Análise de Complexidade em Algoritmos

INTRODUÇÃO

01

Os algoritmos resolvem os problemas através de uma entrada de dados e exibe estes resultados através de uma uma saída.

02

Um problema pode ser considerado um conjunto de dados que desejamos processar a fim de obter um resultado

03

Todos esse processo a fim de solucionar o problema demanda tempo e poder de processamento e em alguns casos se torna impossível de ser resolvido na prática devido ao tempo.

04

Mesmo que dois algoritmos resolvam o mesmo problema, um deles pode utilizar formas mais eficientes para chegar ao resultado e assim economizando tempo e processamento.

A importância da análise matemática

Q2

Uma boa análise matemática implica em em algoritmos mais sofisticados e que são capazes de resolver problemas de forma mais eficiente e até mesmo obter resultados até então impossíveis de serem obtidos devido o tempo necessário para processa-los. Tal análise busca fórmulas menos complexas e que chegam no mesmo resultado utilizando menos processamento.

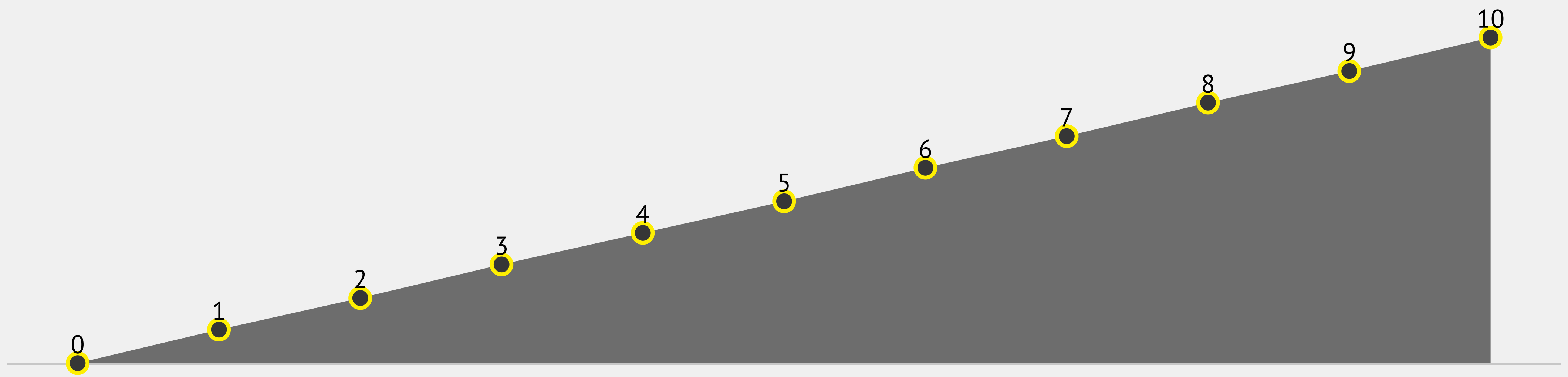
Contando instruções de um algoritmo

Q3

A complexidade de um algoritmo pode ser medida em cima das funções que o algoritmo é capaz de fazer e o volume de dados. Quanto maior o número de instruções um algoritmo possui, mais processamento será necessário para resolver o problema

Comportamento assintótico

O comportamento assintótico é a curva de crescimento de uma função gerada pelo processo de análise de algoritmos. O comportamento assintótico de f(n) representa o limite do comportamento do custo quando **n** cresce.



Tipos de análise assintótica

Q4

O

A notação **O** nos fornece uma simbologia simplificada para representar um limite superior de desempenho para um algoritmo. Um limite máximo de tempo que um algoritmo leva para ser executado.

Ω

A notação **Ω** nos fornece uma simbologia simplificada para representar um limite inferior de desempenho para um algoritmo. Um limite mínimo de tempo que um algoritmo leva para ser executado. Ou seja, a notação Ω é o inverso da notação Big O.

Θ

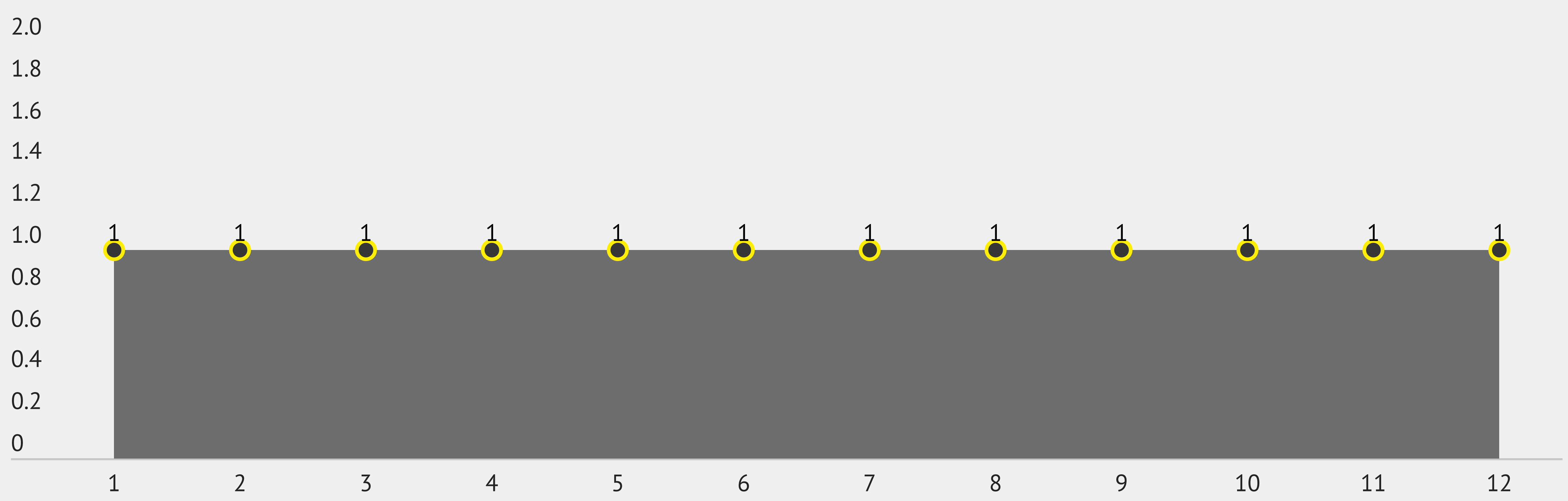
A notação **Θ** nos fornece uma simbologia simplificada para representar um limite justo de desempenho para um algoritmo. Um limite exato de tempo que um algoritmo leva para ser executado. Ou seja, a notação Θ representa o ponto de encontro entre as notações Ω (limite inferior) e Big O (limite superior).

Classes de problemas

05

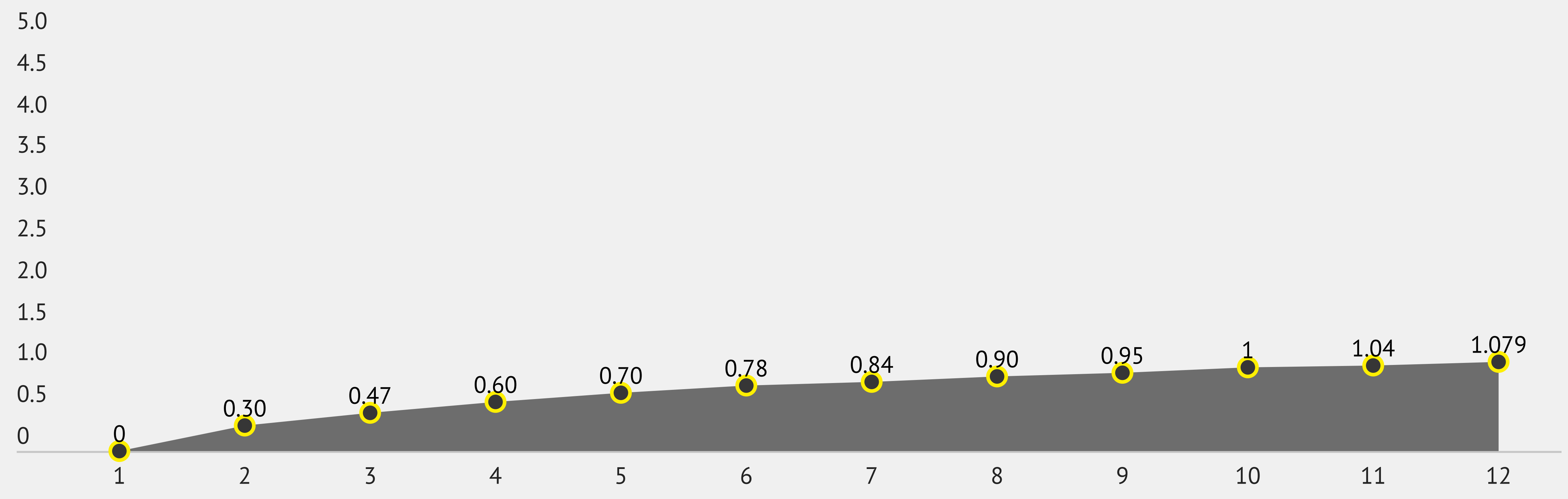
$F(n) = O(1)$

Algoritmos dessa classe possuem uma complexidade constante que não se altera independentemente do valor de n.



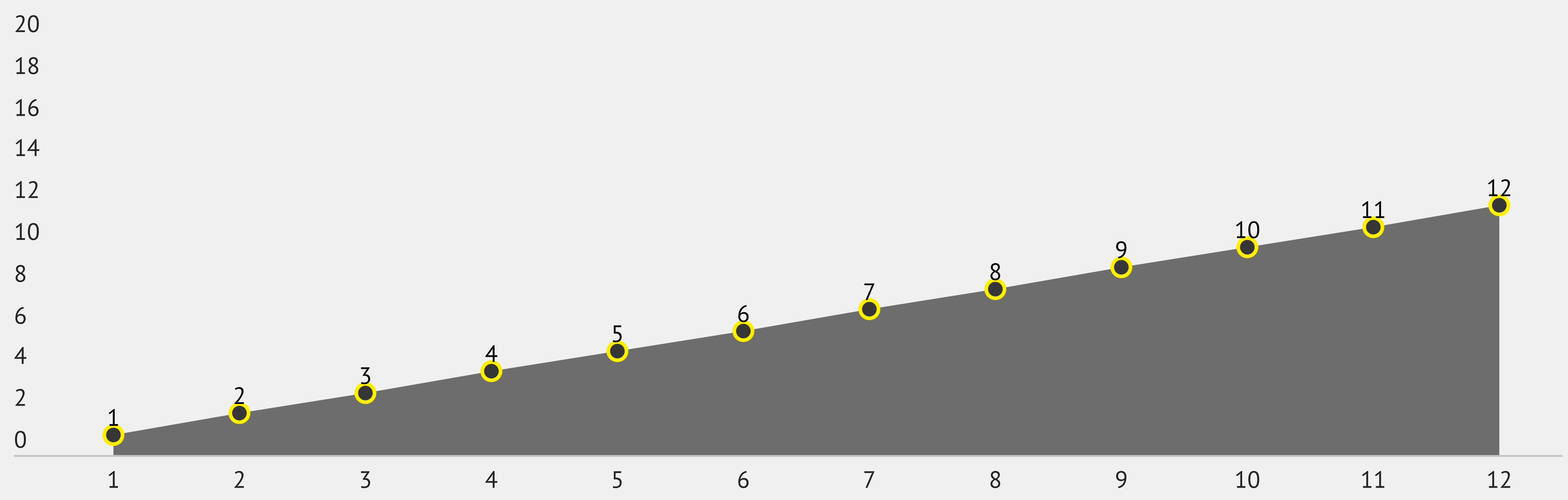
f(n) = O(log n)

Algoritmos dessa classe crescem de forma logarítmica, são conhecidos por dividir um problema em outros menores como por exemplo a busca binaria



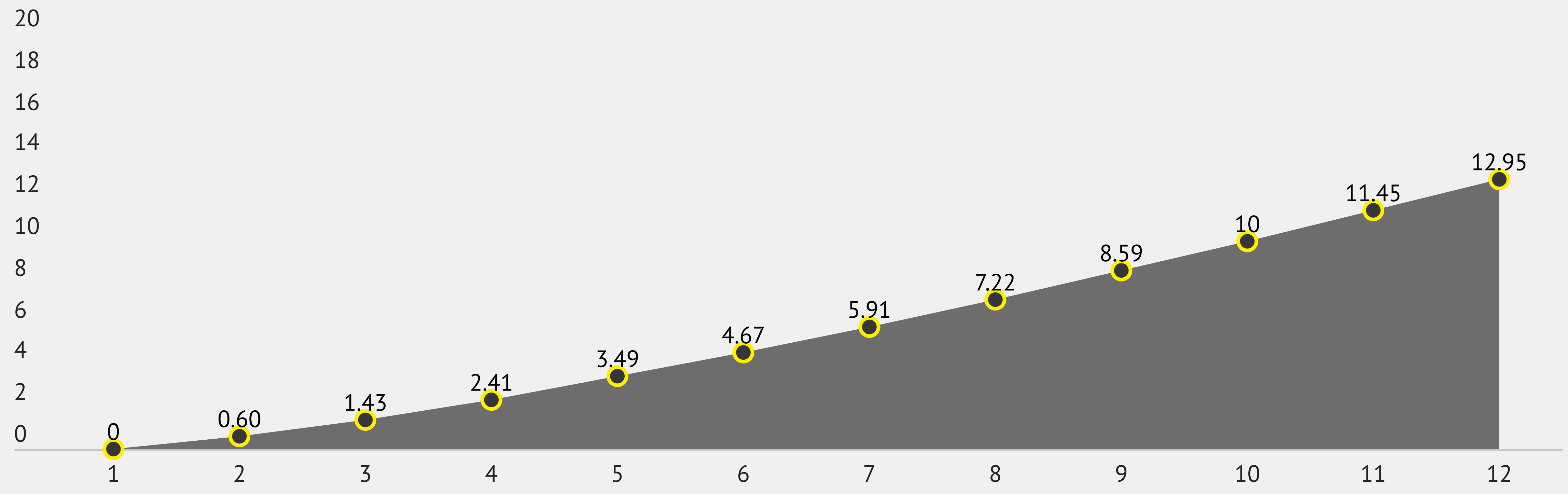
f(n) = O(n)

Algoritmos dessa classe crescem de forma linear em função do n. Ao dobrar o valor de n, o tempo de processamento também dobra.



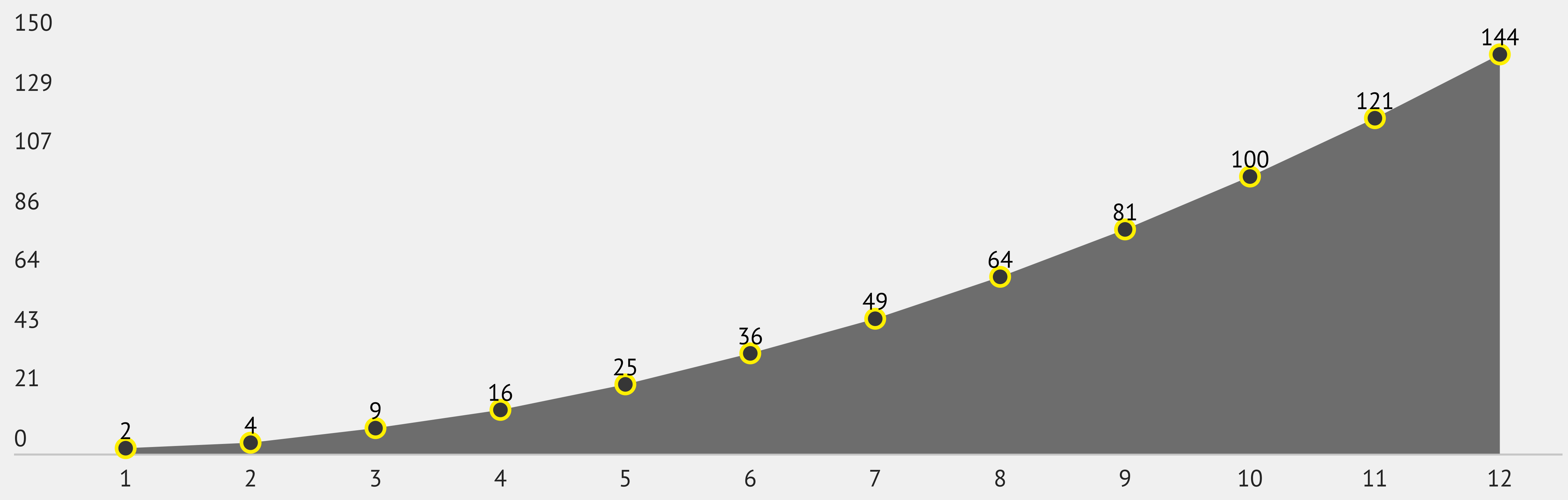
$f(n) = O(n \log n)$

Algoritmos dessa classe dividem um problema, em problemas menores e depois realiza a combinação das soluções obtidas



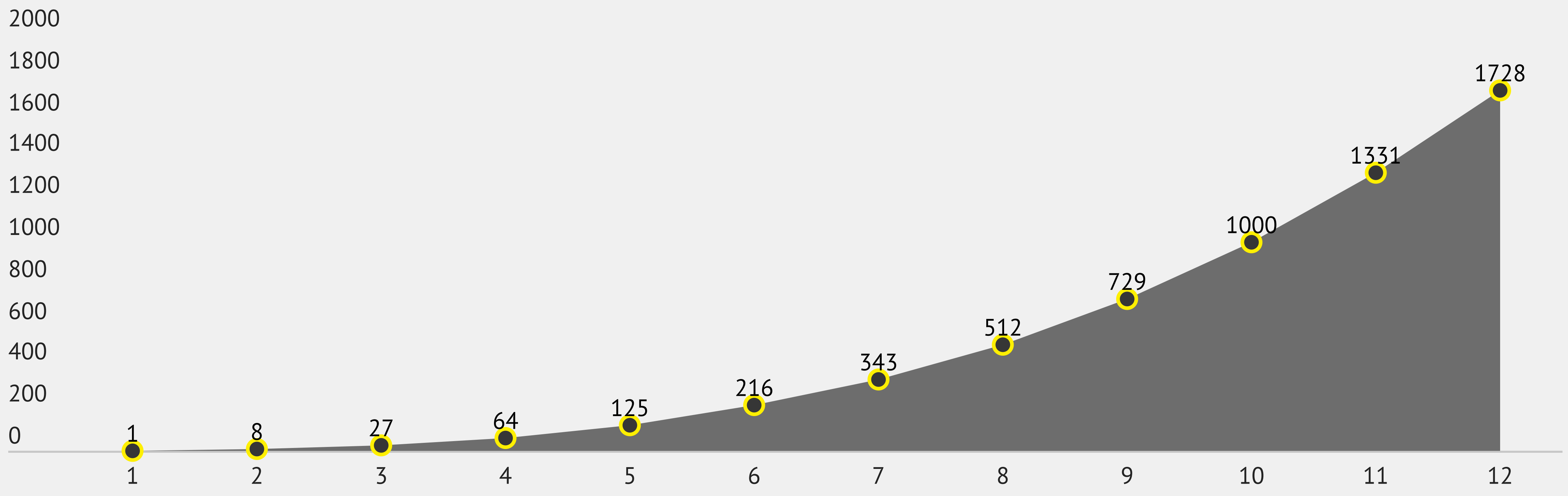
$f(n) = O(n^2)$

Algoritmos dessa classe possuem complexidade quadrática e seu tempo de processamento se multiplica por 4 ao dobrarmos o valor de n.



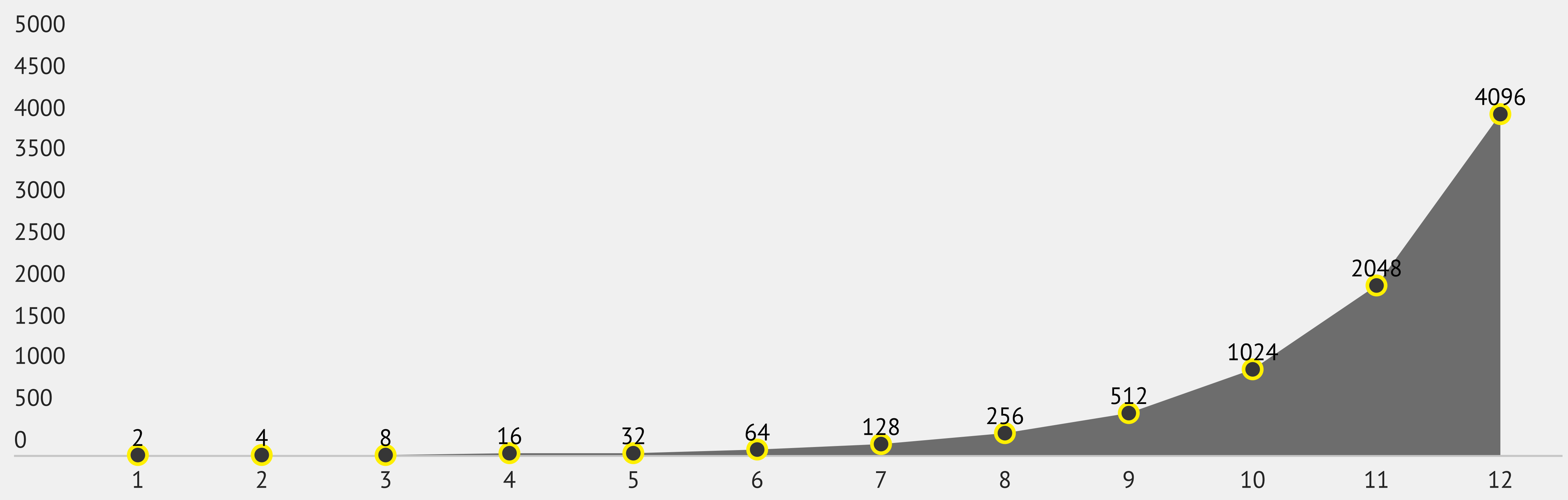
$f(n) = O(n^3)$

Algoritmos dessa classe possuem complexidade cúbica e seu tempo de processamento se multiplica por 8 ao dobrarmos o valor de n



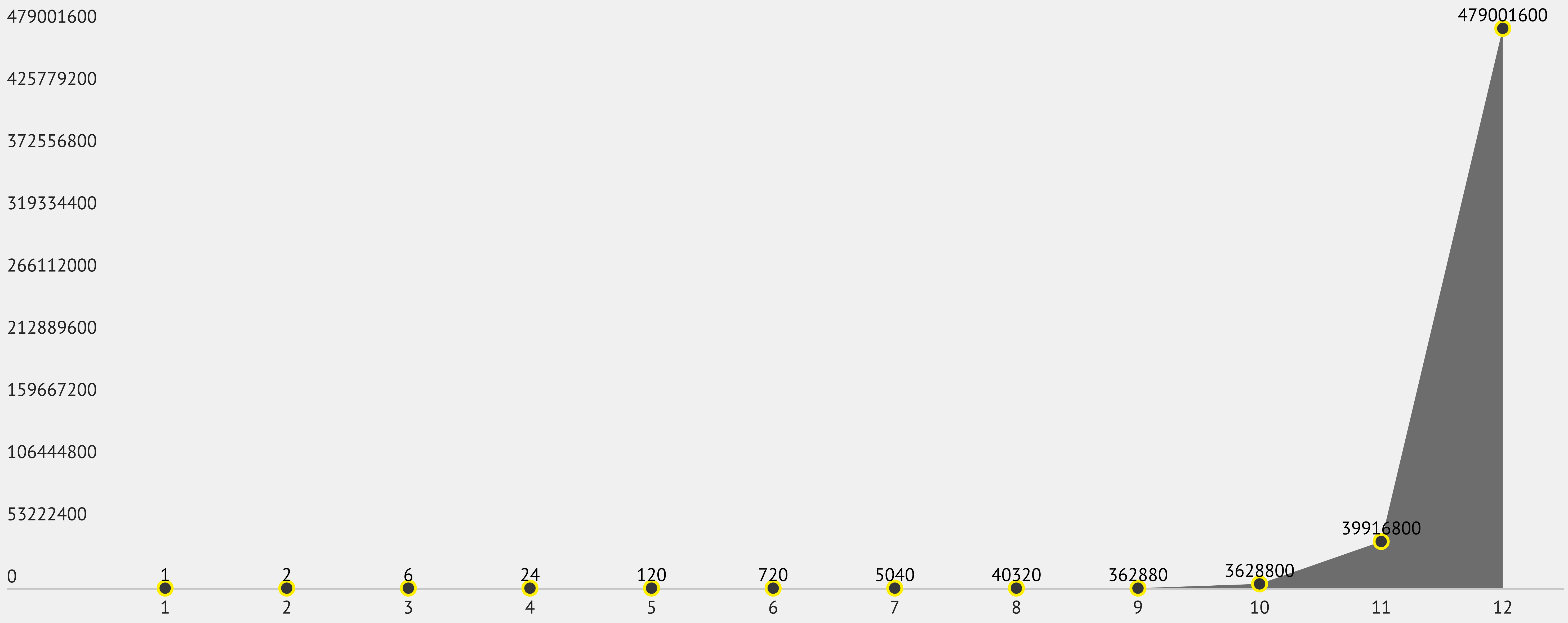
f(n) = O(2^n)

Algoritmos dessa classe possuem complexidade exponencial, não são uteis na prática por possuir um alto consumo de processamento à medida que n aumenta.



f(n) = O(n!)

Algoritmos dessa classe possuem complexidade exponencial, também não são uteis na prática e possuem um tempo de processamento ainda maior. Pequenos problemas levariam séculos para ser resolvidos nesta classe.



Conclusão

Q6

Construir algoritmos de forma eficiente é de extrema importância para desenvolver aplicações de forma eficiente. Mesmo que dois algoritmos tenham a capacidade de resolver o mesmo problema, um deles com uma boa aplicação pode gerar uma grande economia de tempo e processamento.