



# Алгоритмы и структуры данных

Лекция 1. Введение в алгоритмы.

(с) Глухих Михаил Игоревич, [glukhikh@mail.ru](mailto:glukhikh@mail.ru)

## Цель курса

- Познакомиться с основами анализа и оценки эффективности алгоритмов (структур)

## Цель курса

- Познакомиться с основами анализа и оценки эффективности алгоритмов (структур)
- Познакомиться с основными группами известных алгоритмов (структур)

## Цель курса

- Познакомиться с основами анализа и оценки эффективности алгоритмов (структур)
- Познакомиться с основными группами известных алгоритмов (структур)
- Научиться применять известные алгоритмы (структуры) при программировании на известном языке программирования (Java или C++)

## Цель курса

- Познакомиться с основами анализа и оценки эффективности алгоритмов (структур)
- Познакомиться с основными группами известных алгоритмов (структур)
- Научиться применять известные алгоритмы (структуры) при программировании на известном языке программирования (Java или C++)
- Научиться конструировать новые алгоритмы, применяя уже известные идеи

# Квалификация программиста = ?

- Разработка
  - Предельный объём + сложность программы

# Квалификация программиста = ?

- Разработка
  - Предельный объём + сложность программы
- Качество кода
  - Читаемость
  - Надёжность
  - Производительность

# Квалификация программиста = ?

- Разработка
  - Предельный объём + сложность программы
- Качество кода
  - Читаемость
  - Надёжность
  - Производительность
- Знания
  - Языки + Библиотеки
  - Приёмы разработки



# Квалификация программиста = ?

- Разработка
  - Предельный объём + сложность программы
- Качество кода
  - Читаемость
  - Надёжность
  - Производительность
- Знания
  - Языки + Библиотеки
  - Приёмы разработки
- Анализ
  - Умение разобраться в сложном коде / задаче

# Литература

- **Томас Кормен и др. Алгоритмы. Построение и анализ. 3-е издание**
- **Дональд Кнут. Искусство программирования**
- **Никлаус Вирт. Алгоритмы + Структуры данных = Программы**
- **Е. В. Пышкин. Структуры данных и алгоритмы: реализация на C++ (ссылка на странице курса)**
- **McDowell, G. L. Cracking the Coding Interview: 150 Programming Questions and Solutions**
- **<https://github.com/Kotlin-Polytech/JavaKotlinASD>**

# Must have!

## © Cracking the Coding Interview

- Data structures
  - Linked Lists
  - Binary Trees
  - Tries
  - Stacks / Queues
  - Array Lists
  - Hash Tables

# Must have!

## © Cracking the Coding Interview

- Algorithms
  - Breadth-First Search
  - Depth-First Search
  - Binary Search
  - Merge Sort
  - Quick Sort
  - Tree Insert / Find / ...

# Must have!

## © Cracking the Coding Interview

- Concepts
  - Bit manipulation
  - Singleton design pattern
  - Factory design pattern
  - Memory (Stack / Heap)
  - Recursion
  - Big O

# Краткое содержание курса

- Алгоритмы: Big O, рекуррентность, декомпозиция
- Вероятностные алгоритмы (???)
- Алгоритмы сортировки
- Структуры данных: динамическая память
- Структуры данных: таблицы и деревья
- Динамическое программирование
- NP-полнота

# Структуры и алгоритмы

- Структура = данные с определённой организацией
  - Иногда включающие наборы «Инвариантов»
  - Пример: сбалансированное бинарное дерево

# Структуры и алгоритмы

- Структура = данные с определённой организацией
  - Иногда включающие наборы «Инвариантов»
  - Пример: сбалансированное бинарное дерево
- Алгоритм = формальная операция над данными
  - (с определённой организацией)
  - Предусловия / постусловия
  - Сохранение инвариантов



## Пример алгоритма: сортировка

- Input = List<Comparable> in
- Output = List<Comparable> out
- Invariant = None
- Precondition = None
- Postcondition = for any i: out[i+1] >= out[i]

# Многообразие задач

- Интернет

- Оптимальные маршруты перемещения данных
- Поиск страниц с нужной информацией

# Многообразие задач

- Интернет
  - Оптимальные маршруты перемещения данных
  - Поиск страниц с нужной информацией
- Электронная коммерция
  - Шифрование с открытым ключом
  - Цифровая подпись

# Многообразие задач

- Интернет
  - Оптимальные маршруты перемещения данных
  - Поиск страниц с нужной информацией
- Электронная коммерция
  - Шифрование с открытым ключом
  - Цифровая подпись
- Производство / коммерция
  - Оптимизация выгоды в условиях различных ограничений

# Многообразие алгоритмов

- Поиск кратчайшего пути и задача коммивояжёра
- Поиск длиннейшей общей подпоследовательности
- Поиск зависимостей между модулями проекта
- Преобразование Фурье
- Умножение матриц
- Случайная перестановка элементов
- Составление расписаний
- Игра в шахматы
- Распараллеливание

# Корректность алгоритма

- $\text{Output} = \text{Alg}(\text{Input})$
- Математически
  - ЕСЛИ  $\text{Precondition}(\text{Input})$ , ТО  $\text{Postcondition}(\text{Alg}(\text{Input}))$
  - ЕСЛИ  $\text{Invariant}(\text{Input})$ , ТО  $\text{Invariant}(\text{Alg}(\text{Input}))$
- Как доказать корректность?

## Пример: сортировка вставками

- Код на Java в примере lesson1.Sorts
- Псевдокод

```
for j = 1 to list.size - 1:  
    key = list[j]  
    i = j - 1  
    while i >= 0 && list[i] > key:  
        list[i+1] = list[i]  
        i--  
    list[i+1] = key
```

# Сортировка вставками: принцип рекуррентности

- Рекуррентный (индуктивный) алгоритм
  - База: 1 элемент всегда упорядочен



# Сортировка вставками: принцип рекуррентности

- Рекуррентный (индуктивный) алгоритм
  - База: 1 элемент всегда упорядочен
  - Переход: от  $j-1$  упорядочены к  $j$  упорядочены
    - Пусть  $j-1$  элементов уже упорядочено
    - Берём элемент номер  $j$
    - Находим среди  $j-1$  элементов такую пару  $A[i]$  и  $A[i+1]$ ,  
что  $A[i] \leq A[j] \leq A[i+1]$ ;  
либо, если её нет, то одно из двух:  $A[j] \leq A[0]$  или  $A[j-1] \leq A[j]$
    - Вставляем элемент номер  $j$  между этой парой

# Сортировка вставками: принцип рекуррентности

- Рекуррентный (индуктивный) алгоритм
  - База: 1 элемент всегда упорядочен
  - Переход: от  $j-1$  упорядочены к  $j$  упорядочены
    - Пусть  $j-1$  элементов уже упорядочено
    - Берём элемент номер  $j$
    - Находим среди  $j-1$  элементов такую пару  $A[i]$  и  $A[i+1]$ ,  
что  $A[i] \leq A[j] \leq A[i+1]$ ;  
либо, если её нет, то одно из двух:  $A[j] \leq A[0]$  или  $A[j-1] \leq A[j]$
    - Вставляем элемент номер  $j$  между этой парой
  - Результат: все  $N$  элементов упорядочены

## Сортировка вставками: $O(?)$

- ▶ Пусть размер списка  $N$   
for  $j = 1$  to  $\text{list.size} - 1$ :  
     $\text{key} = \text{list}[j]$   
     $i = j - 1$   
    while  $i \geq 0 \ \&\& \ \text{list}[i] > \text{key}$ :  
         $\text{list}[i+1] = \text{list}[i]$   
         $i--$   
     $\text{list}[i+1] = \text{key}$

## Сортировка вставками: $O(?)$

- Пусть размер списка  $N$

```
for j = 1 to list.size - 1:           // N-1
    key = list[j]                      // N-1
    i = j - 1                          // N-1
    while i >= 0 && list[i] > key:      // N-1
        list[i+1] = list[i]           //  $0..N(N-1)/2$ 
        i--
    list[i+1] = key                     // N-1
```

## Сортировка вставками: $O(?)$

- ▶ Пусть размер списка  $N$   
for  $j = 1$  to  $\text{list.size} - 1$ : //  $N-1$   
     $\text{key} = \text{list}[j]$  //  $N-1$   
     $i = j - 1$  //  $N-1$   
    while  $i \geq 0 \ \&\& \ \text{list}[i] > \text{key}$ : //  $N-1$   
         $\text{list}[i+1] = \text{list}[i]$  //  $0 \dots N(N-1)/2$   
         $i--$   
         $\text{list}[i+1] = \text{key}$  //  $N-1$
- ▶  $5(N-1) \dots 5(N-1) + N(N-1)/2$

## Затраты времени

- В наихудшем случае (обычно важнее всего)

## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)

## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)
- В наилучшем случае (как правило, неважно)



## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)
- В наилучшем случае (как правило, неважно)
- Каковы наилучший и наихудший случай для сортировки вставками?

## Затраты времени

- В наихудшем случае (обычно важнее всего)
- В среднем случае (важно, если наихудший случай происходит крайне редко)
- В наилучшем случае (как правило, неважно)
- Каковы наилучший и наихудший случай для сортировки вставками?
- Как разработать алгоритм с минимальными затратами в наилучшем случае?

# Интересные вопросы про $O(f(n))$

➤ Что это такое?

# Интересные вопросы про $O(f(n))$

- Что это такое?
- Верно ли, что:
  - $O(n) + O(n) = 2 * O(n)$
  - $O(n) + O(n) = O(2 * n)$
  - $O(n) + O(n) = O(n)$

# Интересные вопросы про $O(f(n))$

- Что это такое?
- Верно ли, что:
  - $O(n) + O(n) = 2 * O(n)$
  - $O(n) + O(n) = O(2 * n)$
  - $O(n) + O(n) = O(n)$
  - $O(n) * O(n) = O^2(n)$
  - $O(n) * O(n) = O(n^2)$
  - $O(n) * O(n) = O(n)$

## Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ

## Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ
- $f(n) = \Omega(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) > Cg(n)$ 
  - ОЦЕНКА СНИЗУ ~ РАСТЁТ БЫСТРЕЕ

## Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ
- $f(n) = \Omega(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) > Cg(n)$ 
  - ОЦЕНКА СНИЗУ ~ РАСТЁТ БЫСТРЕЕ
- $f(n) = \Theta(g(n))$ : одновременно  $O$  и  $\Omega$ 
  - ОЦЕНКА С ДВУХ СТОРОН ~ РАСТЁТ С ТОЙ ЖЕ СКОРОСТЬЮ



# Математика: варианты оценки затрат

- $f(n) = O(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) < Cg(n)$ 
  - ОЦЕНКА СВЕРХУ ~ РАСТЁТ МЕДЛЕННЕЕ
- $f(n) = \Omega(g(n))$ : существуют  $C$  и  $n_0$ : при  $n > n_0$   $f(n) > Cg(n)$ 
  - ОЦЕНКА СНИЗУ ~ РАСТЁТ БЫСТРЕЕ
- $f(n) = \Theta(g(n))$ : одновременно  $O$  и  $\Omega$ 
  - ОЦЕНКА С ДВУХ СТОРОН ~ РАСТЁТ С ТОЙ ЖЕ СКОРОСТЬЮ
- $f(n) = o(g(n))$ : для любого (сколь угодно малого) коэффициента  $C$  существует  $n_0$ : при  $n > n_0$ ...
  - РАСТЁТ ЗНАЧИТЕЛЬНО МЕДЛЕННЕЕ

# Математика: варианты оценки затрат

- Программисты, как правило, используют  $O$ 
  - Действительно, важна в первую очередь верхняя граница
  - Но найти её важно точно
    - и поэтому часто имеется в виду оценка с двух сторон ( $\Theta$ )

# Принцип декомпозиции

- Или “Разделяй и властвуй”
  - Рекурсивный подход
- Разделение
  - Задача делится на  $k$  меньших частей (часто на две)
- Властвование
  - Каждая из частей решается отдельно (если требуется, таким же образом, как и первая)
- Комбинирование
  - Затем результаты объединяются

# Сортировка слиянием: принцип

- Разделение
  - Делим  $n$  элементов на левую и правую половину
- Властвование
  - Упорядочиваем каждую половину отдельно
- Комбинирование
  - Составляем две половины вместе

## Пример: сортировка слиянием

- Код на Java в примере lesson1.Sorts
- Псевдокод комбинирования (left, right → list)

```
li = 0
ri = 0
for i = 0 to list.size - 1:
    if left[li] <= right[ri]: // ~~~ IOBE
        list[i] = left[li++]
    else:
        list[i] = right[ri++]
```

## Сортировка слиянием: $O(?)$

### ► Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
    m = (p + q) / 2                // 1
    sort(list, p, m)               // S(N/2)
    sort(list, m, q)               // S(N/2)
    merge(list, p, m, q)           // 5N
```

## Сортировка слиянием: $O(?)$

### ► Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
    m = (p + q) / 2                // 1
    sort(list, p, m)                // S(N/2)
    sort(list, m, q)                // S(N/2)
    merge(list, p, m, q)            // 5N
```

## Сортировка слиянием: $O(?)$

➤ Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
    m = (p + q) / 2                // 1
    sort(list, p, m)               // S(N/2)
    sort(list, m, q)               // S(N/2)
    merge(list, p, m, q)           // 5N
```

➤  $S(N) = 2S(N/2) + 5N + 1$

➤ Верно ли:  $O(N) = 2O(N/2) + 5N + 1 = O(N) + O(N) = O(N)$  ?



## Сортировка слиянием: $O(?)$

➤ Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
    m = (p + q) / 2                // 1
    sort(list, p, m)               // S(N/2)
    sort(list, m, q)               // S(N/2)
    merge(list, p, m, q)           // 5N
```

➤  $S(N) = 2S(N/2) + 5N + 1$

➤ Верно ли:  $O(N) = 2O(N/2) + 5N + 1 = O(N) + O(N) = O(N)$  ?

➤ А так:  $S(N) = 2S(N/2) + O(N)$  ?

## Сортировка слиянием: $O(?)$

➤ Псевдокод

```
sort(list, p = 0, q = list.size): // S(N)
    m = (p + q) / 2                // 1
    sort(list, p, m)                // S(N/2)
    sort(list, m, q)                // S(N/2)
    merge(list, p, m, q)            // 5N
```

➤  $S(N) = 2S(N/2) + 5N + 1$

➤ Верно ли:  $O(N) = 2O(N/2) + 5N + 1 = O(N) + O(N) = O(N)$  ?

➤ А так:  $S(N) = 2S(N/2) + O(N)$  ?

➤ Как насчёт наилучшего и наихудшего варианта?

## Как решать (проверять): принцип

- Рекуррентное соотношение

- $S(N) = 2S(N/2) + O(N)$

## Как решать (проверять): принцип

- Рекуррентное соотношение

- $S(N) = 2S(N/2) + O(N)$

- Предположим:  $S(N) = O(N) \sim C * N$

- $C * N = 2C * N/2 + D * N = C * N + D * N$

- НЕВЕРНО

## Как решать (проверять): принцип

- Рекуррентное соотношение

- $S(N) = 2S(N/2) + O(N)$

- Предположим:  $S(N) = O(N) \sim C * N$

- $C * N = 2C * N/2 + D * N = C * N + D * N$

- НЕВЕРНО

- Предположим:  $S(N) = O(N^2) \sim C * N^2$

- $C * N^2 = 2C * N^2/4 + D * N = C * N^2/2 + D * N$

- НЕВЕРНО

## Как решать (проверять): принцип

- Рекуррентное соотношение
  - $S(N) = 2S(N/2) + O(N)$
- Предположим:  $S(N) = O(N) \sim C * N$ 
  - $C * N = 2C * N/2 + D * N = C * N + D * N$
  - НЕВЕРНО
- Предположим:  $S(N) = O(N^2) \sim C * N^2$ 
  - $C * N^2 = 2C * N^2/4 + D * N = C * N^2/2 + D * N$
  - НЕВЕРНО
- Предположим:  $S(N) = O(N \lg N) \sim C * N \lg N$ 
  - $C * N \lg N = 2C * N \lg(N/2) / 2 + D * N = C * N \lg N - C * N \lg 2 + D * N$
  - ВЕРНО

## Максимальный подмассив

- Есть (известен) массив цен на акции `Price[]`
- Требуется выбрать два дня  $i, j$ , такие, что
  - $j > i$  (!!!)
  - $\text{Price}[j] - \text{Price}[i] \rightarrow \text{MAX}$
- То есть ВНАЧАЛЕ купить дешевле, ПОТОМ продать дорого
- Примеры
  - 100, 113, 110, 85, 105, 102, 86, 63, 81, 101, 94, 106, 101, 79, 94
  - 10, 11, 7, 10, 6

## Максимальный подмассив

- Есть (известен) массив цен на акции `Price[]`
- Требуется выбрать два дня  $i, j$ , такие, что
  - $j > i$  (!!!)
  - $\text{Price}[j] - \text{Price}[i] \rightarrow \text{MAX}$
- То есть ВНАЧАЛЕ купить дешево, ПОТОМ продать дорого
- Примеры
  - 100, 113, 110, 85, 105, 102, 86, 63, 81, 101, 94, 106, 101, 79, 94
  - 10, 11, 7, 10, 6



## Вариация

- Берём массив цен `Price[]`
- Формируем массив изменений `Delta[]` :  
 $\text{Delta}[i] = \text{Price}[i+1] - \text{Price}[i]$
- Требуется найти подмассив `Delta`, такой, что сумма его элементов максимальна
  - МАКСИМАЛЬНЫЙ ПОДМАССИВ

## Варианты решения

- В лоб
- Разделяй и властвуй
- Рекуррентный

# Варианты решения

- В лоб
  - Просто перебираем все варианты:  $O(N^2)$
- Разделяй и властвуй
  - Разбиваем массив делит на две половины
  - В каждой ищем максимальный подмассив
  - Сравниваем его с тем, который проходит через обе половины

# Варианты решения

- Разделяй и властвуй
  - Разбиваем массив дельт на две половины
  - В каждой ищем максимальный подмассив
  - Сравниваем его с тем, который проходит через обе половины
- Рекуррентный
  - Ищем максимальный подмассив для массива размером  $J-1$
  - Далее одно из двух
    - Либо он максимален и для размера  $J$
    - Либо для размера  $J$  максимален подмассив, включающий последний элемент

# Задача умножения матриц

- Упрощённый вариант: квадратные матрицы размером  $M \times M$ , причём  $M$  является степенью двойки
- Вариант “в лоб”
  - Вычисление каждого элемента произведения требует  $M$  умножений и  $M-1$  сложений
  - Элементов имеется  $M^2$
  - Общая трудоёмкость  $O(M^3)$
- Рекурсивный вариант
  - Делим каждую матрицу на четыре подматрицы
    - $A_{11} \ A_{12} \ B_{11} \ B_{12}$
    - $A_{21} \ A_{22} \ B_{21} \ B_{22}$
  - Перемножаем подматрицы отдельно, потом сливаем вместе
  - $S(M) = 8S(M/2) + O(M^2) \implies S(M) = O(M^3)$

## Метод Штрассена

- Идея: уменьшить число перемножений подматриц с 8 до 7, сведя рекуррентное соотношение к следующему
  - $S(M) = 7S(M/2) + O(M^2) \implies S(M) = O(M^{2.81})$
- С этой целью из 8 подматриц  $A_{ij} / B_{ij}$  формируется ещё 10 матриц  $S_k = A_{ij} + / - B_{mn}$ , вычисляется 7 произведений, суммируются результаты для формирования  $A \times B$
- Подробности см. книгу Кормена

# Итоги

- Рассмотрели
  - Рекуррентный и рекурсивный (разделяй и властвуй) подходы
  - Оценки эффективности
  - Наилучший / наихудший случай
  - Примеры применения на практике
- Далее
  - Алгоритмы сортировки