Лекция 1

Алгоритмы

Характеристики алгоритма. Сложность алгоритмов. Асимптотики.

Алгоритм (неформально) – это

Набор четко сформулированных правил, в сущности, рецепт для решения некоторой вычислительной задачи.

Примеры:

- отсортировать множество чисел;
- по имеющейся дорожной карте вычислить кратчайший путь от некоторой исходной точки до места назначения;
- выполнить несколько задач до наступления установленных сроков, при этом необходимо упорядочить эти задачи так, чтобы завершить их все вовремя

Зачем изучать алгоритмы?

Важность для всех отраслей *computer science*

Понимание основ алгоритмизации и организации структур данных необходимо для серьезной практической работы в любой сфере информатики.

- Алгоритмы – двигатель технологических инноваций.

  Закон Мура: каждый год компьютеры за счет увеличения плотности транзисторов становятся быстрее в 1,6 раз. Во многих областях прирост производительности за счет улучшения алгоритмов превышает прирост производительности за счет увеличения скорости процессов.

  Алгоритмы позволяют по новому взглянуть на области за пределами *computer science* и *IT* (квантовые вычисления, колебания цен на экономических рынках и т.д.)

- Алгоритмы – гимнастика для мозга.
- Разработка алгоритмов доставляет удовольствие.

Характеристики алгоритма

- Входные данные → Выходные данные

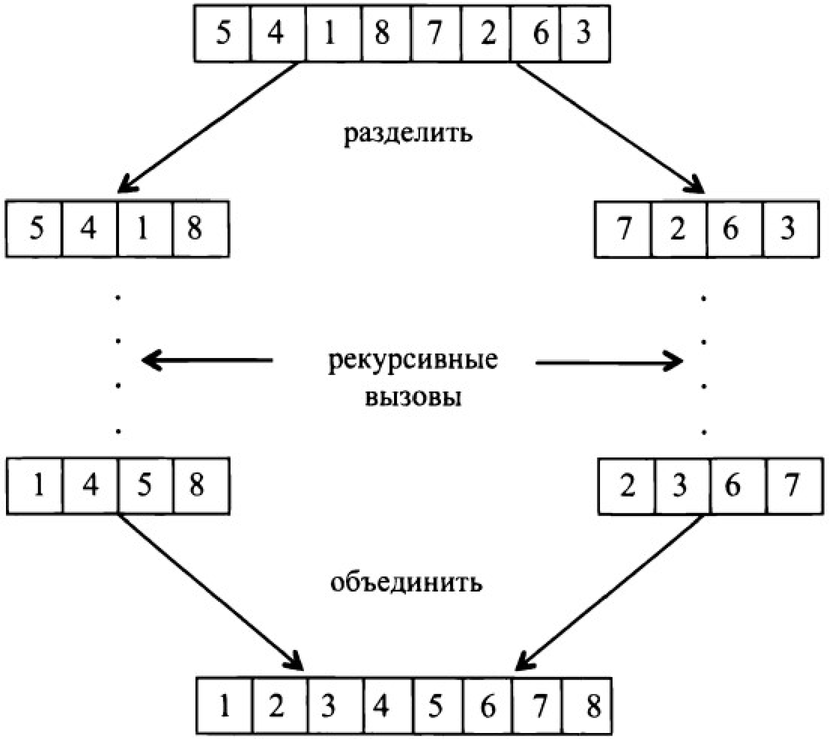
(конечные последовательности) (конечные последовательности)

- Алгоритм описывается конечной последовательностью символов (словом)
- Алгоритм применим ко всевозможным входам
- Пошаговое выполнение (детерминированная последовательность конфигураций, оканчивающаяся завершающей конфигурацией - ответом)
- Для некоторых входов выход не определен (не приводит к завершающей конфигурации.

Вычислительные ресурсы

- По вычислению можно посчитать разные задействованные ресурсы
- Экономия на одном из ресурсов может обернуться дополнительными расходами другого
- 3 главных ресурса:
  - Время (все алгоритмы выполняются пошагово. Количество элементарных шагов – время работы)
  - Память (пространство для промежуточных расчетов. Количество занятых единиц используемой памяти)
  - Случайность. Для некоторых задач известны рандомизированные алгоритмы. Ресурс - число используемых случайных битов (могут стоить дорого)

Анализ временной сложности работы алгоритма (пример). *MergeSort* - сортировка слиянием



Разделяй и властвуй

MergeSort (псевдокод)

MERGESORT

Вход: массив A из n разных целых чисел.

Выход: массив с теми же самыми целыми числами, отсортированными от наименьшего до наибольшего.

```
// базовые случаи проигнорированы
```

C := рекурсивно отсортировать первую половину A

D := рекурсивно отсортировать вторую половину A

вернуть `Merge` (C, D)

Подпрограмма *Merge* (слияние)

MERGE

Вход: отсортированные массивы C и D (длиной $n/2$ каждый).

Выход: отсортированный массив B (длиной n).

Упрощающее допущение: n — четное.

```
i := 1
j := 1
for k := 1 to n do
    if C[i] < D[j] then
        B[k] := C[i]        // заполнить выходной массив
        i := i + 1                        // прирастить i
    else                                  // D[j] < C[i]
        B[k] := D[j]
        j := j + 1
```

Анализ алгоритма *Merge*

```
1 i := 1
2 j := 1
3 for k := 1 to n do
4     if C[i] < D[j] then
5         B[k] := C[i]
6         i := i + 1
7     else
8         B[k] := D[j]
9         j := j + 1
```

Пусть *Merge* работает с двумя отсортированными массивами длиной $l/2$ каждый.

Строки 1, 2 → 2 операции

Цикл *for* l раз. Внутри цикла:

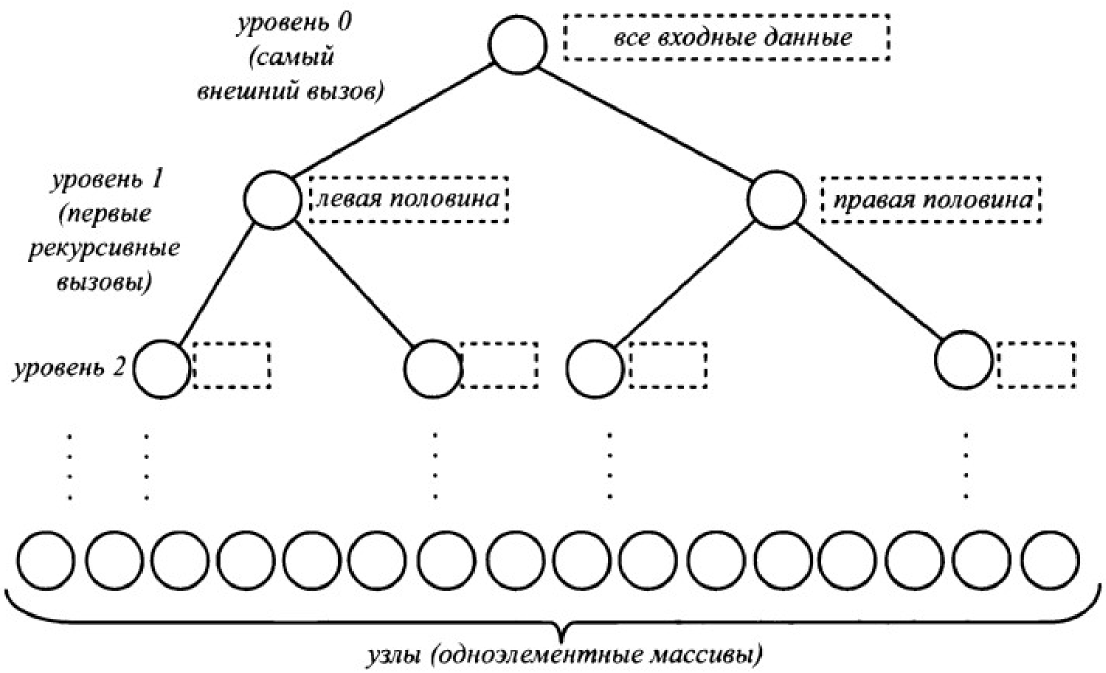
- Строка 4 – 1 сравнение, строка 5 (8) – 1 присваивание → (2 операции)
- Строка 6 (9) – 1 приращение → 1 операция
- k увеличивается на 1 → 1 операция

Итого $4l + 2$ операций. При $l \geq 1$ $4l + 2 \leq 6l.$ Возьмем $6l$ в качестве допустимой верхней границы для *Merge.*

Теорема (предел времени исполнения алгоритма *MergeSort).*

Для каждого входного массива длиной *n≥1* алгоритм *MergeSort* выполняет не более $\boxed{6n \cdot log_2 n + 6n}$ операций.

Доказательство.



1 задача

2 подзадачи

4 подзадачи

j=0,1,…,k –уровней

n подзадач

$n = 2^k \Rightarrow m = log_2 n$+1 количество уровней вместе с нулевым

$$\underbrace{\text{Количество подзадач уровня } j}_{=2^j} \times \underbrace{\text{работа в расчете на подзадачу уровня } j}_{=\frac{6n}{2^j}}$$

на j-м уровне рекурсии выполняется не более

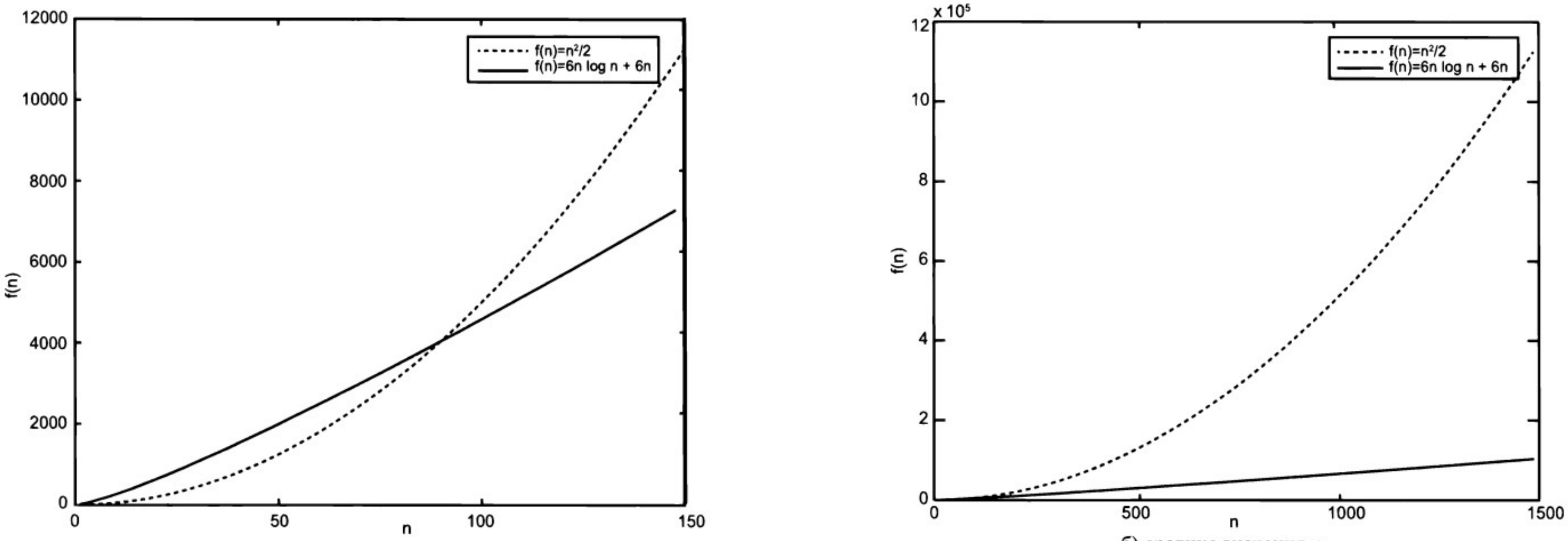
$$2^j \times \frac{6n}{2^j} = 6n \text{ операций}$$

$$\boxed{\underbrace{\text{количество уровней}}_{=\log_2 n+1} \times \underbrace{\text{работа в расчете на уровень}}_{\le 6n} \le 6n \log_2 n + 6n.}$$

Ч.т.д.

Сравнение времени работы двух алгоритмов сортировки (асимптотический анализ)

Пусть есть другой алгоритм сортировки, который выполняет не более $\frac{n^2}{2}$ операций. $6n \cdot log_2 n + 6n \leqslant \frac{n^2}{2}$?



а) малые значения *n*

б) средние значения *n*

MergeSort работает быстрее при достаточно *больших размерах исходных данных*.

О-символика

Пусть даны функции *f(n)* и *g(n)*, значениями которых являются положительные действительные числа. Говорят, что $f = O(g)$ (*f* растет не быстрее, чем *g*), если существуют такая константа c>0 и $n_0 \in \mathrm{N}$, что $f(n) \le c \cdot g(n) \;\; \forall n > n_0$. (Или $\frac{f(n)}{g(n)} \le c$).

(Говорят $f \le g$ с точностью до константы).

Если $f_1(n) = \frac{n^2}{2}, f_2(n) = 6n \cdot log_2 n + 6n$, то $f_2(n) = O(\, f_1(n))$

$$\frac{f_2(n)}{f_1(n)} = \frac{6n \cdot log_2 n + 6n}{\frac{n^2}{2}} = \frac{12}{n} log_2 n + \frac{12}{n} \le 24$$

Наоборот, $\frac{f_1(n)}{f_2(n)} > \frac{n}{12}$ - неограниченно

Еще пример:

$$f_1(n) = n^2, f_2(n) = 2n + 20$$

$$\frac{f_2(n)}{f_1(n)} = \frac{2n+20}{n^2} = \frac{2}{n} + \frac{20}{n^2} \leq 22 \text{ ограниченно}$$

$f_2(n) = O(f_1(n))$, но

$f_1(n) \neq O(f_2(n))$, т.к.

$\frac{n^2}{2n+20} \geq \frac{n^2}{22n} \geq \frac{n}{22}$ неограниченно $(f_1(n) = \Omega(f_2(n)))$

Пусть есть алгоритм

$f_3(n) = n + 1$

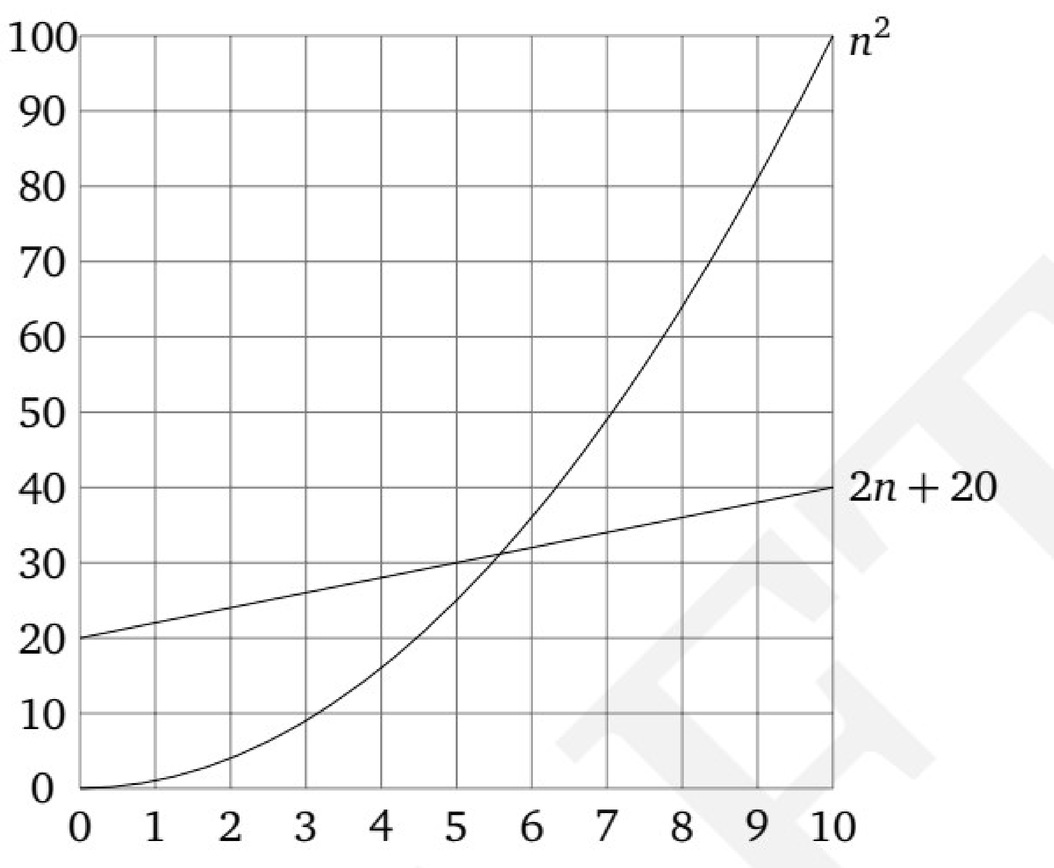
$f_2(n) = O(f_3(n))$ и $f_3(n) = O(f_2(n))$ т.к.

$\frac{f_2(n)}{f_3(n)} = \frac{2n+20}{n+1} \leq 20$ и $\frac{f_3(n)}{f_2(n)} = \frac{n+1}{2n+20} \leq 1$

f_2 и f_3 имеют одинаковый порядок роста $\Rightarrow f_2 = \Theta(f_3)$

Можно так: если $f = O(g)$ и $f = \Omega(g)$ то $f = \Theta(g)$

Сравнение квадратичной и линейной функций



О-большое для полиномов

Утверждение. Пусть $T(n) = a_k n^k + \dots a_1 n + a_0,$

где $k \geq 0$ — это неотрицательное целое число и a_i — вещественные числа (положительные или отрицательные). Тогда $T(n) = O(n^k)$.

Доказательство.

$$T(n) \leq |a_k| n^k + \dots + |a_1| n + |a_0|.$$

При $n \geq 1$ $n^k \geq n^i$ для $i = 1, 2, \dots, k$

$$T(n) \leq |a_k| n^k + \dots + |a_1| n^k + |a_0| n^k = \underbrace{\left(|a_k| + \dots + |a_1| + |a_0|\right)}_{=c} \times n^k.$$

Это неравенство справедливо для каждого $n \geq n_0 = 1$, то есть то, что мы и хотели доказать. *Ч. т. д.*

O-большое для экспоненты

Утверждение. Пусть $T(n) = 2^{n+10}$,

то тогда $T(n) = O(2^n)$.

Доказательство.

$$T(n) = 2^{n+10} = 2^{10} \times 2^n = 1024 \times 2^n.$$

Возьмем c=1024
ч.т.д

Утверждение. Пусть $T(n) = 2^{10n}$,

тогда $T(n)$ *не является* $O(2^n)$.

Доказательство. Допустим обратное:
$T(n) = O(2^n)$. Тогда $\exists\, c > 0$ и $n_0 \geq 1$: $2^{10n} \leq c \times 2^n$
для $n > n_0$. Следовательно, $2^{9n} \leq c$
неверно!

Благодаря O-символике

- Можно пренебречь константами и слагаемыми более низкого порядка. (Для *MergeSort* временная сложность *T(n)=O(nlogn)*)
- n^α растет быстрее n^β при $\alpha > \beta$
- Экспонента растет быстрее любой степенной функции (a^n быстрее n^α)
- Полином быстрее логарифма (даже $\sqrt{n}$ быстрее *logn*)

$$O(n^2 + n) = O(n)$$
$$O(n + logn) = O(n)$$
$$O(5 \cdot 2^n + 10 \cdot n^{100}) = O(2^n)$$
$$O(n^2 + B) = O(n^2 + B)$$

$O(1) =$ постоянная работа, не зависит от размера исходных данных

Сложность последовательных и вложенных циклов

```
FOR  i:=1 TO n DO
    print A[i]
FOR  j:=1 TO m DO
    print B[j]
```

Последовательные действия - сложение

O(n+m)

```
FOR  i:=1 TO n DO
  FOR  j:=1 TO m DO
        print (A[i], B[j])
```

Для каждой из *n* итераций внешнего цикла выполняется *m* вложенного цикла – умножение

O(n·m)

Основные принципы анализа алгоритмов

Необходимо уравновесить точность с непротиворечивостью. Для этого принимаются некоторые допущения, позволяющие создать оценки, чтобы увидеть достаточно точную картину того, какие алгоритмы работают, как правило, быстрее других.

1. Принцип №1: анализ наихудшего случая (подходит для универсальных подпрограмм, предназначенных для работы в различных областях применения)

2. Принцип №2: анализ значимых деталей (не стоит слишком беспокоиться о малых коэффициентах или членах низших порядков). Принцип обеспечивает

- математическую простоту
- то, что постоянные коэффициенты зависят от реализации
- нет риска потерять в точности прогноза о результате.

3. Принцип №3: асимптотический анализ (смещение в сторону больших входных данных)