БУДУЩЕЕ
В НАШИХ
РУКАХ

Алгоритмы

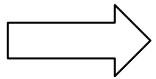
Lecture 6

Гончаров Марк, Шумаков Иван

Оценка сложности алгоритма

- Алгоритмы могут быть оценены без привязки к конкретной архитектуре
- Представим абстрактную машину, у которой:
  - Инструкции исполняются за неизменяемое константное время
  - Random Access Memory (RAM)
  - Есть только стандартные инструкции и типы данных

```
while (i < n) {
    i += 2;      <- t1
    Arr[i] = i;  <- t2
}
```

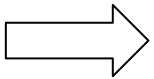


Время исполнения:
(t1 + t2) * n / 2

Оценка сложности алгоритма

- При исполнении на разных машинах t1, t2 будут отличаться. Значение n – нет
- Скорость работы алгоритма – **асимптотическая** зависимость от входных данных
- Самый распространенный способ оценки это O-нотация

```
while (i < n) {
    i += 2;      <- t1
    Arr[i] = i;  <- t2
}
```

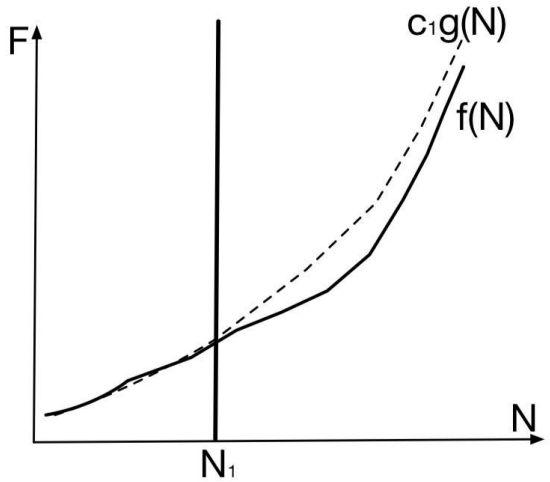


Время исполнения:
(t1 + t2) * n / 2
O-нотация:
O(n)

O-нотация

- Формальное определение:

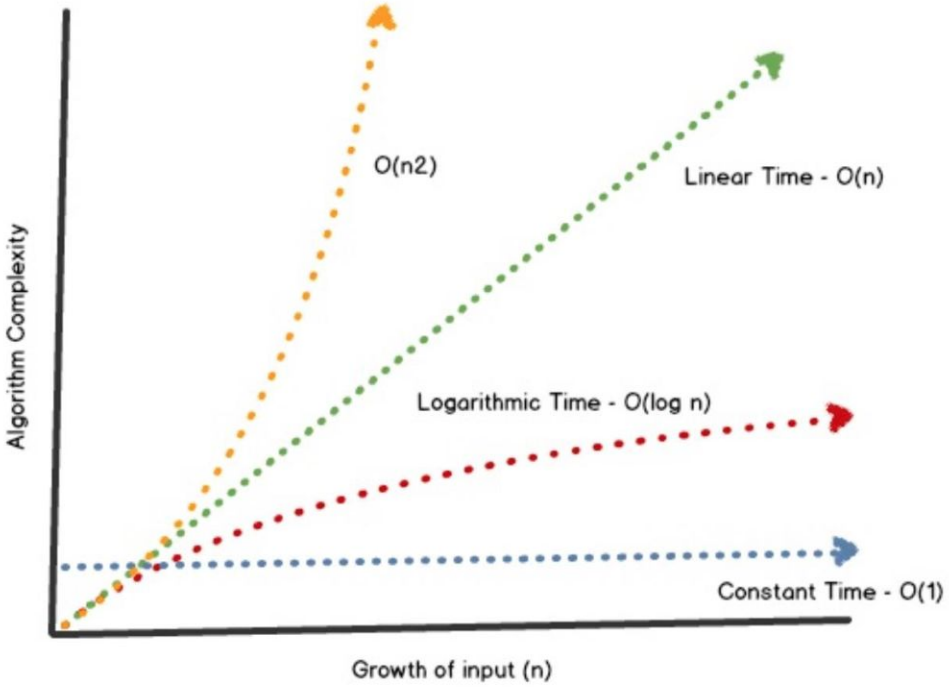
$$f(n) = O(g(n)) \Leftrightarrow \exists k, M : \forall n > k \mapsto Mg(n) \geq |f(n)|$$



- Примеры:
  - 10 n -> O(n)
  - log(n) + n -> O(n)
  - n + 1 -> O(n) -> O(n * n)

O-нотация

	n	$nlog(n)$	n^2	2^n
10	1	1	1	1
50	1	1	1	13д
10^6	1	1	15м	∞
10^{10}	10с	2м	3г	∞
10^{14}	2ч	28ч	∞	∞



Виды асимптотики

- Необходимо добавить элемент в конец массива. Какая сложность у этого действия?

Виды асимптотики

- Необходимо добавить элемент в конец массива. Какая сложность у этого действия?
- Добавление в выделенную память $O(1)$

Виды асимптотики

- Необходимо добавить элемент в конец массива длиной n.
- Какая сложность у этого действия?
  - Добавление в выделенную память $O(1)$
  - Перевыделение + добавление $O(n)$
- Какая сложность у k добавлений?

Виды асимптотики

- Необходимо добавить элемент в конец массива длиной n.
- Какая сложность у этого действия?
  - Добавление в выделенную память O(1)
  - Перевыделение + добавление O(n)
- Какая сложность у k добавлений?

$$\sum_{i=1}^{k}(n+i) = \sum_{i=1}^{k} n + \sum_{i=1}^{k} i = n * k + 1/2 * k * (k+1)$$

- Можно ли сделать лучше?
  - Да, если каждый раз выделять в 2 раза больше памяти

$$\sum_{i=1}^{k} 1 + \sum_{i=1}^{log_2(k/n)} 2^i = k + \frac{2(2^{log_2(k/n)} - 1)}{2-1} = k + 2\frac{k}{n} - 2$$

Декомпозиция

- Многие задачи имеют рекурсивную структуру: для решения основной задачи необходимо решить аналогичные на меньших данных
- Для определения подзадачи необходимо найти инварианты
- Время работы такого алгоритма может быть выражена формулой:
  - a – количество подзадач
  - n/b – размер подзадачи
  - O(n^d) – сложность консолидации

$$T(n) = aT\left(\left\lceil \frac{n}{b} \right\rceil\right) + O(n^d)$$

$$T(n) = \begin{cases} O(n^d), & \text{если } d > \log_b a, \\ O(n^d \log n), & \text{если } d = \log_b a, \\ O(n^{\log_b a}), & \text{если } d < \log_b a. \end{cases}$$

Задачи

- На вход подается последовательность чисел в диапазоне от 1 до 20000, разделенных пробелами и заканчивающаяся нулем — ограничителем последовательности, не входящим в множество. Все эти элементы образуют множество A, то есть, среди нет одинаковых пар. После разделителя подаются элементы множества B, тоже завершающиеся нулём, не входящим в множество. На выход программы нужно вывести симметрическую разность множеств A и B в порядке возрастания элементов.

Задачи

- Множество задано строкой, то есть каждая буква есть элемент множества.
- Но это множество — не совсем простое. Элементы в нём могут повторяться.
- Два подмножества считаются одинаковыми, если все элементы одного множества совпадают с элементами другого. Например, множества, представленные строками abc и cba совпадают. Совпадают также множества abra и raba.
- Ваша задача по заданной строке, представляющей исходное множество, вывести все различные его подмножества, каждое на отдельной строке вывода