B é mais eficiente até n = 7.



Atividade - 2

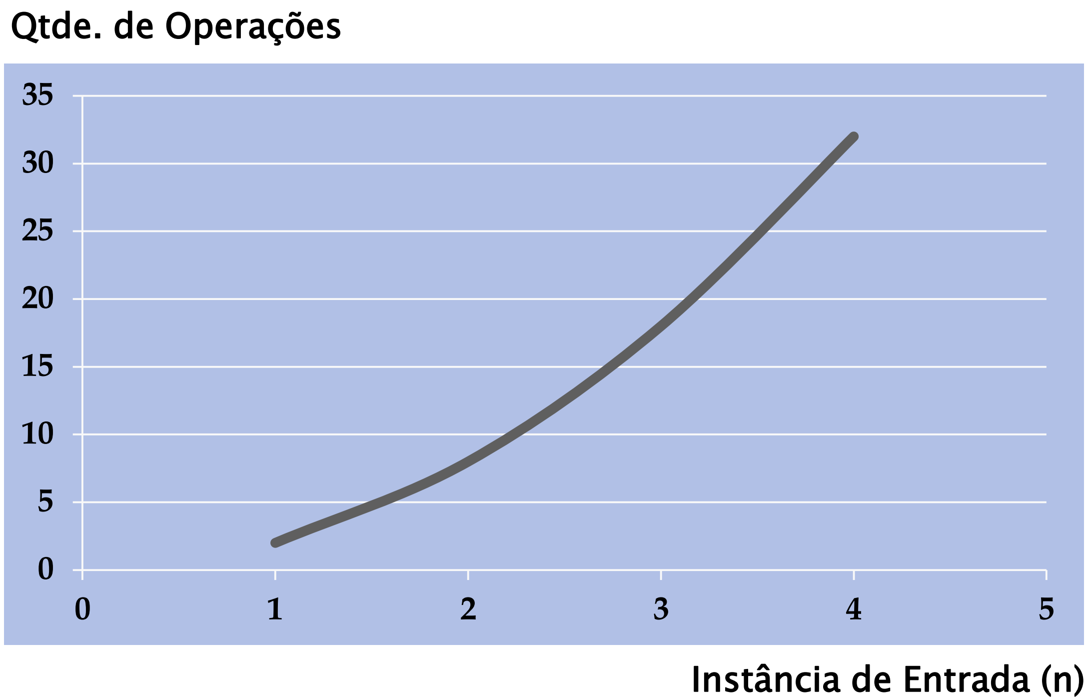
a) Um algoritmo tem complexidade $2n^2$. Num certo computador, num tempo t, o algoritmo resolve um problema de tamanho 25. Imagine agora que se tenha disponível um computador 100 vezes mais rápido. Qual o tamanho máximo de problema que o mesmo algoritmo resolve no mesmo tempo t no computador mais rápido ?

b) Considere o mesmo problema para um algoritmo de complexidade 2^n.

Atividade – 2a

Função de Complexidade $2n^2$

n	$2n^2$
1	2
2	8
3	18
4	32
5	50
10	200
25	1250



Atividade – 2a

- ✓ Analisando-se o comportamento da função de complexidade, pode-se afirmar que para **n**=25, são necessárias 1250 operações;
- ✓ Estas operações são executadas no computador antigo em um determinado tempo **t** ;
- ✓ No computador novo as operações são executadas **100** vezes mais rápidas ;
- ✓ Assim, no computador novo (no mesmo tempo t) pode-se executar 1250*100 = **125.000** operações.

Atividade – 2a

- Como o algoritmo é o mesmo (tanto no computador novo quando no antigo), a função de complexidade é a mesma ($f(n) = 2n^2$).

- Assim, no mesmo tempo t, o computador novo executa 125.000 operações, o que representa (certamente) uma instância maior.

- Teremos então: $f(n) = 2n^2 \Rightarrow 125000 = 2n^2$

$$\Rightarrow 62500 = n^2$$

$$\Rightarrow n = 250$$

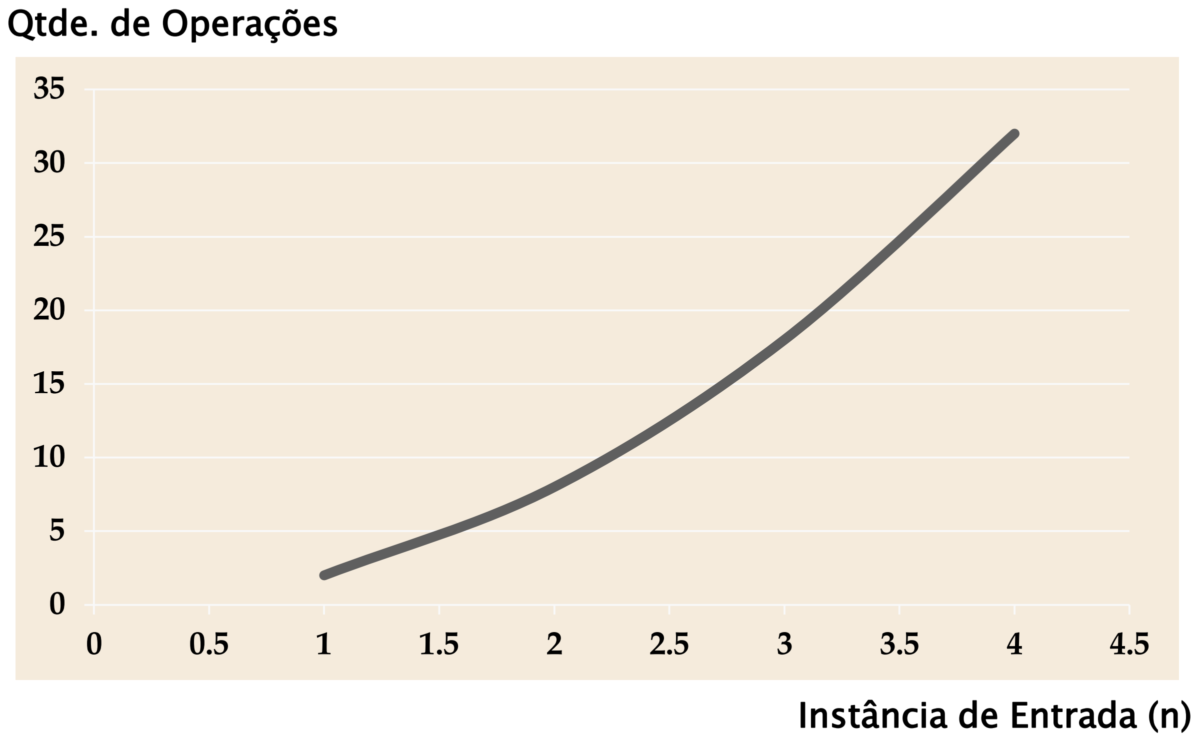
- Resposta: O tamanho máximo de problema que o mesmo algoritmo resolve no tempo t (no computador mais rápido) é 250.

- Observação: Embora o computador mais novo seja 100 vezes mais rápido, o tamanho do problema aumentou apenas 10 vezes (de 25 para 250).

Atividade – 2b

Função de Complexidade 2^n

n	2^n
1	2
2	4
3	8
4	16
5	32
10	1024
25	33554432



Atividade – 2b

- ✓ Analisando-se o comportamento da função de complexidade, podemos afirmar que para n=25, são necessárias 33.554.432 operações.
- ✓ Estas operações são executadas no computador antigo em um determinado tempo t.
- ✓ No computador novo as operações são executadas **100** vezes mais rápidas.
- ✓ Assim, no computador novo (no mesmo tempo t) podem-se executar 33.554.432 *100 = 3.355.443.200 operações.

Atividade – 2b

- Como o algoritmo é o mesmo (tanto no computador novo quando no antigo), a função de complexidade é a mesma ($f(n) = 2^n$).

- Assim, no mesmo tempo t, o computador novo executa 3.355.443.200 operações, o que representa (certamente) uma instância maior.

- Teremos então: $f(n) = 2^n \Rightarrow 2^n = 3355443200$

  $\log_2 2^n = \log_2 3355443200$

  $n = 31$

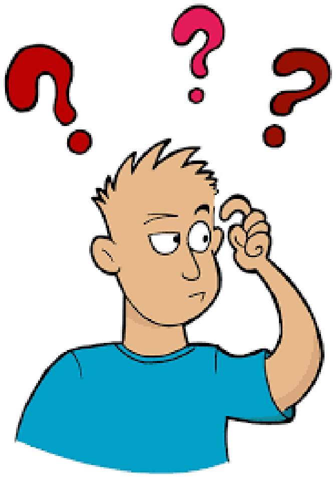
- Resposta: O tamanho máximo de problema que o mesmo algoritmo resolve no tempo t (no computador mais rápido) é 31.

- Observação: Embora o computador mais novo seja 100 vezes mais rápido, o tamanho do problema aumentou apenas 1,24 vezes (de 25 para 31) .

Atividade - 3

a) Suponha que uma empresa utiliza um algoritmo de complexidade n^2 que, em um tempo t, na máquina disponível, resolve um problema de tamanho x. Suponha que o tamanho do problema a ser resolvido aumentou em 20%, mas o tempo de resposta deve ser mantido. Para isso, a empresa pretende trocar a máquina por uma mais rápida. Qual percentual de melhoria no tempo de execução das operações básicas é necessário para atingir sua meta?

b) Suponha que no problema anterior, ainda se queira reduzir em 50% o tempo de resposta. Qual a melhoria esperada para a nova máquina?



Atividade – 3a

Máquina Velha

Qtde. de Operações	Tempo de cada Operação	Tempo Total
x^2	t_v	$tt_v = x^2 . t_v$

Máquina Nova

Qtde. de Operações	Tempo de cada Operação	Tempo Total
$(1.2x)^2$	t_n	$tt_n = 1.44x^2 . t_n$

Atividade – 3a

- O problema afirma que o tempo total da máquina nova deve ser igual ao tempo total da máquina velha;
- Assim, $tt_n = tt_v$
- Portanto, como: $tt_v = x^2 . t_v$ e $tt_n = 1.44.x^2 . t_n$
- Teremos: $x^2 . t_v = 1.44 . x^2 . t_n$
- Portanto: $t_v = 1.44 . t_n$
- Assim: $t_v = 1.44 . t_n \Rightarrow tv = 1.t_n + 0.44\, t_n \Rightarrow t_v = t_n + 44/100 . t_n$

A máquina velha é 44% mais lenta que a nova, ou
A máquina nova é 44% mais rápida que a velha

Atividade – 3b

Máquina Velha		
Qtde. de Operações	Tempo de cada Operação	Tempo Total
x^2	t_v	$tt_v = x^2. t_v$

Máquina Nova		
Qtde. de Operações	Tempo de cada Operação	Tempo Total
$(1.2x)^2$	t_n	$tt_n = 1.44x^2. t_n$

Atividade – 3b

- ✓ O problema afirma que o tempo total da máquina nova deve ser 50% inferior ao tempo total da máquina velha.

- ✓ Assim, $tt_n = (50\%)\ tt_v \Rightarrow tt_n = 0{,}5\ tt_v$

- ✓ Portanto, como: $tt_v = x^2 .\ t_v$ e $tt_n = 1{,}44.x^2 .\ t_n$

- ✓ Teremos: $1{,}44\ .\ x^2 .\ t_n = 0{,}5\ .\ x^2 .\ t_v$

- ✓ Portanto: $1{,}44\ .\ t_n = 0.5 .\ t_v$

- ✓ Assim: $2{,}88\ .\ t_n = t_v \Rightarrow t_v = t_n + 1{,}88 .\ t_n \Rightarrow t_v = t_n + 188/100 .\ t_n$

A máquina velha é 188% mais lenta que a nova, ou
A máquina nova é 188% mais rápida que a velha

Melhor Caso, Pior Caso e Caso Médio

- ✓ **Melhor caso**: menor tempo de execução sobre todas as entradas de tamanho n.
- ✓ **Pior caso**: maior tempo de execução sobre todas as entradas de tamanho n.
- ✓ **Caso médio** (ou caso esperado): média dos tempos de execução de todas as entradas de tamanho n.

Análise do Caso Médio

- ✓ Na análise do caso esperado, supõe-se uma distribuição de probabilidades sobre o conjunto de entradas de tamanho n e o custo médio é obtido com base nessa distribuição;

- ✓ A análise do caso médio é geralmente muito mais difícil de obter do que as análises do melhor e do pior caso;

- ✓ É comum supor uma distribuição de probabilidades em que todas as entradas possíveis são igualmente prováveis;

- ✓ Na prática isso nem sempre é verdade.

Notação Big O

- Ao ver uma expressão como **n+10** ou **n^2+1**, usualmente avalia-se o comportamento delas com valores pequenos de **n**;
- A análise de algoritmos faz exatamente o contrário: **ignoram-se** os valores pequenos de n e concentram-se em **grandes valores** de **n**;
- Para valores de n muito grandes, as funções n^2, $(3/2)n^2$, $9999n^2$, $n^2/1000$, n^2+100n, etc. crescem todas da mesma forma e portanto são todas equivalentes;
- Esse tipo de análise, no qual se considera somente valores muito grandes de **n**, é chamado **análise assintótica**;
- A notação **Big O**, é uma das notações utilizadas para **análises assintóticas**;
- Assim, $(3/2)n^2$ é **O**(n^2) e $9999n^2$ também é **O**(n^2).