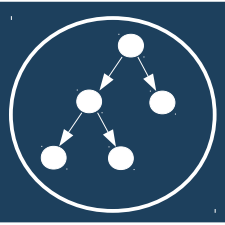


Data Structures and Algorithms

Алгоритмы.
Асимптотическая сложность.
O - нотация.

Анализ алгоритмов

Для анализа эффективности алгоритмов используется следующие критерии:

- **Временная эффективность** — время затрачиваемое на решение.
- **Пространственная эффективность** — объемы дополнительной памяти необходимые для решения.

Анализ временной и пространственной эффективности производится в зависимости от объема входных данных. Объем входных данных часто обозначают буквой n. Как следствие эффективность алгоритма описывают в виде функциональной зависимости от n.

Проблемы при анализе эффективности алгоритмов

При анализе временной эффективности возникает ряд проблем:

1) Разная аппаратная конфигурация ПК на котором исполняется алгоритм. Например один CPU может оказаться быстрее другого.
2) Использование разных оптимизаций со стороны компилятора.
3) Оптимизации которые вносит среда выполнения (например наличие JIT у JVM)
4) Сложность строгого математического вывода зависимости количества шагов алгоритма от объема входных данных.

Проблемы при анализе эффективности алгоритмов

При анализе пространственной эффективности возникает ряд проблем:

1) Разная аппаратная конфигурация ПК на котором исполняется алгоритм.
2) Использование разных оптимизаций со стороны компилятора.
3) Зависимость занимаемого места данными от типа платформы или операционной системы (например в C значение int может быть как 16 так и 32 бит).

Способы решения указанных выше проблем

Для решения проблемы анализа временной эффективности применяют одновременно два подхода.

1) Разделяют функциональную зависимость на две части. Первая часть зависит от конфигурации данного ПК, оптимизаций компилятора и т. д. Т.е. характеризует время выполнения одной команды именно на целевом ПК (принимают за константу) и функциональную зависимость от количества необходимых операций. В итоге функциональная зависимость приобретает вид

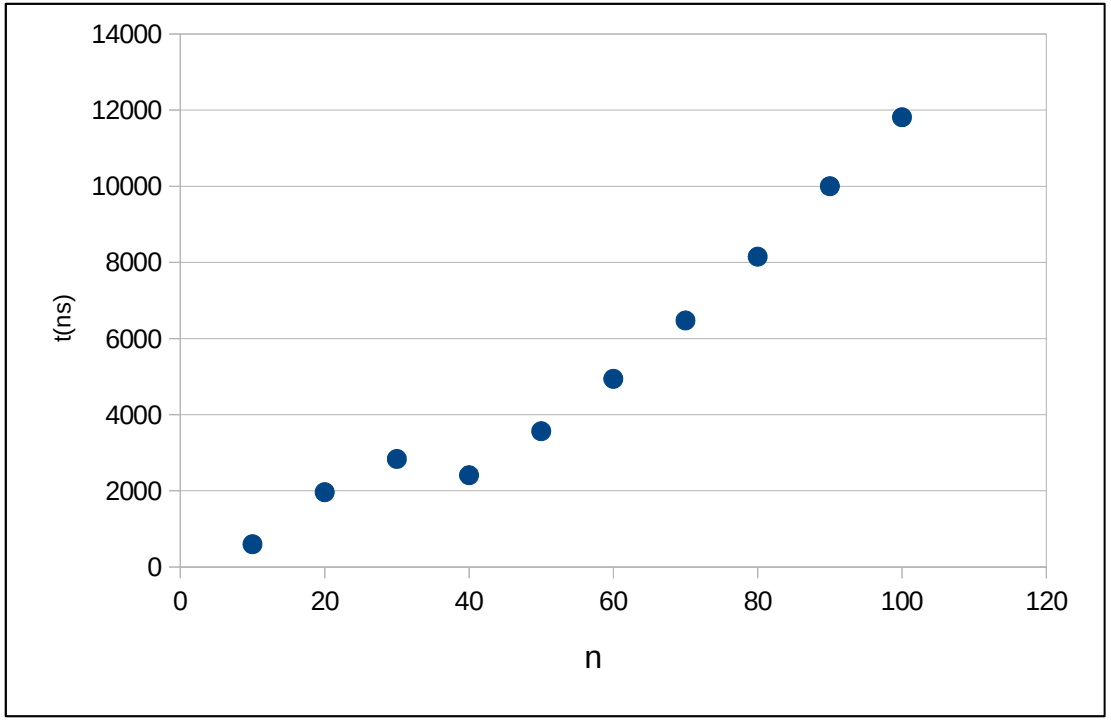
$$C_{PC} \cdot t(n)$$

C_{PC} — константа зависящая от конкретного исполнителя алгоритма (время выполнения одной конкретной операции)
t(n) — функциональная зависимость между объемом входных данных и количеством этих операций.

2) Для исследования функциональной зависимости t(n) — используют асимптотический подход.

О проблеме точного указания t(n)

Была написана программа реализующая алгоритм сортировки массива пузырьком. И проведен хронометраж его работы для разного объема входных данных (разный размер массива). Результат представлен на графике ниже.



Из графика видно, что установить точный вид зависимости времени выполнения от объема входных данных проблематично. Т.е. указать какой вид имеет зависимость t(n) сложно.

Но для малых объемов данных можно провести экспериментальные замеры. Однако обычно интересует как поведет себя алгоритм в случае именно больших объемов данных. В таком случае сам вид t(n) не важен. Важно как быстро она растет.

Это и есть по сути асимптотическое исследование.

Асимптотическая сложность алгоритма

В ряде случаев проблематично (или даже не возможно) указать вид t(n). Поэтому оценку временной эффективности проводят для больших объемов входных данных и оценке подвергается порядок роста t(n). В таком случае можно говорить о асимптотической сложности алгоритма.

При этом алгоритм с меньшей асимптотической сложностью является более эффективным для всех входных данных, за исключением лишь, возможно, данных малого размера.

Т.е. исследуют именно как возрастает t(n) при увеличении n. Можно рассматривать как предел при $n \to \infty$.

O - нотация

Предположим у нас есть неизвестная функция t(n) и известная (желательно простая) функция g(n). В таком случае говорят, что :

$$t(n)=O(g(n))$$

В том случае если $\exists(C>0),\exists(n_0):\forall(n>n_0)\ |t(n)| \leqslant |C\cdot g(n)|$

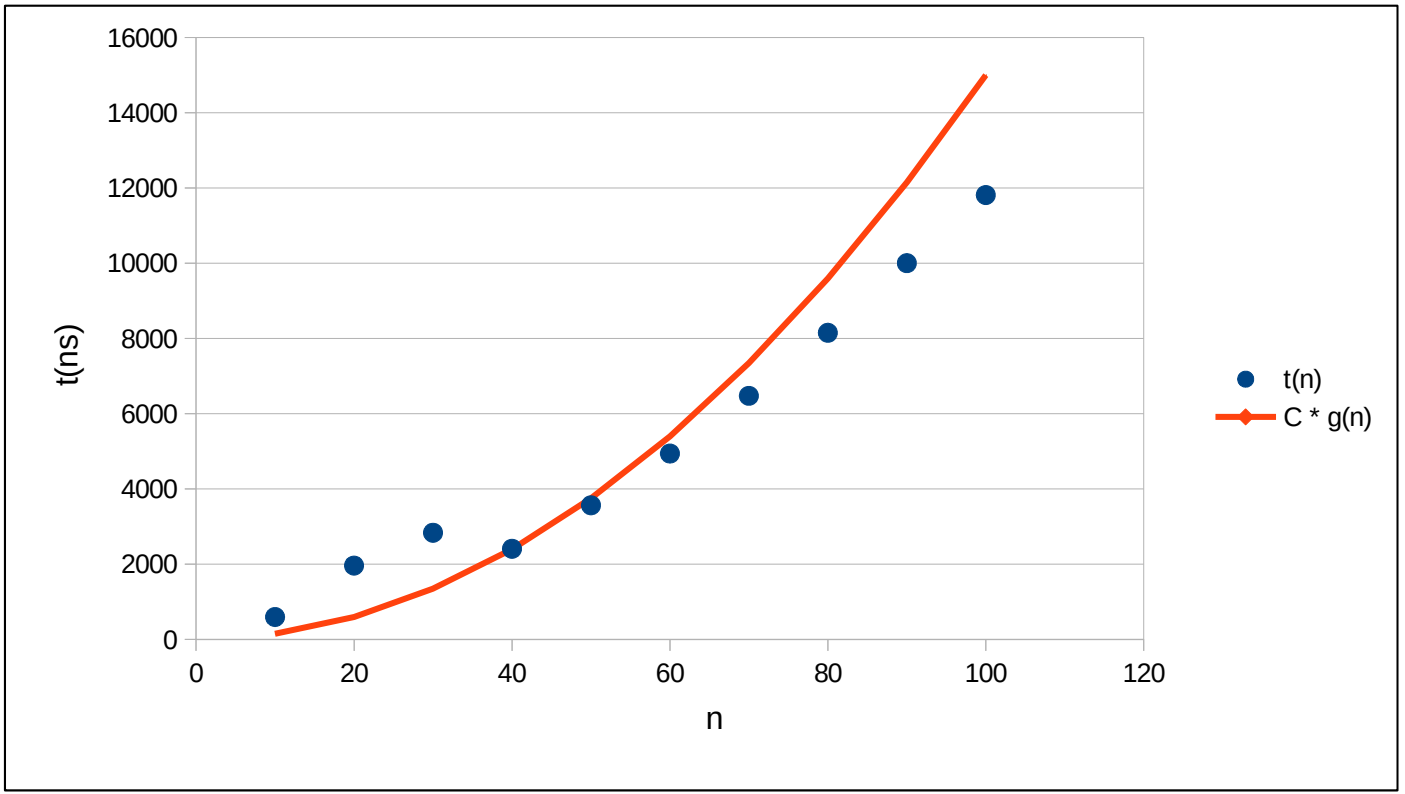
$$C \in \mathbb{R}$$
$$n, n_0 \in \mathbb{N}$$
$$t, g \in \mathbb{N} \to \mathbb{R}$$

Существует такая константа C больше нуля и такое n_0, что для всех $n > n_0$ значение функции t(n) меньше или равно значению C g(n).

В таком случае говорят что, t принадлежит классу функций, которые растут не быстрее, чем функция g (n) с точностью до постоянного множителя.

Оценка O представляет собой верхнюю асимптотическую оценку трудоёмкости алгоритма.

O - нотация



Начиная с какого то n функция C*g(n) растет быстрее чем t(n).

O - нотация

Из определения O нотации следует, что t ограничена сверху функцией g (с точностью до постоянного множителя) асимптотически в том случае если

$$t(n) \in O(g(n))$$

Это означает что порядок роста t меньше или равен порядку роста O(g(n)). Поэтому O(g(n)) можно (и нужно использовать) для оценки максимального времени решения задачи.

Именно поэтому **O нотация важна**. Ведь оценку времени выполнения (и как следствие запас по аппаратным ресурсам) для отказоустойчивых задач стоит производить в расчете на наихудший сценарий выполнения.

o - нотация

Предположим у нас есть неизвестная функция t(n) и известная (желательно простая) функция g(n). В таком случае говорят, что :

$$t(n)=o(g(n))$$

В том случае если $\exists(C>0),\exists(n_0):\forall(n>n_0)\ |t(n)|\ <\ |C\cdot g(n)|$

$$C\ \in\ \mathbb{R}$$
$$n, n_0\ \in\ \mathbb{N}$$
$$t, g\ \in\ \mathbb{N}\ \rightarrow\ \mathbb{R}$$

Существует такая константа C больше нуля и такое n_0, что для всех $n > n_0$ значение функции t(n) строго меньше значению C g(n).

Ω - нотация

Предположим у нас есть неизвестная функция t(n) и известная (желательно простая) функция g(n). В таком случае говорят, что :

$$t(n)=\Omega(g(n))$$

В том случае если $\exists(C>0),\exists(n_0):\forall(n>n_0)\ |C\cdot g(n)| \leqslant |t(n)|$

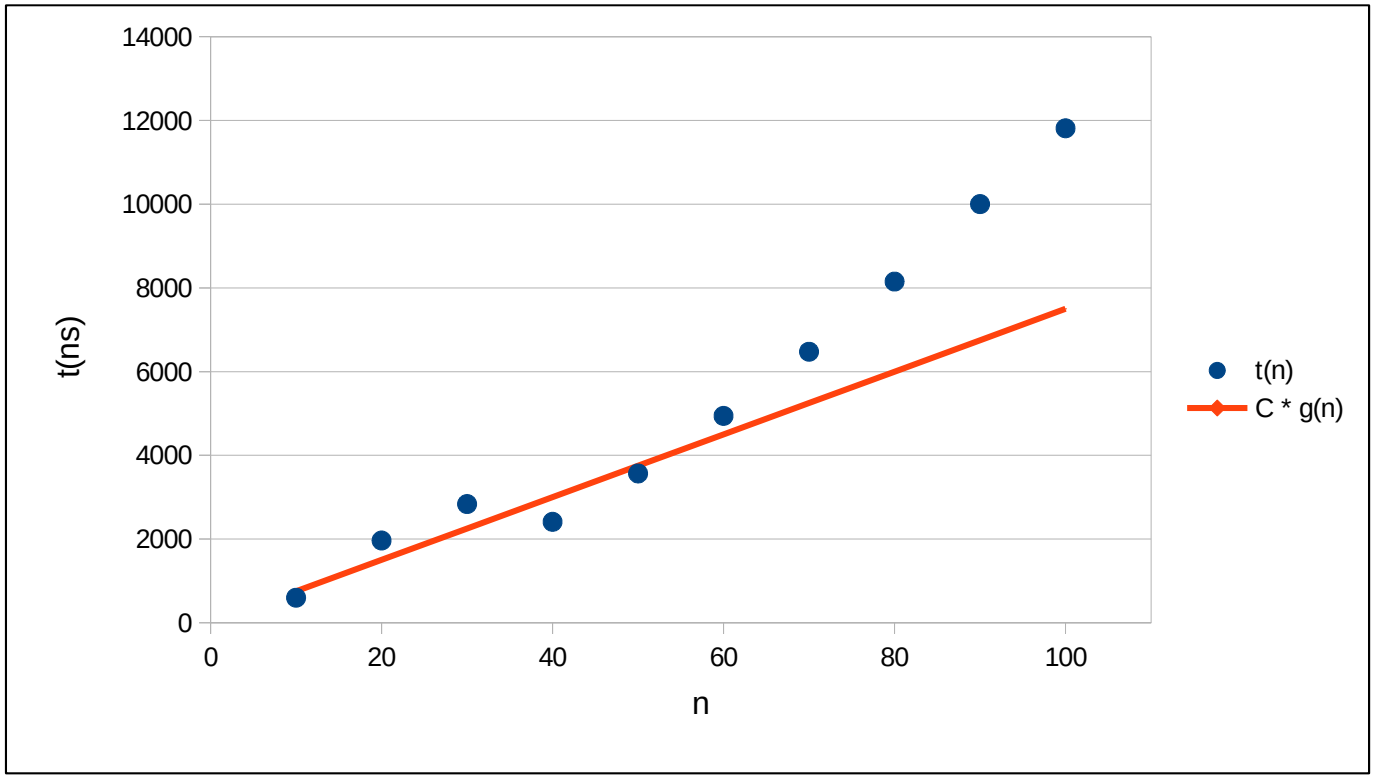
$$C \in \mathbb{R}$$
$$n, n_0 \in \mathbb{N}$$
$$t, g \in \mathbb{N} \to \mathbb{R}$$

Существует такая константа C больше нуля и такое n_0, что для всех $n > n_0$ значение функции t(n) больше или равно значению C умноженное на значение функции g(n).

В таком случае говорят что, t принадлежит классу функций, которые растут не медленнее, чем функция g (n) с точностью до постоянного множителя.

Оценка Ω задает нижнюю асимптотическую оценку роста функции t(n).

Ω - нотация



Начиная с какого то n функция C*g(n) растет медленнее чем t(n).

ω - нотация

Предположим у нас есть неизвестная функция t(n) и известная (желательно простая) функция g(n). В таком случае говорят, что :

$$t(n)=\omega(g(n))$$

В том случае если $\exists(C>0),\exists(n_0):\forall(n>n_0)\ |C\cdot g(n)|\ <\ |t(n)|$

$$C\ \in\ \mathbb{R}$$
$$n, n_0\ \in\ \mathbb{N}$$
$$t, g\ \in\ \mathbb{N}\ \rightarrow\ \mathbb{R}$$

Существует такая константа C больше нуля и такое n_0, что для всех $n > n_0$ значение функции t(n) строго больше значения C умноженное на значение функции g(n).

В таком случае говорят что, t принадлежит классу функций, которые растут не медленнее, чем функция g (n) с точностью до постоянного множителя.

Оценка ω задает нижнюю асимптотическую оценку роста функции t(n).

Θ - нотация

Предположим у нас есть неизвестная функция t(n) и известная (желательно простая) функция g(n). В таком случае говорят, что :

$$t(n)=\Theta(g(n))$$

В том случае если $\exists(C_1>0, C_2>0), \exists(n_0): \forall(n>n_0)\ |C_1 \cdot g(n)| \leqslant |t(n)| \leqslant |C_2 \cdot g(n)|$

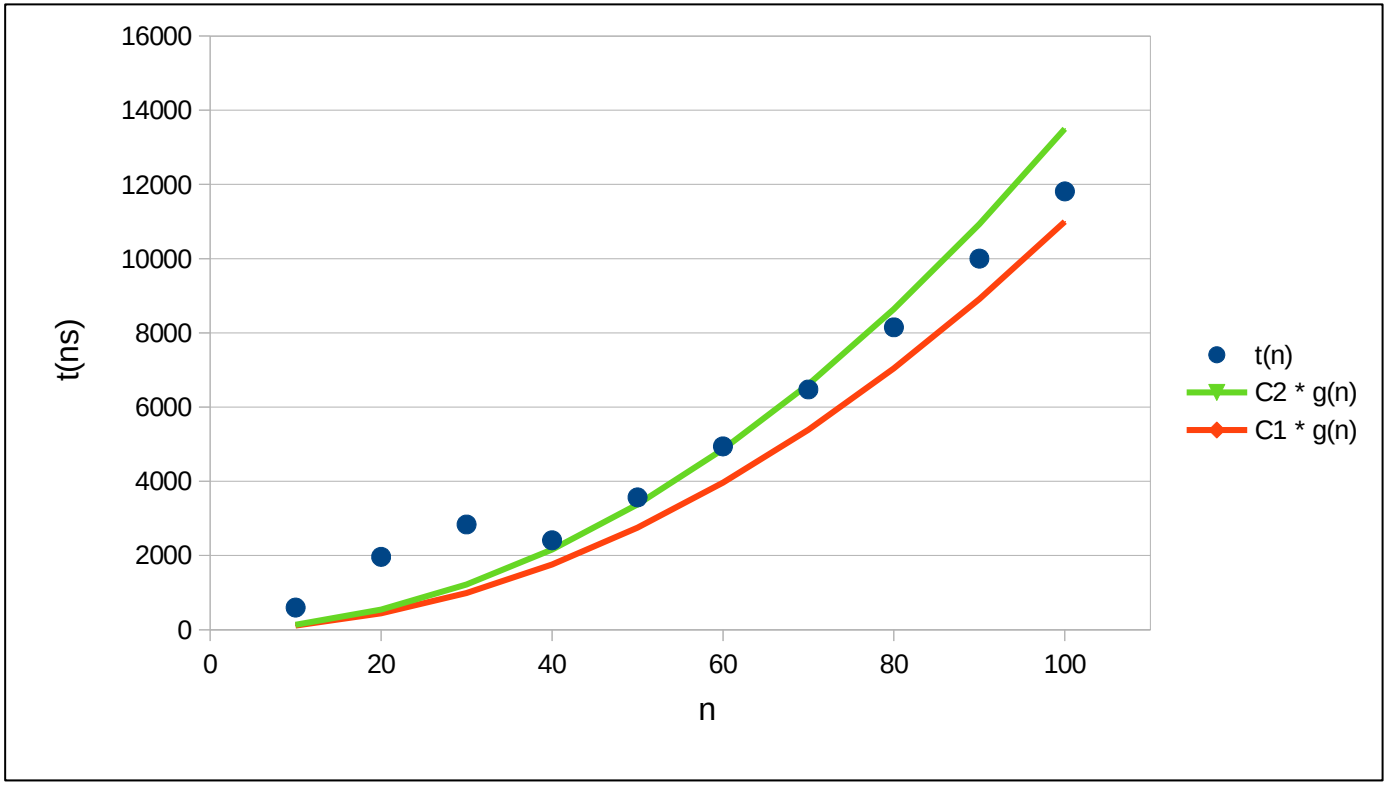
$$C_1, C_2 \in \mathbb{R}$$
$$n, n_0 \in \mathbb{N}$$
$$t, g \in \mathbb{N} \to \mathbb{R}$$

Существует такая константы C_1,C_2 больше нуля и такое n_0, что для всех $n > n_0$ значение функции t(n) больше или равно значению C_1 g(n) но меньше или равно значению C_2g(n).

В таком случае говорят ещё, что функция g(n) является асимптотически точной оценкой функции t(n).

Θ даёт одновременно верхнюю и нижнюю оценки роста функции.

Θ - нотация



Начиная с какого то n функция C1*g(n) растет медленнее чем t(n), а C2*g(n) быстрее.

Θ - нотация

Стоит отметить что хоть Θ и дает асимптотически точную оценку роста функции t(n) ее вычисление также является в проблематичным или слишком трудоемким.

Хотя именно Θ является наиболее желаемой оценкой, все же гораздо чаще используется оценка в виде O.

Асимптотическая эквивалентность двух функций

Предположим у нас есть неизвестная функция t(n) и известная (желательно простая) функция g(n). Такие функции называются асимптотически эквивалентными

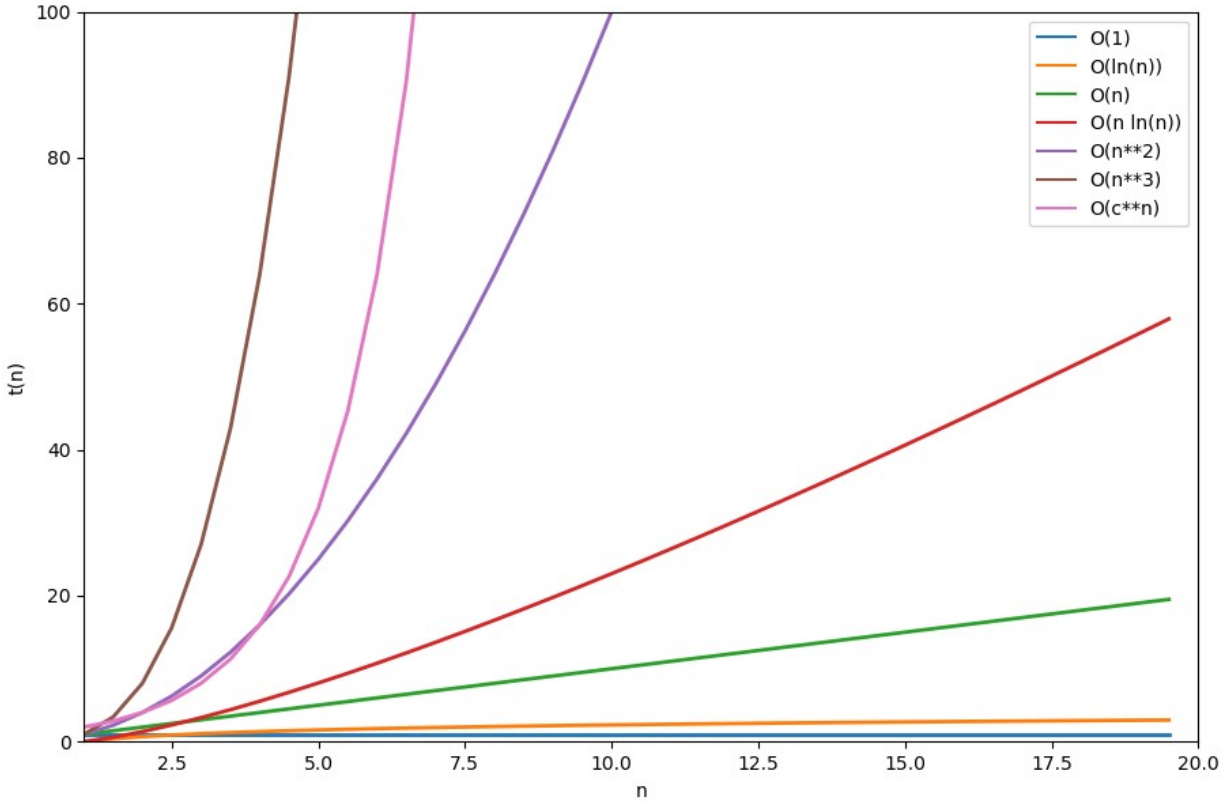
$$t(n) \sim g(n)$$

В том случае если $\lim_{n \to \infty} \frac{t(n)}{g(n)} = 1$

Некоторые асимптотические сложности

Сложность	Описание	Пример
O(1)	Константная. Не зависит от объема данных.	Получение элемента массива по индексу
O(ln(n))	Логарифмическая	Бинарный поиск
O(n)	Линейная.	Поиск элемента не сортированном массиве
O(n ln(n))	Линеаритмичная.	Быстрая сортировка
$O(n^2)$	Квадратичная.	Сортировка выбором
$O(n^3)$	Кубическая.	Перемножение матриц
$O(n^k)$	Полиномиальная.	Алгоритм Кармаркара
$O(C^n)$	Экспоненциальная.	Задачи полного перебора графа
O(n!)	Факториальная.	Комбинаторные алгоритмы

Графики асимптотических сложностей



O - нотация

Но как этим пользоваться? Например если указанно, что сложность сортировки $O(n^2)$.

Это означает что время работы этого алгоритма возрастает медленнее чем функция $g(n) = n^2$ умноженная на константу.

Это же позволяет оценить как увеличится (в самом худшем случае) время работы алгоритма при увеличении объема данных в определенное количество раз.

Рассмотрим пример такой оценки. Предположим что массив из 2000000 объектов на заданном ПК сортируется 50 ms. Нужно оценить как возрастет время если нужно сортировать массив из 8000000 элементов.

$$t = C_{PC} \cdot t(n) \Rightarrow t \leqslant C_{PC} \cdot C \cdot g(n) \Rightarrow \frac{t_1}{t_2} \leqslant \frac{C_{PC} \cdot C \cdot g(n_1)}{C_{PC} \cdot C \cdot g(n_2)} \Rightarrow t_2 \leqslant t_1 \cdot \frac{n_2^{\,2}}{n_1^{\,2}} = 50 \cdot \frac{8000000^2}{2000000^2} = 50 \cdot 16 = 800\,ms$$

Именно поэтому константу C обычно и не указывают, она все равно сокращается при оценке увеличения времени.

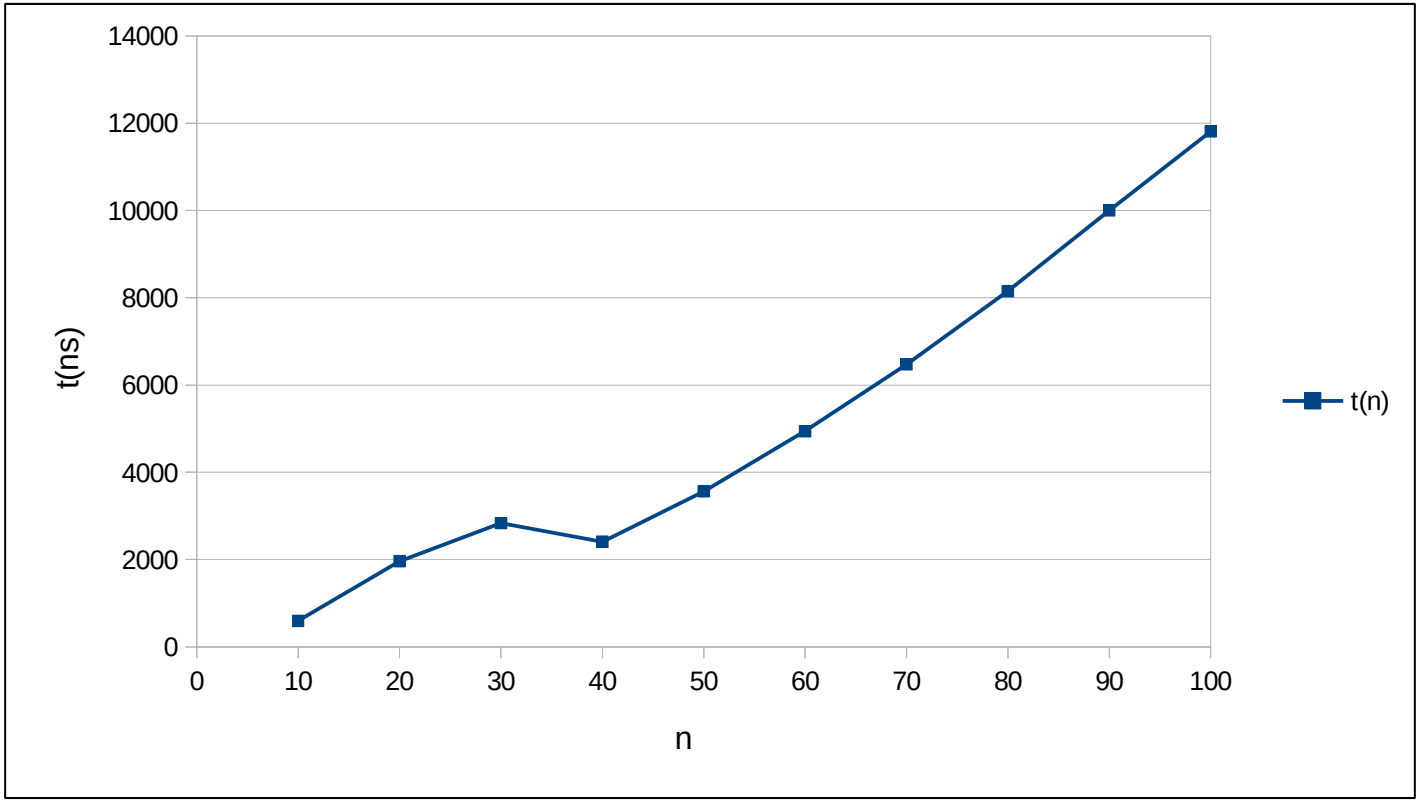
Вычислительный эксперимент

Для проверки асимптотических оценок проведем вычислительный эксперимент.

Возьмем алгоритм сортировки с известной оценкой асимптотической сложности. Для примера возьмем алгоритм сортировки пузырьком (будет изучаться в этом курсе в дальнейшем). Сложность алгоритма сортировки пузырьком равна $O(n^2)$.

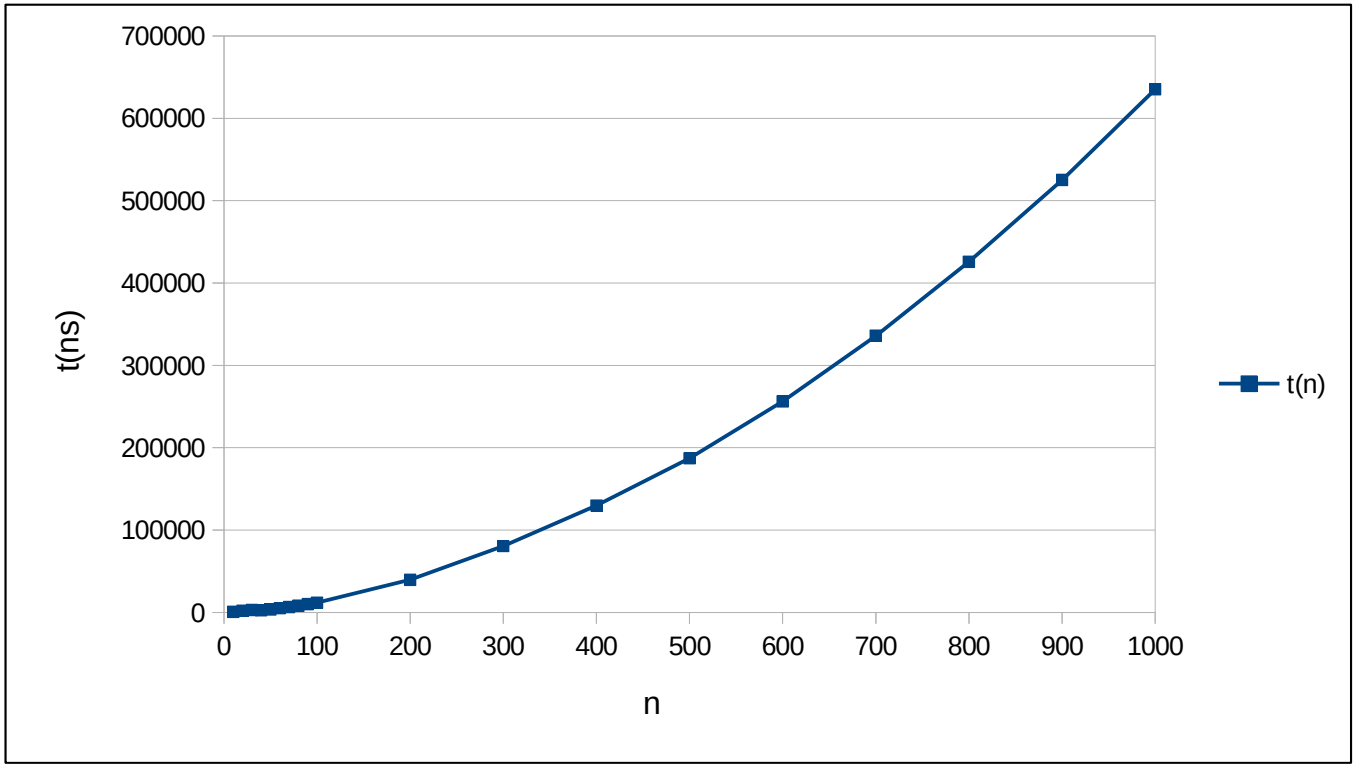
Замерим время его сортировки для массивов разного размера и проследим асимптотическое поведение. Для того что бы нивелировать влияние случайных колебаний каждый замер будем повторять много кратно и использовать усредненное значение.

Результаты вычислительного эксперимента



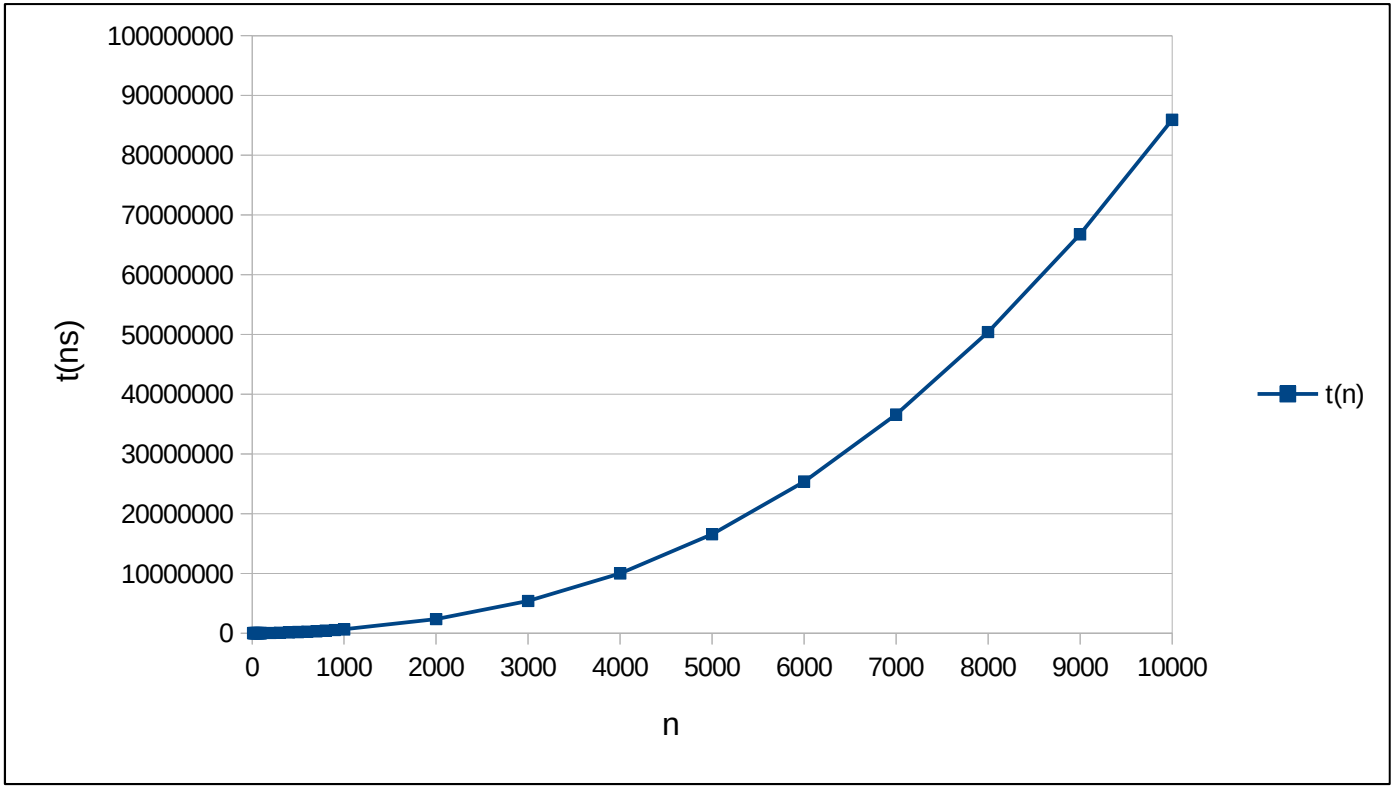
Для малых значений n время ведет себя отлично от параболы.

Результаты вычислительного эксперимента



При увеличении значения n поведение функции t(n) приближается к параболе

Результаты вычислительного эксперимента



Чем больше n тем ближе поведение t(n) к своей асимптотической оценке

Результаты вычислительного эксперимента

Как видно из графиков при увеличении объема входных данных поведение t(n) приближается к поведению асимптотической оценки сложности этого алгоритма.

Список литературы

1) Лафоре Р. Структуры данных и алгоритмы в Java. Классика Computers Science. 2-е изд. —СПб.: Питер, 2013. — 704 с.:ISBN 978-5-496-00740-5. Стр.[166-181]