Лекция 3.2. Эффективность алгоритмов.

Анализ любого алгоритма должен дать четкое представление:

1. ёмкостной
2. временной сложности процесса.

Емкостная сложность (пространственная) (space efficiency) - это размер памяти, в которой предстоит размещать все данные, участвующие в вычислительном процессе (входные наборы данных, промежуточная и выходная информация).

Возможно, не все перечисленные наборы требуют одновременного хранения, - значит, удается сэкономить.

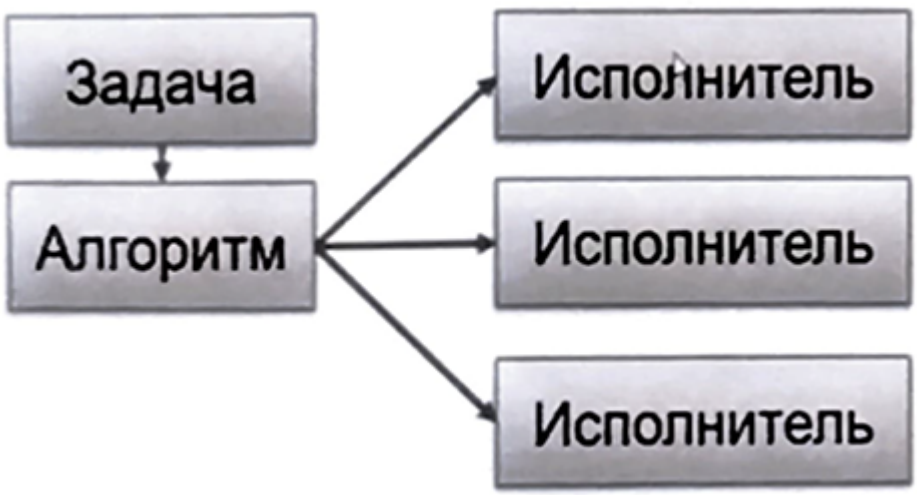
В ряде случаев, оценка емкостной сложности становится менее очевидной, например при использовании динамических структур.

Временная сложность (time efficiency) - время работы программы

Проведем анализ временной трудоемкости.

Поставлена некоторая задача, и для ее решения спроектирован алгоритм. Он описывает вычислительный процесс, который завершается за конечное число действий-шагов..

Анализ временной трудоемкости



Реальное время выполнения каждого шага алгоритма зависит от конкретного вычислительного устройства.

Т.е., неотъемлемым участником вычислительного процесса, - не алгоритма! - является исполнитель.
Имея ввиду предполагаемого исполнителя, лучше заранее оценить его вычислительные способности.

На эффективность алгоритмов влияет:

- производительность вычислительной системы (набор элементарных, инструкций системы)
- представление самих данных
- язык программирования (набор элементарных инструкций языка).

Возможны следующие варианты:

- либо алгоритм явно предписывает выполнять арифметико-логические операции, - и такой уровень программирования рассчитан непосредственно на работу процессора
- либо используются «укрупненные» инструкции, и для их обработки применяется специальный язык.

При оценке быстродействия алгоритма учитывают:

- Поведение вычислительного процесса в «среднем»
- Работа в «экстремальных» условиях

Моделирование «худших» случаев всегда связано с содержанием самого алгоритма. Возможные варианты:

- Проверка поведения алгоритма на входных данных, принимающих граничные значения из разрешенного диапазона
- Тестирование алгоритма на максимально больших по объему входных наборах (что важно для анализа как временной, так и емкостной эффективности).

Умение предвидеть «нехорошие» ситуации отличает квалифицированного алгоритмиста от обыкновенного кодировщика.

В связи с расширением сферы производства программного обеспечения сформировалась самостоятельная специализация - «тестеры программ».

Эффективность алгоритма оценивается:

- временем его работы, или временной сложностью (time efficiency)
- и объемом памяти, требуемой для его выполнения, или емкостной (пространственной) (space efficiency) сложностью.

Время выполнения зависит

- от порядка следования элементов и часто определяется размером входа.
- Размером входа может быть:
- число элементов на входе (сортировка, преобразование Фурье);
- количество байт памяти для представления данных;

При анализе алгоритмов интерес представляет максимальный размер входа.

Порядком некоторой функции $F(n)$

- (при достаточно больших n) называют другую функцию $G(n)$, такую, что:
- $\lim_{n\to\infty} \frac{F_n}{G_n} = const \neq 0$
- Обозначение: $F(n) = O[G(n)]$ (O-нотация)

Например, $F(n) = 2n^4 - 3n^3 + 4n^2 - 5$ порядком является n^4 или $F(n) = \mathrm{O}(n^4)$

Время работы

- оценивается числом элементарных шагов (операций) для каждой строки алгоритма.
- При вызове подпрограмм оценивается время их исполнения.
- Т. о., для каждого конкретного алгоритма мы можем приблизительно подсчитать время его выполнения.

Для C++ на Pentium

#	Операторы C++	Обозначение	Количсество тактов
1	`a+b`, `a>b`, `a>>b` и т.д.	t_+	2
2	`a*b`	t_*	20
3	`a/b`	$t_/$	28
4	`t++`, `a+`, `a > const` и Т.Д.	t_	1
5	`*p`	t_p	1

#	Операторы C++	Обозначение	Количсество тактов
6	`a[]`	t_	2
7	`if (...) P1 else P2`	t_	t_{усл} + 1 + P_1 t_{ветвь1}+ P_2 t_
8	`for() //n раз`	t_	$t_{\text{усл}} + t_{\text{пров}} + 2 + n(t_{\text{тела}} + t_{\text{модиф.}} + t_{\text{пров}} + 1)$

Микроконтроллеры AVR семейства Tiny и Mega фирмы ‘ATMEL’

№	Команда	Кол-во машинных циклов (тактов)
1	Условный переход	Результат True - 2 или 3 Результат False - 1
2	Безусловный переход	2
3	Прерывание	Max 4
4	Вызов подпрограмм	3 (2 из кот. -сохран. в стеке 2 байта сч. команд)
5	Возврат из п/программ и обработки прерываний	4
6	Логические операции	1
7	Арифметические операции и сдвиг	1
8	Умножение и обработка 2-х байтовых значений	9
9	Сложение с константой	2

Если алгоритм не содержит вложенных циклов, то функция времени - будет какая-то линейная функция от n: T (п) = (п), где п - размер входа и тип цикла не влияет на сложность алгоритма.

Если алгоритм использует вложенные циклы, то это будет квадратичная функция T(п2) = (п2),

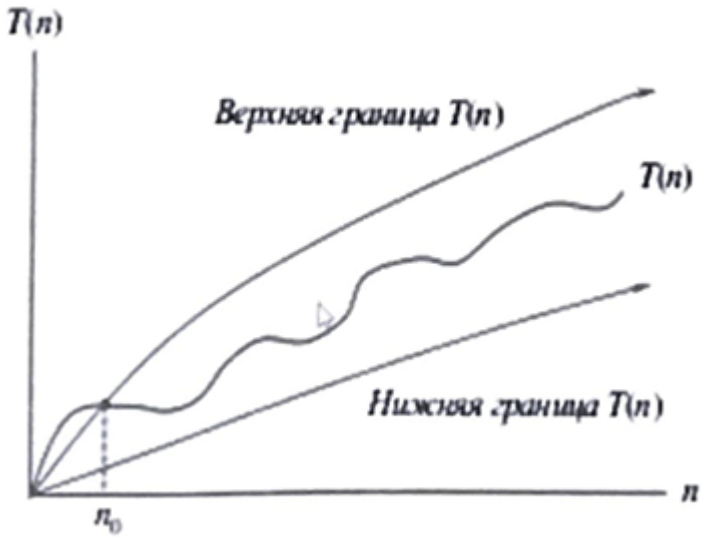
Т. е., вложенность повторений является основным фактором усложнения алгоритмов.

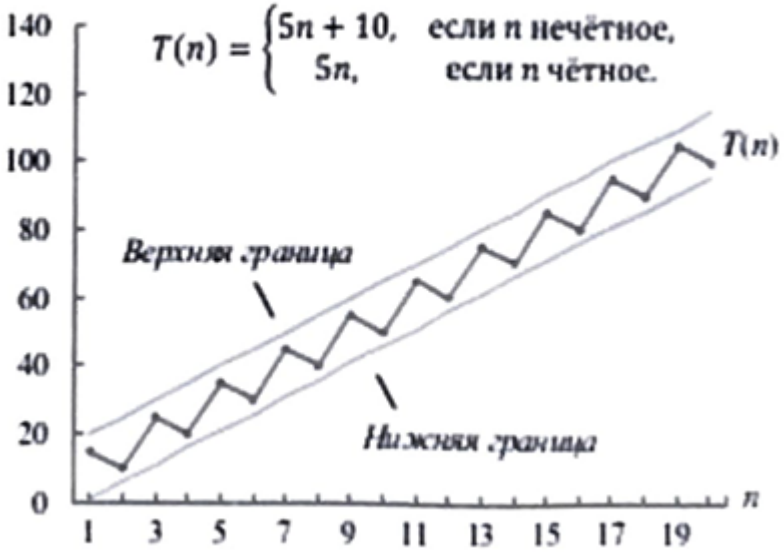
Рекурсивность алгоритма обычно более затратна по времени и памяти, чем вложенные циклы.

В общем случае, анализируя алгоритмы, можно не подсчитывать общее количество выполняемых операций, а проанализировать применяемые управляющие структуры и оценить асимптотику роста времени работы алгоритма при стремлении п в бесконечность.

Асимптотические обозначения

- Как правило, функция времени Г(л) выполнения алгоритма имеет большое количество локальных экстремумов - неровный график с выпуклостями и впадинами
- Проще работать с верхней и нижней оценками (границами) времени выполнения алгоритма





В теории вычислительной сложности алгоритмов для указания границ функции используют асимптотические обозначения:

- O (O большое)
  - Функция f(n) ограничена сверху функцией g(n) (порядком ф-ции f(n)) с точностью до постоянного множителя
- Ω (омега большое)
  - Функция f(n) ограничена снизу функцией g(n) с точностью до постоянного множителя
- Θ(тета большое)
  - Функция f(n) ограничена снизу и сверху функцией g(n) с точностью до постоянного множителя