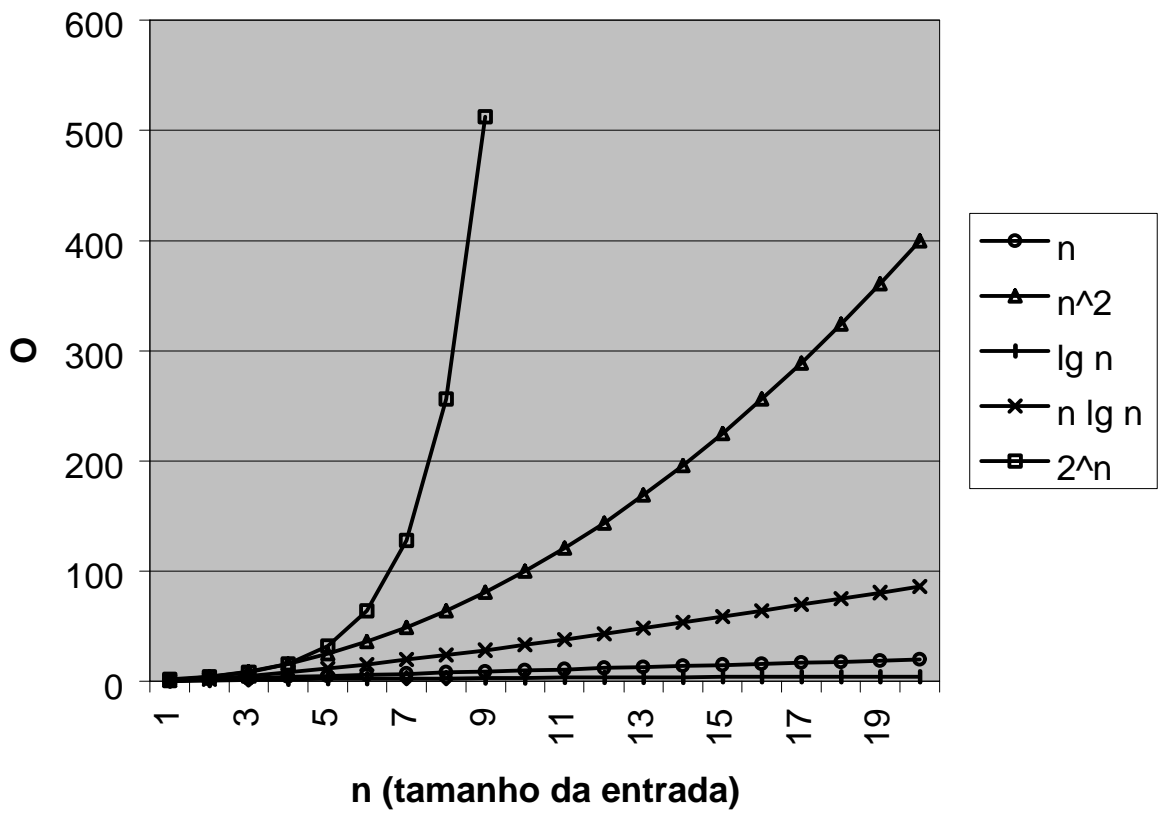


Comparação das curvas das funções de complexidade



**COMPARAÇÃO DO TEMPO DE EXECUÇÃO DAS VÁRIAS
FUNÇÕES DE COMPLEXIDADE
EM UM COMPUTADOR HIPOTÉTICO**

Função de Compl.	Tamanho de n					
	10	20	30	40	50	60
n	0,00001 seg	0,00002 seg	0,00003 Seg	0,00004 seg	0,00005 seg	0,00006 seg
n ²	0,0001 seg	0,0004 seg	0,0009 Seg	0,0016 seg	0,0025 seg	0,0036 seg
n ³	0,001 seg	0,008 seg	0,027 Seg	0,64 seg	0,125 seg	0,216 seg
n ⁵	0,1 seg	3,2 seg	24,3 Seg	1,7 min	5,2 min	13 min
2 ⁿ	0,001 seg	1 seg	17,9 min	12,7 dias	35,7 anos	366 sec
3 ⁿ	0,059 seg	58 min	6,5 anos	3855 sec	10 ⁸ sec	10 ¹³ sec

**AUMENTO DAS INSTÂNCIAS SOLUCIONÁVEIS COM O
APRIMORAMENTO DOS COMPUTADORES**

Função de complexid.	Tamanho da maior instância solucionável em 1h		
	Computador Atual	Computador 100 vezes mais rápido	Computador 1000 vezes mais rápido
n	N	100 N	1.000 N
n lg n	N ₁	22,5 N ₁	140,2 N ₁
n ²	N ₂	10 N ₂	31,6 N ₂
n ³	N ₃	4,6 N ₃	10 N ₃
2 ⁿ	N ₄	N ₄ +6	N ₄ +10
3 ⁿ	N ₅	N ₅ +4	N ₅ +6

OBTENDO A FUNÇÃO DE COMPLEXIDADE

1. O tempo de execução de um comando de atribuição, escrita ou leitura pode ser considerado $O(1)$.
2. O tempo de execução de uma seqüência de comandos é determinado pelo maior tempo de execução de qualquer comando da seqüência.
3. O tempo de execução de um comando de decisão é composto pelo tempo de execução dos comandos executados dentro do comando condicional, mais o tempo para avaliar a condição, que é $O(1)$.
4. O tempo para executar um *loop* é a soma do tempo de execução do corpo do loop, mais o tempo de avaliar a condição para terminação, multiplicado pelo número de iterações do *loop*. Geralmente o tempo para avaliar a condição de terminação é $O(1)$.

Exemplo:

```
void ordena (vetor A)
{
    int i, j, min, x;
    for (i = 1; i < n; i++)           // linha 1
    {
        min = i;                       // linha 2
        for (j = i + 1; j <=n; j++)   // linha 3
        {
            if (A[j - 1] < A[min -1]) // linha 4
                min = j;              // linha 5
        }
        x = A[min - 1];                // linha 6
        A[min - 1] = A[i - 1];         // linha 7
        A[i - 1] = x;                  // linha 8
    }
}
```

A análise deste algoritmo pode ser feita da seguinte maneira:

- n representa o tamanho da entrada
- existem dois loops, um dentro do outro
- o loop externo abrange as linhas 2 a 8 e o loop interno abrange as linhas 4 e 5
- o loop interno contém um comando de decisão e um de atribuição, ambos de tempo constante
- na decisão assume-se sempre o pior caso, ou seja, a linha 5 será sempre executada
- o tempo para incremento e teste do loop é sempre constante (linha 3)
- o tempo para executar uma iteração do loop interno (linhas 3, 4 e 5) é $O(\max(1,1,1)) = O(1)$
- o número de iterações do loop interno é (n-i), portanto, o tempo total do loop interno é $O((n-i) * 1) = O(n-i)$
- o loop externo contém, além do loop interno, comandos de atribuição de tempo constante (linhas 2, 6, 7, e 8)
- o tempo de execução de uma iteração do loop externo é $O(\max(1, (n-i), 1,1,1)) = O(n-i)$
- o número de iterações do loop externo (número de vezes que a linha 2 é executada) é igual a n – 1
- portanto, o tempo de execução total da função está limitado ao produto de uma constante pelo somatório de (n – i), a saber:

$$\sum_{1}^{n-1} (n-i) = (n * (n - 1) / 2) \Rightarrow (n^2 - n) / 2 \Rightarrow n^2/2 - n/2 \Rightarrow \boxed{O(n^2)}$$