Эффективность алгоритмов

Алгори́тм — точный набор инструкций, описывающих порядок действий исполнителя для достижения результата решения задачи за конечное время. В старой трактовке вместо слова «порядок» использовалось слово «последовательность», но по мере развития параллельности в работе компьютеров слово «последовательность» стали заменять более общим словом «порядок».

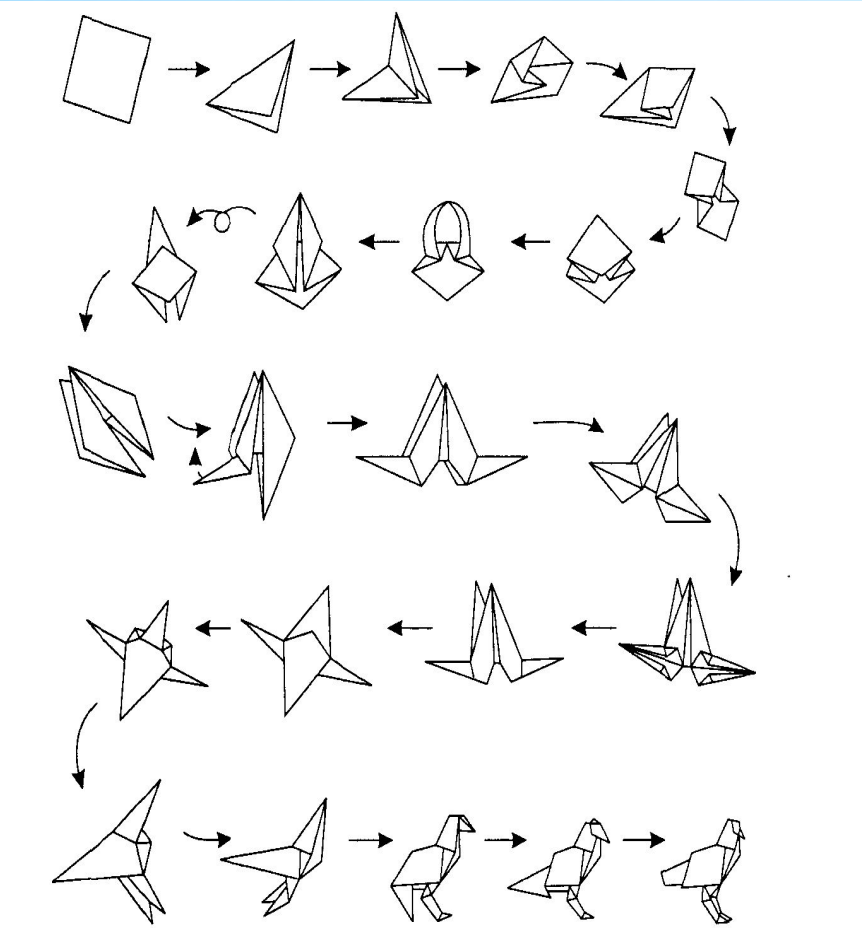


Рис. 4.1. Сложение птицы из квадратного листа бумаги

Это связано с тем, что работа каких-то инструкций алгоритма может быть зависима от других инструкций или результатов их работы.

Таким образом, некоторые инструкции должны выполняться строго после завершения работы инструкций, от которых они зависят. Независимые инструкции или инструкции, ставшие независимыми из-за завершения работы инструкций, от которых они зависят, могут выполняться в произвольном порядке, параллельно или одновременно, если это позволяют используемые процессор и операционная система.

Анализ алгоритмов – совокупность действий, в результате которых оцениваются какие-либо параметры алгоритма (эффективность, сложность, правильность, трудоемкость,…).
Основным практическим применением результатов подобных исследований является оценка относительных достоинств альтернативных алгоритмов.

Временная эффективность алгоритма обычно выражается как функция размера входных данных *n* (в ряде алгоритмов для оценки размера входных данных может использоваться несколько параметров одновременно — например, в алгоритмах для работы с графами такими параметрами являются количество вершин и количество ребер графа). В большинстве случаев выбор такого параметра не составляет труда. Например, для задач сортировки, поиска, нахождения наименьшего элемента и многих других алгоритмов обработки списков таким параметром является размер списка. При вычислении значения многочлена степени *n* $p(x\}=a_n x^n$ +L +a_0 таким параметром может быть степень многочлена или количество его коэффициентов, которое на единицу больше степени многочлена. Подобные небольшие отличия не влияют на результаты анализа эффективности алгоритма.

Поскольку конкретные временные характеристики алгоритма зависят от его реализации, использованного компилятора и компьютера, для оценки эффективности алгоритмов применяется такая характеристика, как асимптотическая зависимость количества базовых операций от размера входных данных при очень больших величинах последнего. Следует учесть, что для разных входных данных количество базовых операций может весьма существенно различаться.

Имеется еще один вид эффективности — так называемая амортизированная эффективность, когда рассматриваются не конкретные операции, а их последовательности. Возможны ситуации, когда конкретная операция над структурой данных занимает длительное время, но совокупность операций занимает меньше времени, чем сумма времен выполнения в наихудшем случае.

Для того чтобы можно было классифицировать и сравнивать между собой порядки роста, введены три условных обозначения

$$\Theta\ \Omega\ O$$

Ниже через t(n) обозначено время выполнения некоторого алгоритма (выражающееся как количество базовых операций); *g(n)* — не некоторая простая функция, с которой будет проводиться сравнение количества операций t(n).

При анализе поведения функции трудоемкости алгоритма часто используют принятые в математике асимптотические обозначения, позволяющие показать скорость роста функции, маскируя при этом конкретные коэффициенты.

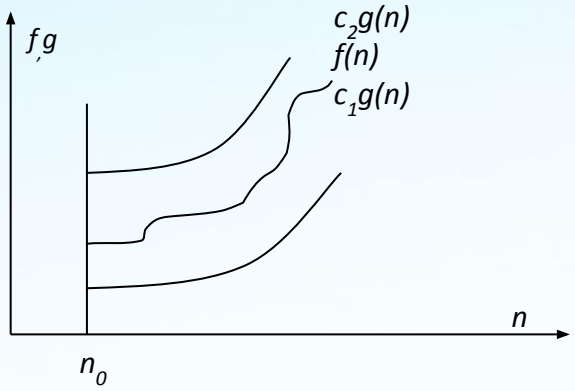
Такая оценка функции трудоемкости алгоритма называется *сложностью алгоритма* и позволяет определить предпочтения в использовании того или иного алгоритма для больших значений размерности исходных данных. В асимптотическом анализе приняты следующие обозначения:

Оценка Θ (тетта). *Пусть* $f(n)$ и $g(n)$ – положительные функции положительного аргумента, $n \geq 1$ (количество объектов на входе и количество операций – положительные числа), тогда:

$f(n) = \Theta(g(n))$,

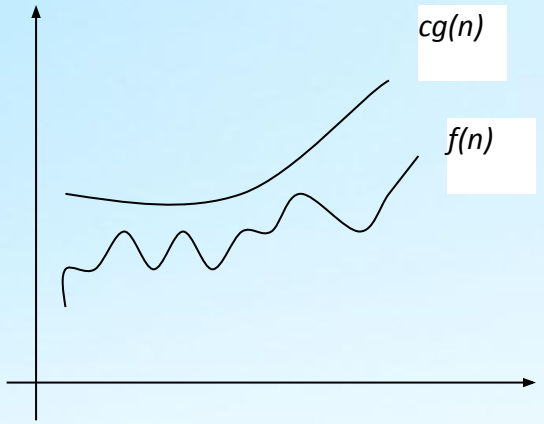
если существуют положительные c_1, c_2, n_0, такие, что:

$$c_1 * g(n) \leq f(n) \leq c_2 * g(n), \text{ при } n > n_0$$



Оценка *O* (*O* большое). В отличие от оценки *Θ*, оценка *O* требует только, что бы функция *f(n)* не превышала *g(n)* начиная с *n* > *n0*, с точностью до постоянного множителя:

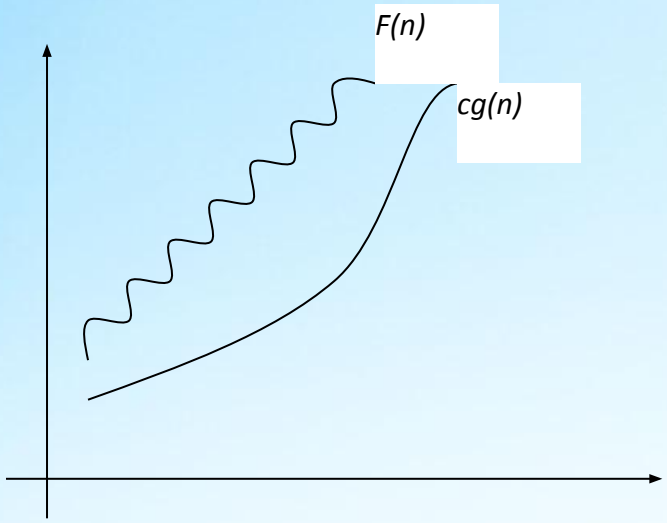
$$\exists\ c > 0,\ n0 > 0 :$$
$$0 \le f(n) \le c * g(n),\ \forall n > n0$$



Вообще, запись *O(g(n))* обозначает класс функций, таких, что все они растут не быстрее, чем функция *g(n)* с точностью до постоянного множителя, поэтому иногда говорят, что *g(n)* мажорирует функцию *f(n).*

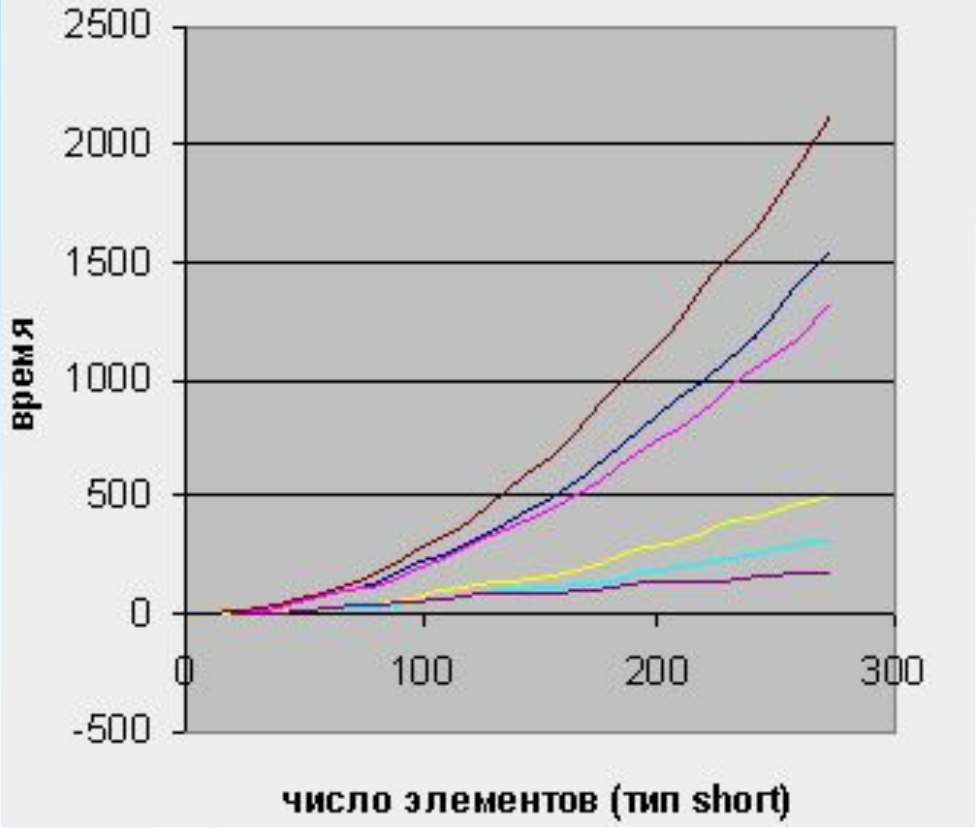
Оценка Ω (Омега). *В отличие от оценки O*, оценка Ω является оценкой снизу – т.е. определяет класс функций, которые растут не медленнее, чем *g(n)* с точностью до постоянного множителя:

$$\exists\ c > 0,\ n0 > 0 :$$
$$0 \le c * g(n) \le f(n)$$



Рассмотрим важность эффективности алгоритмов на графике, изображенном ниже:

- коричневая линия: сортировка пузырьком;
- синяя линия: шейкер-сортировка;
- розовая линия: сортировка выбором;
- желтая линия: сортировка вставками;
- голубая линия: сортировка вставками со сторожевым элементом;
- фиолетовая линия: сортировка Шелла.



Трудоёмкость алгоритма. Под трудоёмкостью алгоритма для данного конкретного входа – $F_a(N)$, будем понимать количество «элементарных» операций совершаемых алгоритмом для решения конкретной проблемы в данной формальной системе.

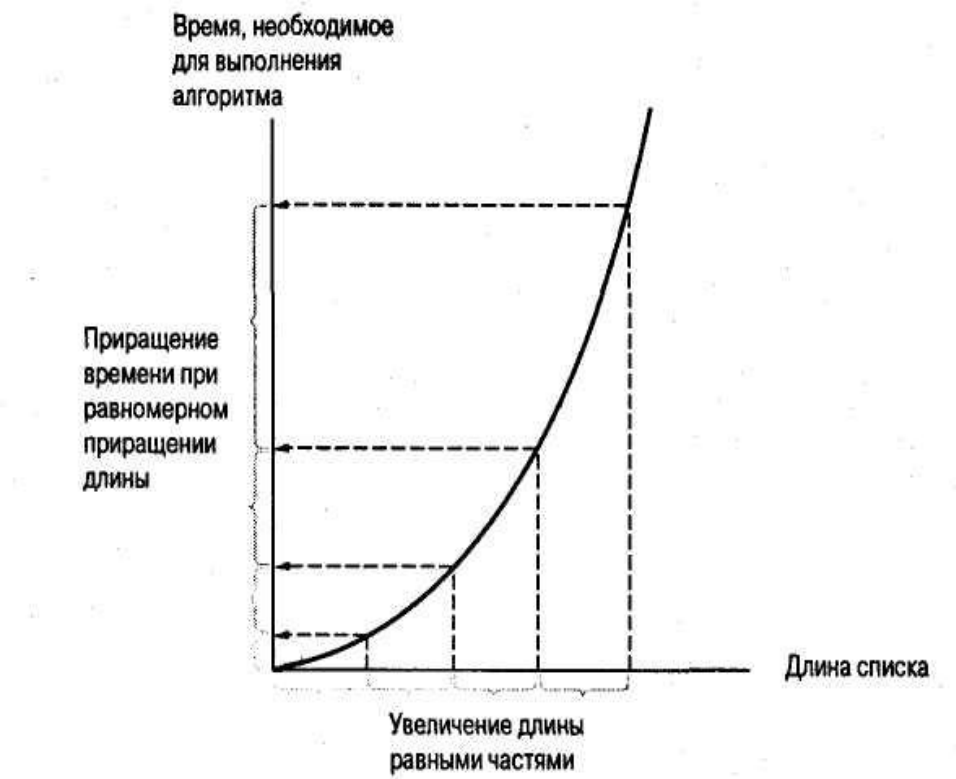
Комплексный анализ алгоритма может быть выполнен на основе комплексной оценки ресурсов формальной системы, требуемых алгоритмом для решения конкретных проблем. Очевидно, что для различных областей применения веса ресурсов будут различны, что приводит к следующей комплексной оценке алгоритма:

$$\psi_A = c_1 * F_a(N) + c_2 * M + c_3 * S_d + c_4 * S_r,$$

где c_i – веса ресурсов, M – *количество машинных инструкций,* S_r – память для организации вычислительного процесса (память, необходимая для реализации рекурсивных вызовов и возвратов), S_d – память для хранения промежуточных результатов.

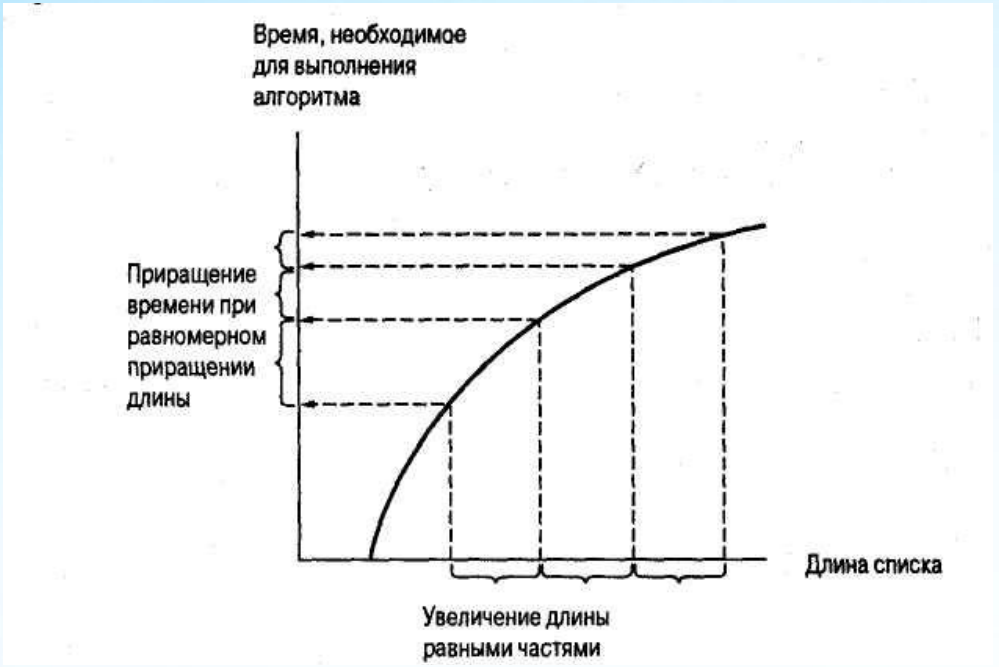
Проанализируем алгоритм сортировки методом вставки списка из n элементов. В лучшем случае потребуется n-1 сравнений, для выполнения задачи, а в худшем n(n-1)/2 сравнений. С помощью тестов можно построить график, который будет характеризовать данный алгоритм.

Из графика видно, что при увеличении длины списка на одно и то же количество элементов время, необходимое для сортировки списка, все больше и больше возрастает. Таким образом, с увеличением длины списка эффективность данного алгоритма уменьшается.



При использовании алгоритма двоичного поиска из n элементов потребуется проанализировать не более $\log_2$ и элементов. Это позволяет оценить время, необходимое для выполнения алгоритма при различной длине сортируемого списка. На рисунке представлен график, построенный по результатам данного анализа. Видно, что темпы роста времени выполнения алгоритма снижаются по мере увеличения длины списка, т.е. эффективность алгоритма двоичного поиска возрастает с увеличением

длины списка.



Верификация – это процесс определения, выполняют ли программные средства и их компоненты требования, наложенные на них в последовательных этапах жизненного цикла разрабатываемой программной системы.

Основная цель верификации состоит в подтверждении того, что программное обеспечение соответствует требованиям. Дополнительной целью является выявление и регистрация дефектов и ошибок, которые внесены во время разработки или модификации программы.

Верификация является неотъемлемой частью работ при коллективной разработке программных систем. При этом в задачи верификации включается контроль результатов одних разработчиков при передаче их в качестве исходных данных другим разработчикам. Для повышения эффективности использования человеческих ресурсов при разработке, верификация должна быть тесно интегрирована с процессами проектирования, разработки и сопровождения программной системы.