Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Реферат

По дисциплине «Основы алгоритмизации и программирования»

На тему «Нотации для оценки сложности алгоритмов»

 Выполнила:

Студентка1 курса 6 группы

Альшевская Алина Михайловна

Преподаватель: доц Белодед Н.И

2023, Минск

Нотации для оценки сложности алгоритмов

Вычисли́тельная сло́жность — понятие в информатике и теории алгоритмов, обозначающее функцию зависимости объёма работы, которая выполняется некоторым алгоритмом, от размера входных данных.

Обычно у алгоритмов бывает две сложности:

* *временная сложность* — как количество операций, которые выполняются при работе алгоритма, связано с объёмом входных данных;
* *сложность по памяти* — как количество памяти, которое нужно алгоритму, связано с размером входных данных.

В обоих случаях оцениваем, как связаны используемые алгоритмом ресурсы (время или память) с количеством входных данных. Может показаться, что алгоритм медленнее работает и потребляет больше памяти, когда размер входных данных большой. Это не всегда так.

Есть несколько способов оценки сложности алгоритмов. Их основная идея – получить ограничение для функции, которая связывает размер входных данных и количество операций или размер памяти. Не стоит определять эту функцию точно, нам нужна именно оценка.

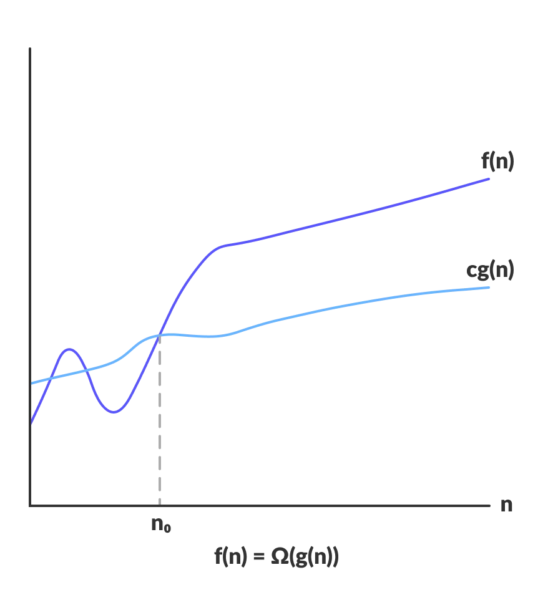
Теперь разберём некоторые способы оценки сложности алгоритма.

**O (О-большое)**

*O*, читается как «О», «О-большое» или «биг (big) О», описывает оценку сложности сверху. То есть максимальное количество операций, которое алгоритм может выполнить в худшем случае. В скобках после О указывают функцию, которая ограничивает сложность.

**Ω (сигма)**

*Ω*, читается как «Сигма» или «Сигма-большая», описывает оценку сложности снизу. То есть минимальное количество операций, которое алгоритм будет выполнять в лучшем случае. В скобках после Ω указывают функцию, которая ограничивает сложность. Например Ω(n) означает, что сложность растёт так же или быстрее, чем линейно. Например, квадратичная сложность n × n — это тоже Ω(n).



Ω(g(n)) = { f(n): существуют положительные константы c и n0

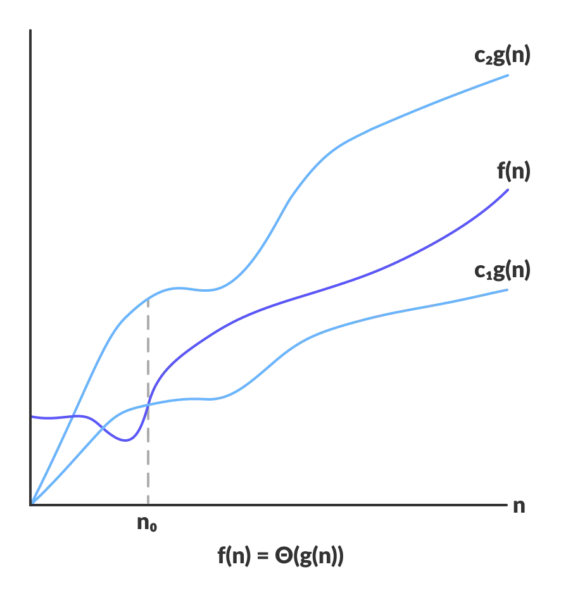
такие что 0 ≤ cg(n) ≤ f(n) для всех all n ≥ n0 }

Это выражение может быть описано как функция f(n) принадлежит множеству Ω(g(n)), если существует положительная константа c, такая что функция находится выше cg(n) для достаточно больших значений n.

Для любого значения n минимальное время, необходимое для выполнения алгоритма, определяется нижней границей Omega Ω(g(n)).

**Θ (тета)**

Θ, читается как «Тета» или «Тета-большая», описывает плотную оценку алгоритма. В скобках после ϴ указывают функцию, которая ограничивает сложность как сверху, так и снизу. Поскольку она представляет верхнюю и нижнюю границы времени выполнения алгоритма, то используется для анализа среднего случая сложности алгоритма.



Для функции g(n), Θ(g(n)) определяется следующим образом:

|  |  |
| --- | --- |
|  | Θ(g(n)) = { f(n): существуют положительные константы c1, c2 и n0  такие что 0 ≤ c1g(n) ≤ f(n) ≤ c2g(n) для всех n ≥ n0 } |

Следующее выражение может быть описано как функция f(n) принадлежит множеству Θ(g(n)), если существуют положительные константы c1 и c2, такие что функция находится между c1g(n) и c2g(n) при достаточно больших n.

Если функция f(n) находится где-то между c1g(n) и c2g(n) для всех n ≥ n0, то говорят, что f(n) имеет асимптотически точную границу.

Рассмотрим на примере

#include <iostream>

#include <vector>

void printPairs(const std::vector<int>& arr) {// функция, которая печатает все пары элементов в массиве

for (int i = 0; i < arr.size(); i++) {

for (int j = 0; j < arr.size(); j++) {

std::cout << "(" << arr[i] << ", " << arr[j] << ")" << std::endl;

}

}

}

int main() {

std::vector<int> arr = { 1, 2, 3, 4, 5 };

printPairs(arr);

return 0;

}

Проанализируем сложность этого кода:

* Большое O (O): В данном случае, мы проходим по массиву дважды, поэтому сложность равна O(n2), где n- это размер массива.
* Сигма (Ω): Это нижняя граница времени выполнения. В нашем случае, даже в лучшем случае, нам все равно придется пройти по всему массиву дважды, чтобы напечатать все пары. Поэтому сложность также равна Ω(n2)
* Тета (Θ): Это среднее время выполнения. Поскольку в нашем случае верхняя и нижняя границы совпадают, среднее время выполнения также равно Θ(n2)

Таким образом, сложность этой функции равна O(n2), Ω(n2) и Θ(n2)