Recursão, problema do máximo, crescimento de funções

"Talvez o mais importante princípio do bom projetista de algoritmos seja se recusar a estar satisfeito" - Aho, Hopcroft e Ullman, the design and analysis of computer algorithms, 1974.

Problema do máximo

Definição:

- Dado um vetor v de inteiros com tamanho n,
    - devolva o valor do maior elemento deste vetor.

Ideia de uma abordagem recursiva:

- Tome um elemento arbitrário do vetor.
- Encontre recursivamente o máximo do subproblema
    - que contém os demais elementos.
- Compare o elemento tomado com o máximo do subproblema
    - e devolva o maior deles.
- Observe que se o subproblema tem apenas um elemento,
    - então ele pode ser resolvido diretamente.

Algoritmo recursivo que reduz o vetor pelo fim.

```
int maximoRend(int v[], int n) {
   int x; // auxiliar que guarda o máximo do subproblema
   if (n == 1)
       return v[0];
   x = maximoRend(v, n - 1);
   if (x > v[n - 1])
       return x;
   return v[n - 1];
}
```

Algoritmo recursivo que reduz o vetor pelo início.

```
int maximoRbegin(int v[], int inicio, int n) {
   int x; // auxiliar que guarda o máximo do subproblema
   if (inicio == n - 1)
       return v[inicio];
   x = maximoRbegin(v, inicio + 1, n);
   if (x > v[inicio])
       return x;
   return v[inicio];
}
```

```c
int maximo(int v[], int n)
{
   return maximoRbegin(v, 0, n);
}
```

Eficiência de tempo:

- Qual a ordem do número de operações dos algoritmos recursivos?
    - Da ordem de n, i.e., O(n), pois cada chamada da função
        - realiza um número constante de operações locais
    - e desencadeia no máximo uma chamada recursiva
        - na qual o tamanho do subvetor é reduzido em uma unidade.
- Quiz1: Resolver a recorrência T(n) = 1 + T(n-1) com T(1) = 1.

Eficiência de espaço:

- Qual a quantidade de memória auxiliar utilizada?
    - Nesse caso é igual à eficiência de tempo, i.e., O(n).
- Isso acontece porque cada nova chamada recursiva
    - utiliza algumas variáveis auxiliares locais,
- que são colocadas na pilha de execução e só começam a ser liberadas
    - depois que a última chamada é resolvida.

Ideia de uma abordagem iterativa:

- Percorra o vetor da esquerda para a direita,
    - mantendo numa variável auxiliar
        - o maior valor do subvetor já percorrido.

Algoritmo iterativo.

```c
int maximoI(int v[], int n) {
   int max, i;
   max = v[0];
   for (i = 1; i < n; i++)
       if (max < v[i])
           max = v[i];
   return max;
}
```

Invariante e corretude:

- Qual o invariante principal do algoritmo iterativo?
    - No início de cada iteração temos que
        - max armazena o maior valor do subvetor v[0 .. i - 1].
- Observe que, ao final do laço, quando i = n,
    - o invariante garante que max é o maior elemento do vetor.

Quiz2: se houver mais de uma repetição do maior elemento,

- o algoritmo devolve o mais à esquerda.
- Aponte duas maneiras de fazê-lo devolver o mais à direita.

Eficiência de tempo:

- Qual o número de iterações em função de n?
  - É igual a n, pois o laço começa com i = 1 e vai até i = n - 1.
- Como o número de operações realizadas em cada iteração é constante,
  - o número total de operações é proporcional a n, i.e., O(n).

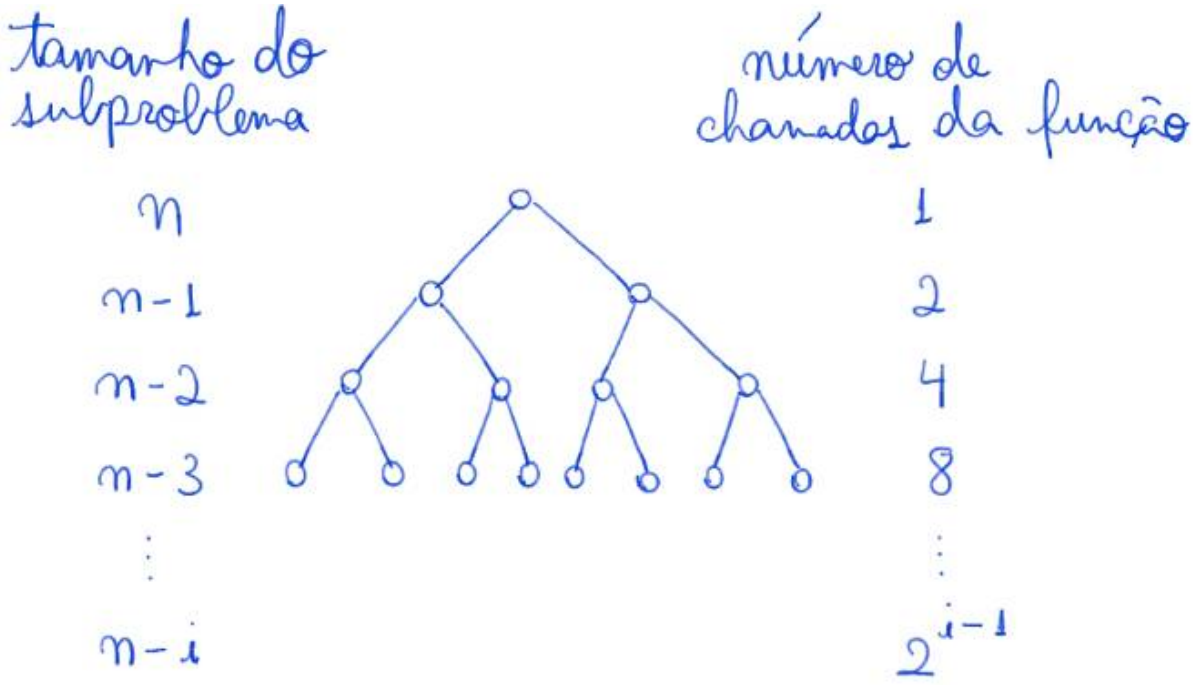
Eficiência de espaço:

- Qual a quantidade de memória auxiliar usada?
  - O(1), pois a função possui um pequeno número de variáveis simples.

Quiz3: qual a eficiência de pior caso do seguinte algoritmo recursivo?

```
int maximoR(int v[], int n)
{
   int x;
   if (n == 1)
       return v[0];
   if (maximoR(v, n - 1) > v[n - 1])
       return maximoR(v, n - 1);
   return v[n - 1];
}
```

- No pior caso ele leva tempo O(2^n), pois cada chamada da função
  - pode dar origem a duas chamadas recursivas.



- Isso torna esse algoritmo muito ineficiente, embora
  - ele esteja correto e, conceitualmente, seja muito parecido
    - com o primeiro algoritmo recursivo que estudamos.
- Para além da intuição da figura anterior,
  - esse resultado deriva da resolução da seguinte recorrência
    - T(n) = 2 T(n-1) + 1 com T(1) = 1

Resolvendo a Recorrência: $T(n) = 2T(n-1) + 1$ e $T(1) = 1$

Expandindo:

- $T(n) = 2T(n-1) + 1$
- $T(n-1) = 2T(n-2) + 1$
- $T(n-2) = 2T(n-3) + 1$
- $T(n-3) = 2T(n-4) + 1$
- …

Substituindo:

- $T(n) = 2T(n-1) + 1$
- $T(n) = 2(2T(n-2) + 1) + 1 = 4T(n-2) + 3$
- $T(n) = 4(2T(n-3) + 1) + 3 = 8T(n-3) + 7$
- $T(n) = 8(2T(n-4) + 1) + 7 = 16T(n-4) + 15$
- …

Arrumando:

- $T(n) = 2^1 T(n-1) + 2^1 - 1$
- $T(n) = 2^2 T(n-2) + 2^2 - 1$
- $T(n) = 2^3 T(n-3) + 2^3 - 1$
- $T(n) = 2^4 T(n-4) + 2^4 - 1$
- …

Generalizando:

- $T(n) = 2^i T(n-i) + 2^i - 1$

No final:

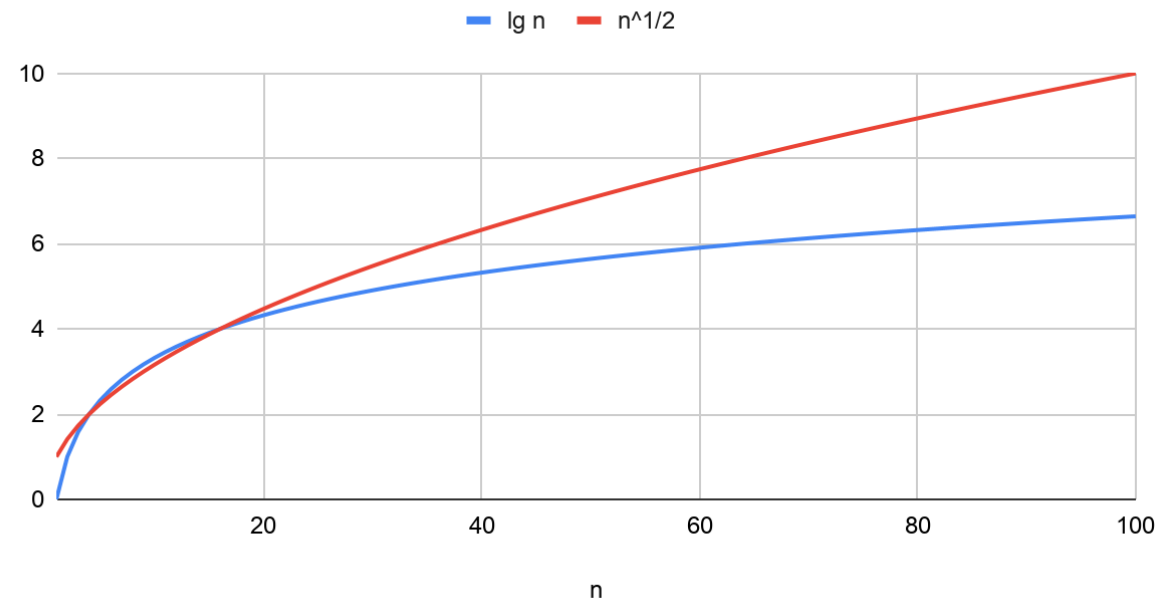
- $n - i = 1 \Rightarrow i = n-1$
- $T(n) = 2^{(n-1)} T(n - (n-1)) + 2^{(n-1)} - 1$
- $T(n) = 2^{(n-1)} T(1) + 2^{(n-1)} - 1$
- $T(n) = 2^{(n-1)} + 2^{(n-1)} - 1$
- $T(n) = 2^n - 1 = O(2^n)$

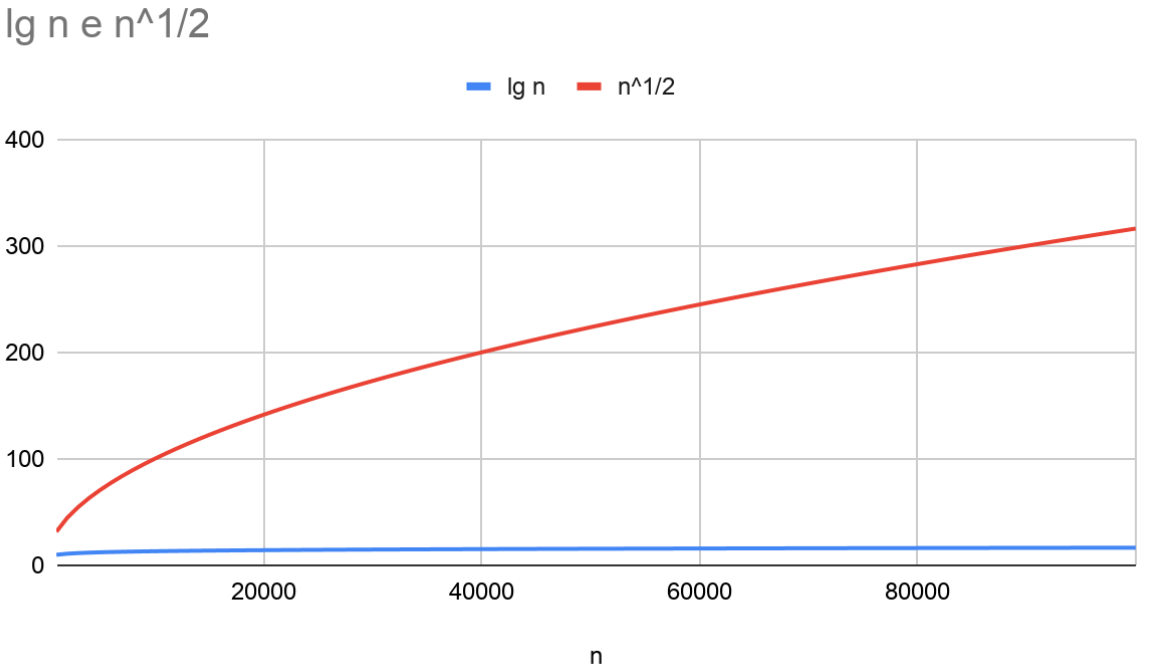
Crescimento de funções

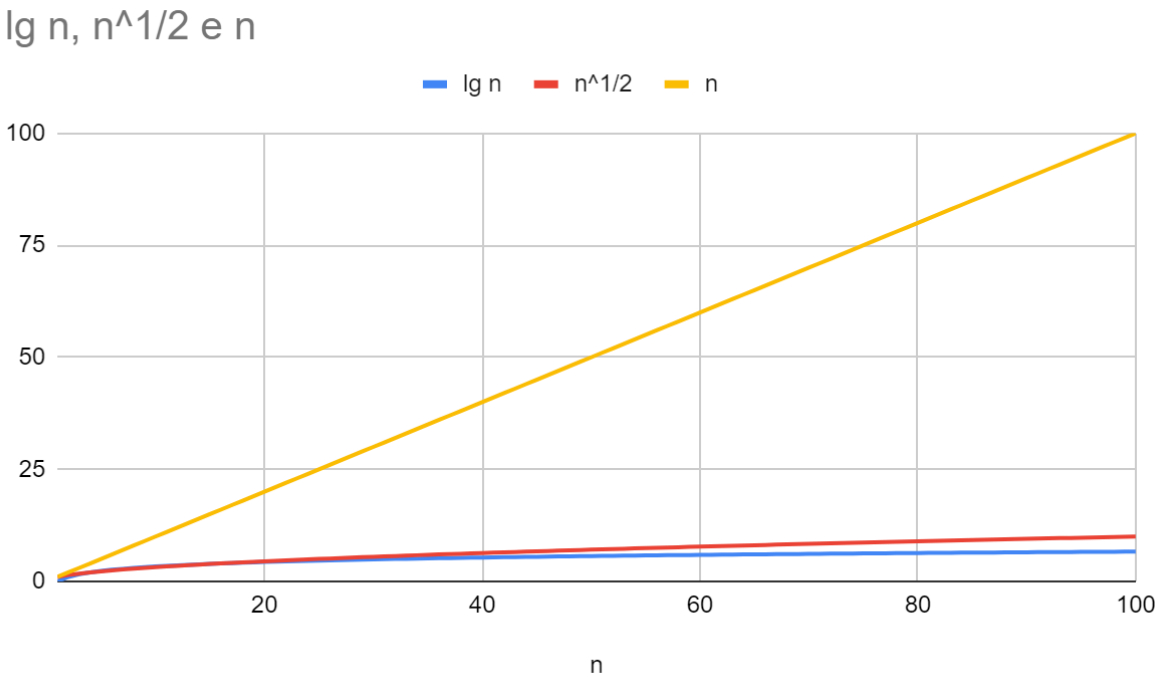
n	**10^3**	**10^6**	**10^9**
log_2 n	10	20	30
n^½	32	10^3	3*10^4
n	10^3	10^6	10^9
n log_2 n	10^4	2*10^7	3*10^10
n^2	10^6	10^12	10^18
n^3	10^9	10^18	10^27
2^n	10^300	10^300000	10^(3*10^8)

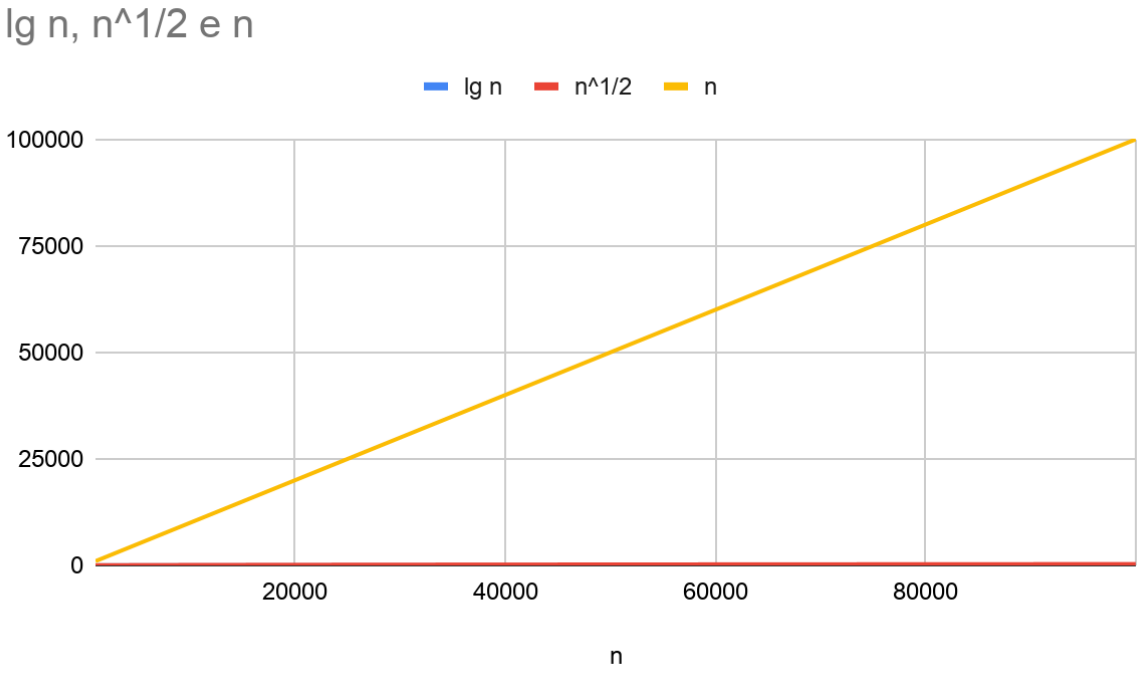
Interpretação temporal considerando um computador de 1GHz

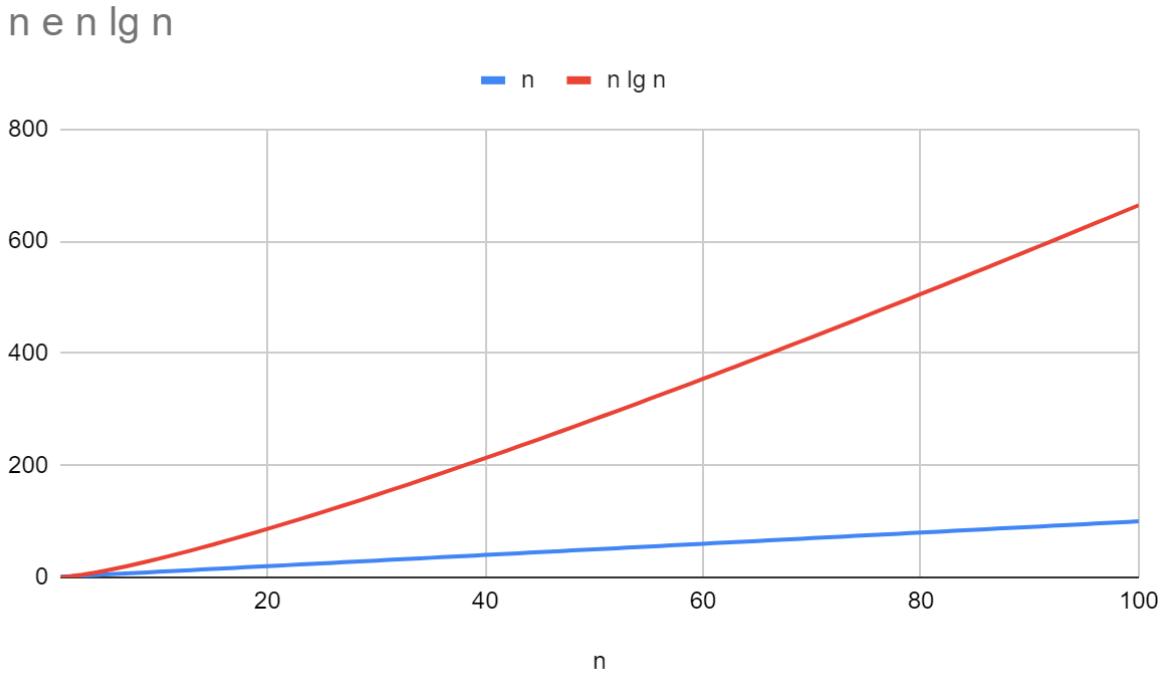
n	**10^3**	**10^6**	**10^9**
log_2 n	<< 1s	<< 1s	<< 1s
n^½	<< 1s	<< 1s	<< 1s
n	<< 1s	<< 1s	1s
n log_2 n	<< 1s	<1s	30s
n^2	<< 1s	16 min	31 anos
n^3	1s	31 anos	31709791 milênios
2^n	esquece...		

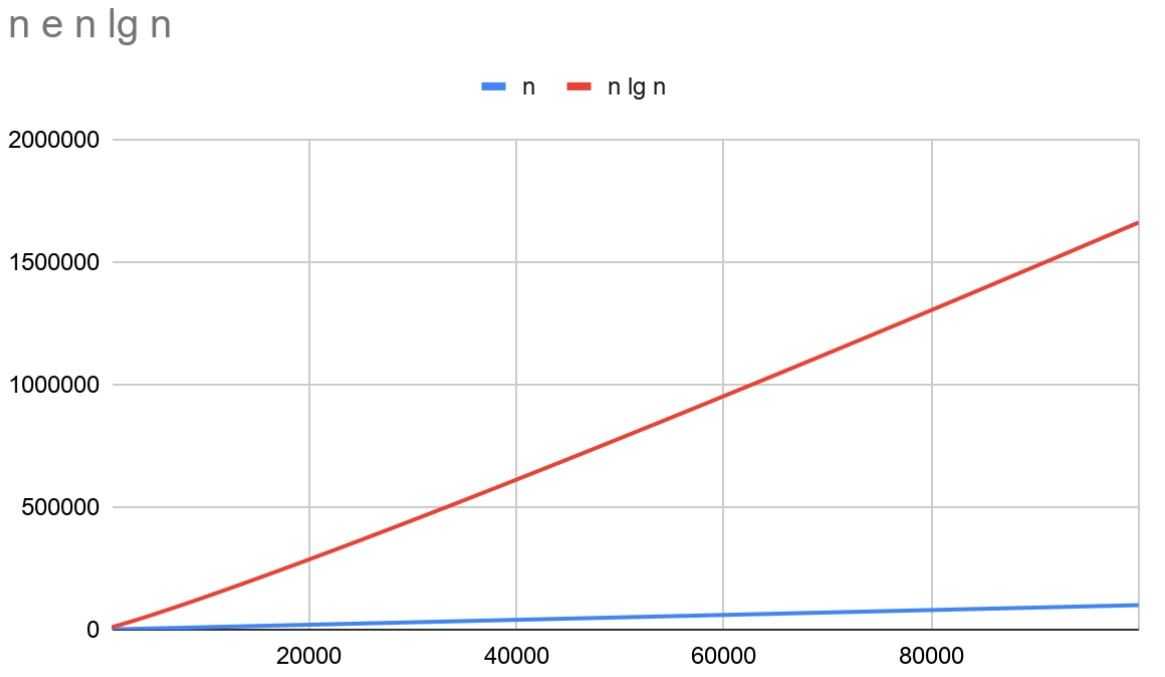

lg n e n^1/2
lg n
n^1/2
10
8
6
4
2
0
20
40
60
80
100
n

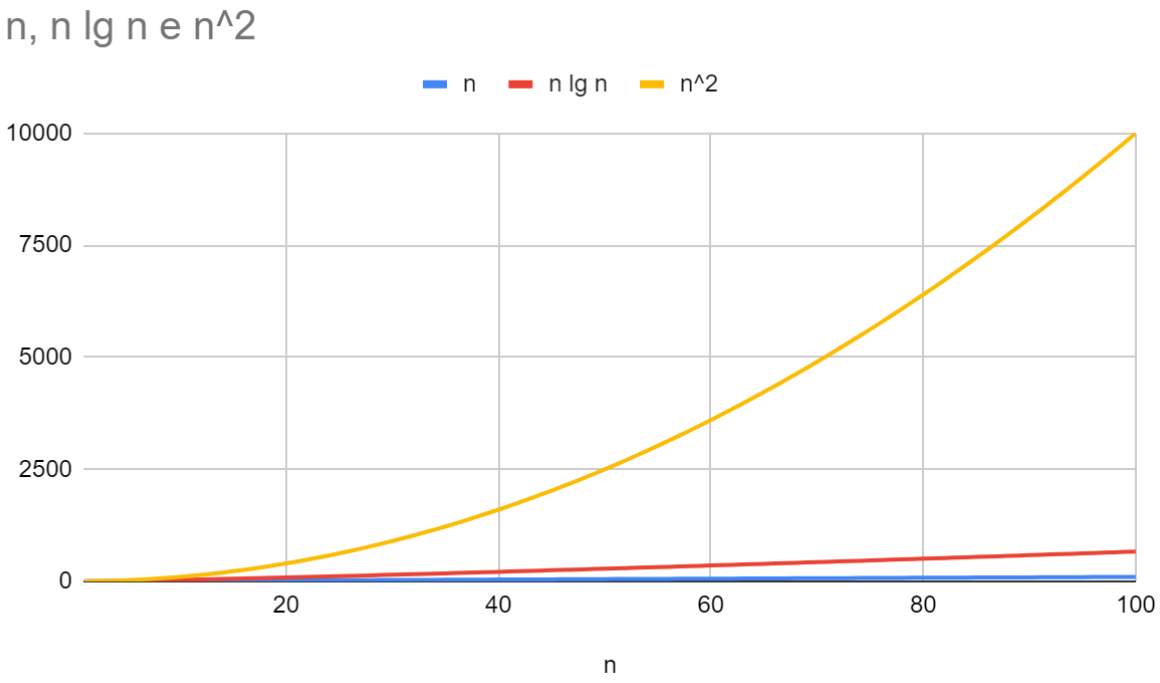

lg n e n^1/2
lg n
n^1/2
400
300
200
100
0
20000
40000
60000
80000
n

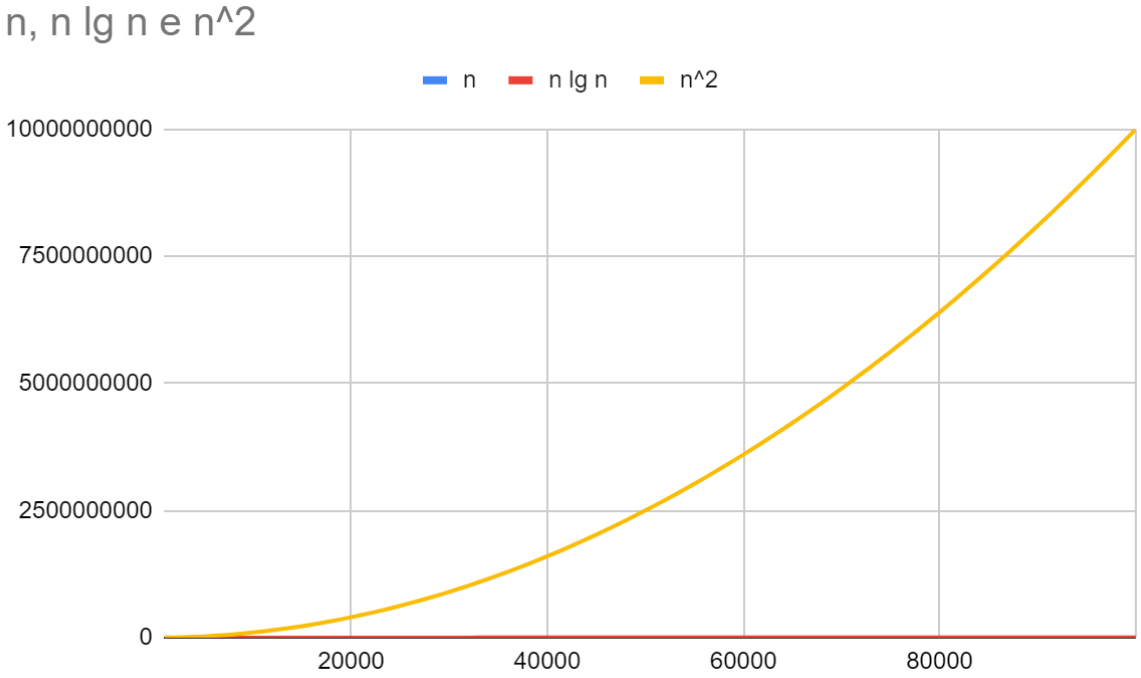

lg n, n^1/2 e n
lg n
n^1/2
n
100
75
50
25
0
20
40
60
80
100
n

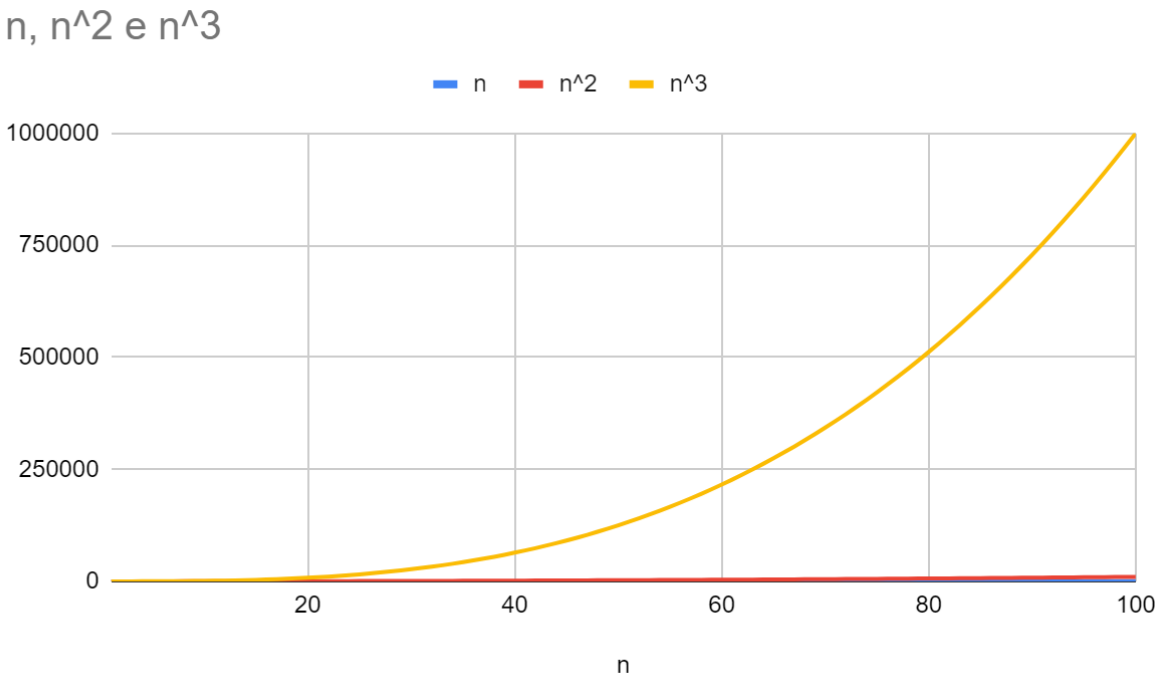

lg n, n^1/2 e n
lg n
n^1/2
n
100000
75000
50000
25000
0
20000
40000
60000
80000
n

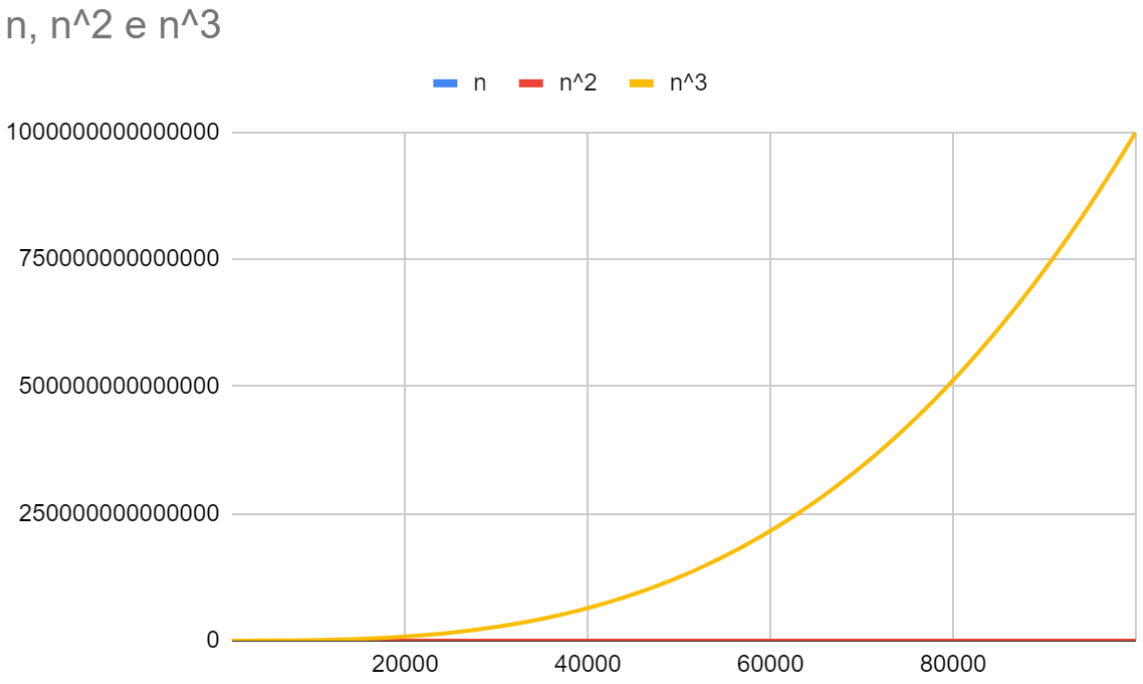

n e n lg n
n
n lg n
800
600
400
200
0
20
40
60
80
100
n

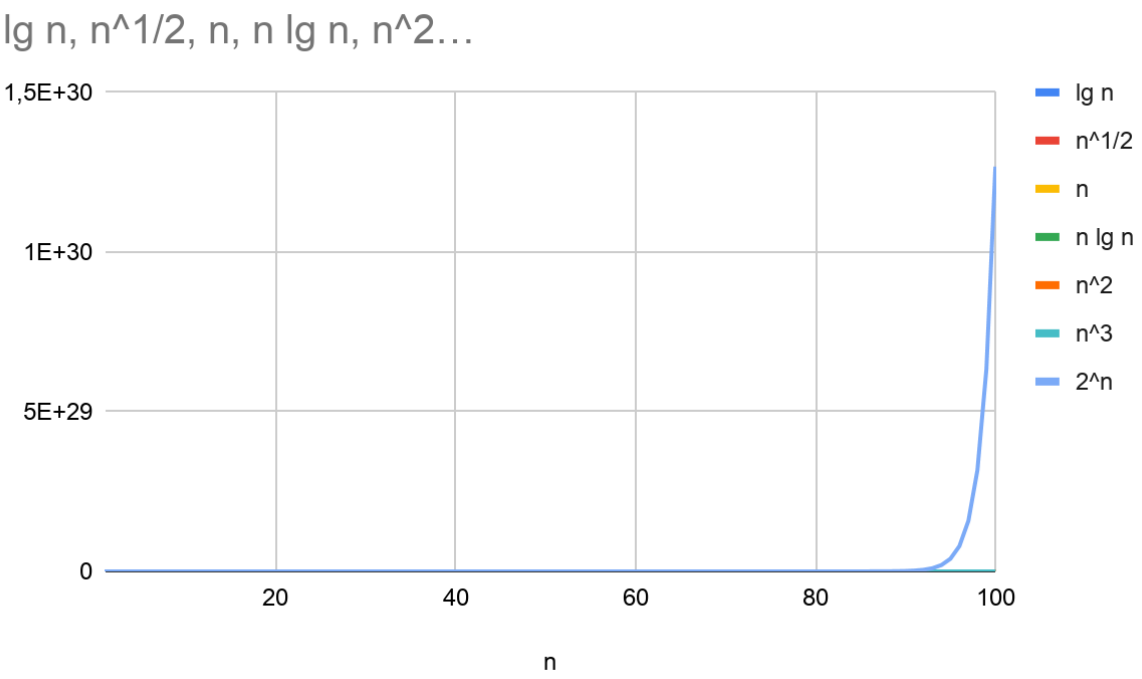

n e n lg n
n
n lg n
2000000
1500000
1000000
500000
0
20000
40000
60000
80000

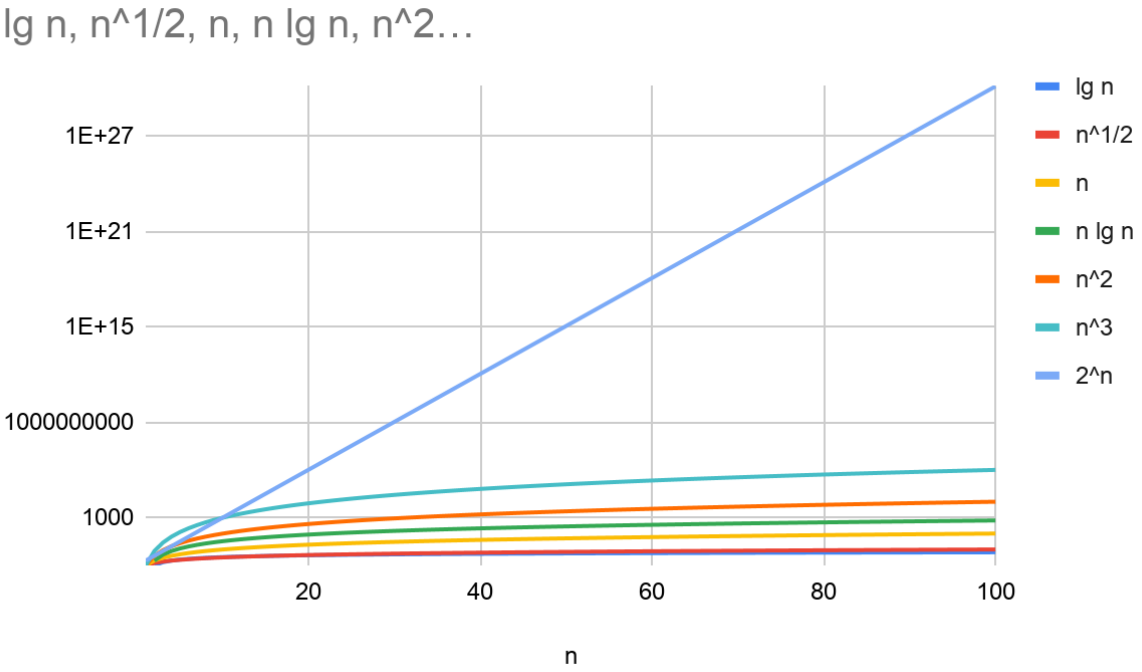

n, n lg n e n^2
n
n lg n
n^2
10000
7500
5000
2500
0
20
40
60
80
100
n


n, n lg n e n^2
n
n lg n
n^2
10000000000
7500000000
5000000000
2500000000
0
20000
40000
60000
80000


n, n^2 e n^3
n
n^2
n^3
1000000
750000
500000
250000
0
20
40
60
80
100
n


n, n^2 e n^3
n
n^2
n^3
1000000000000000
750000000000000
500000000000000
250000000000000
0
20000
40000
60000
80000


lg n, n^1/2, n, n lg n, n^2…
1,5E+30
1E+30
5E+29
0
20
40
60
80
100
n
lg n
n^1/2
n
n lg n
n^2
n^3
2^n


lg n, n^1/2, n, n lg n, n^2…
1E+27
1E+21
1E+15
1000000000
1000
20
40
60
80
100
n
lg n
n^1/2
n
n lg n
n^2
n^3
2^n