**Алгоритмы и структуры данных.**

Лекция 1.

**Алгоритм** – корректно определенная вычислительная процедура, принимающая на вход и выдающая на выход данные определенного вида.

**Вычислительная процедура** – последовательность формальных манипуляций с данными.

*Корректный алгоритм для любых допустимых входных данных выполняется конечное время и выдает корректные выходные данные.*

*Корректные выходные данные должны удовлетворять требованиям, поставленным в решаемой задаче.*

**Структура данных** – способ организации данных, корректно поддерживающий определенный набор операций.

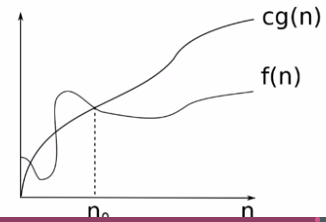
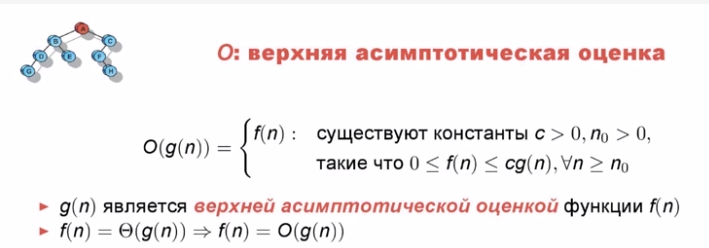
Зачем нужны эффективные алгоритмы? – Ресурсы компьютера ограничены (время работы, память)   
- Маршрутизация пакетов (поиск кратчайших путей)   
- Фильтрация спама (фильтры домена)   
- Базы данных, строковое алгоритмы  
- Сборка генома и тд.

Сложность алгоритмов, пример:

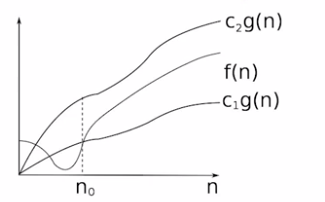
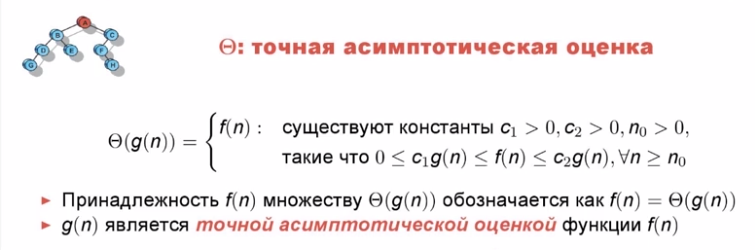
|  |  |  |
| --- | --- | --- |
| Сортировка | Вставками | Слиянием |
| Сложность | ~c1 \* n2 | ~c2 \* n\*log2n |
| От чего зависят c1, c2 – От реализации. На каком языке. | Ассемблер, c1 = 2 | JS, c2 = 100 |
| Вычислитель | Суперкомпьютер | Смартфон |
| Скорость | 1011 операций в секунду | 108 |
| Время | 2\*1016/1011 = 2\*105с | 100\*108\*27/108 = 2700с |

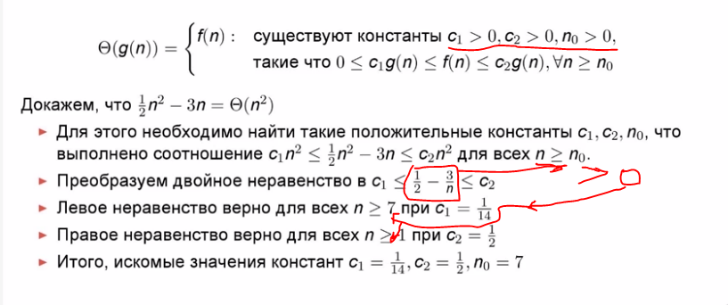
Оценка эффективности работы алгоритмов.

*Асимптотическая обозначения*:

О(g(n)) – Верхняя асимптотическая оценка  
  
т.е. существует такое значение n0, при котором функция f(n) меньше функции cg(n). (см. график)

Чаще всего используется для оценки времени работы алгоритмов. Свойства:   
Пусть f1(n) = О(g1(n)), f2(n) = О(g2(n)), тогда верно следующее:   
- f1(n) + f2(n) = О(max(g1(n), g2(n)))   
- f1(n) \* f2(n) = О(g1(n) \* g2(n))   
- c\*f1(n) = O(f1(n))   
- c + f1(n) = O(f1(n))

Ω(g(n)) - Нижняя асимптотическая оценка  
   
Есть некоторая функция f(n), которая, начиная с некоторого n0 больше функции cg(n). (см. график)   
  
θ(g(n)) - точная асимптотическая оценка.   
Выбирается максимальное g(n) для оценки функции. Т.е. функция с максимальной степенью или самая большая.  


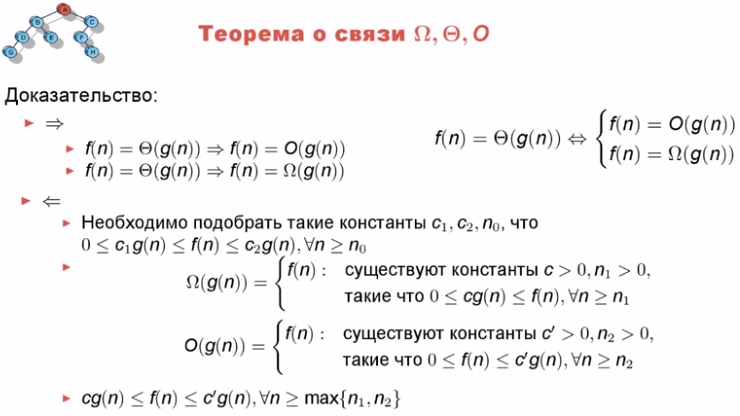


Время работы алгоритма: T(n), где n – целое число, размер входных данных.   
Входные данные можгут быть одним или несколькими числами.  
Пример Algoritm A(int n):   
1. j <- 1  
2. for i=1, to n do  
3. j <- j\*i

Инициализация переменной j выполняется за постоянное время с1  
Увелечение счетчика, проверка условия цикла, время на обновления переменной j в теле цикла обозначим, как с2  
Т.о T(n) = c1 + c2 \* n, т. к. 3 строка выполняется n раз

Асимтотическая оценка времени работы алгоритма: T(n) = θ(n)

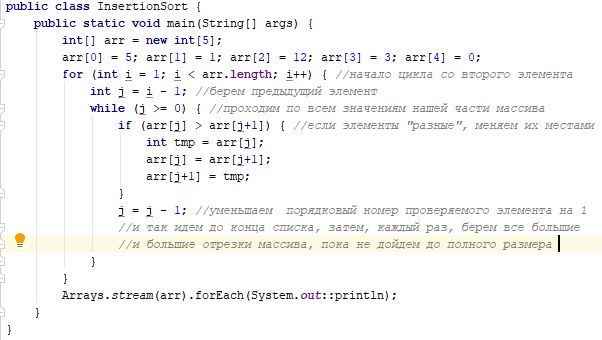
Теорема связи асимптотических оценок:

  
Т.е. если мы имеем θ(g(n)), то получается мы априори имеем верхнюю О(g(n)) и нижнюю Ω(g(n)) границы функции.

Алгоритм сортировки вставками.

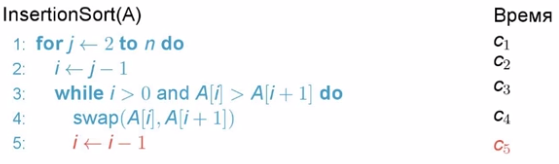
Задача сортировки чисел по возрастанию.   
На вход подается последоватьность чисел a1, …, anВыход: a’1, …, a’n , где a’1 <= a’2 <= a’3 …Пример:  
Вход – 10, 8, 2, 12, 5, 100  
Выход – 2, 5, 8, 10, 12, 100

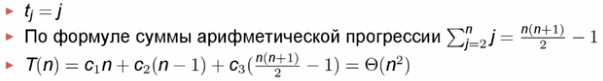
Можно сортировать все, что угодно, не только цифры.

Алгоритм сортировки вставками:  
Начинаем рассматривать числа со второго элемента, его индекс хранится в переменной j. В переменной i хранится индекс предыдущего элемента. (Процедура swap – это обмен двух переменных.) Проверяем условие A[i] > A[i+1] и если верно, то меняем местами элементы. Затем мы уменьшаем i, чтобы пройтись по предыдущим значениям и сравнить их. И так до момента i > 0. Потом возвращаемся к циклу for и у нас меняются j и i. Каждую итерацию мы сравниваем текущий элемент с предшествующим и если он меньше, то меняем их местами.  
<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->  
5 2 4 6 1 3  
InsertionSort(A)  
1. for j = 2 to n do  
2. i = j – 1   
3. while i > 0 and A[i] > A[i+1] do  
4. swap(A[i], A[i+1])  
5. i = i-1  
Пример на java:  


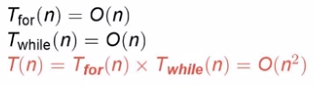
<!-- \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* -->

Оценка времени работы алгоритма.

Точная оценка:   
Время однократного выполнения каждой строки псевдокода – будем считать, что арифметические операции, занесения значения в память, проверка условий выполняются за фиксированное время c.  
Для однократного выполнения каждой строки необходимо:   
  
Но каждая строка выполняется не по одному разу, а несколько, в зависимости от некоторых условий. Число проверок условия цикла выполняется на одно больше, чем у нас значений, так как последняя проверка поймет, что мы вышли за границы и больше не пойдем в цикл (n раз). Тело цикла выполняется n – 1 раз. Количество выполнений цикла while обозначим за tj, т.о. он выполнится . Строки 4 и 5 псевдокода будет выполняться :

Для суммарного времени выполнения алгоритма достаточно умножить «сколько выполняется одна строка на число раз выполнения этой строки и сложить данные величины».   
  
Но чему же равно tj? Это будет зависеть от входных данных. Рассмотрим разные случаи:  
Лучший случай: элементы уже отсортированы в нужном порядке. Если так, то каждый последующий элемент нужно сравнить с предшествующим и убедиться в корректности.  
  
Худший случай: Элементы входной последовательности отсортированы в порядке, обратном требуемому. Т.е. каждый элемент будет сравниваться со всеми предшествующими ему элементу, чтобы поставить в начало массива. И время tj мы можем вычислить по формуле арифметической прогрессии.  
  
Константы и функции, которые растут медленнее, чем самая быстрорастущая функция можно отбросить.   
В общем случае алгоритм будет работать не хуже, чем верхняя граница (худший случай) О(n2).

Верхняя оценка времени работы алгоритмов:  
Самый удобный способ оценки работы алгоритма.   
Время работы цикла for - O(n). Для цикла while оно такое же, так как в этом цикле происходит сравнение текущего элемента с предшествующими, а предшествующих элементов не может быть больше, чем n. И по свойству асимптотических умножений, так как цикл wile запускается при каждой итерации цикла for, общее время – их произведение:

.

Лекция 2.