Лекция 3.
Алгоритмы сортировки и поиска.

План лекции

- Зачем нужны сортировки?
- Сортировка подсчётом
- Сортировка пузырьком
- Сортировка вставками
- Сортировка слиянием
- Сортировка выбором
- Быстрая сортировка

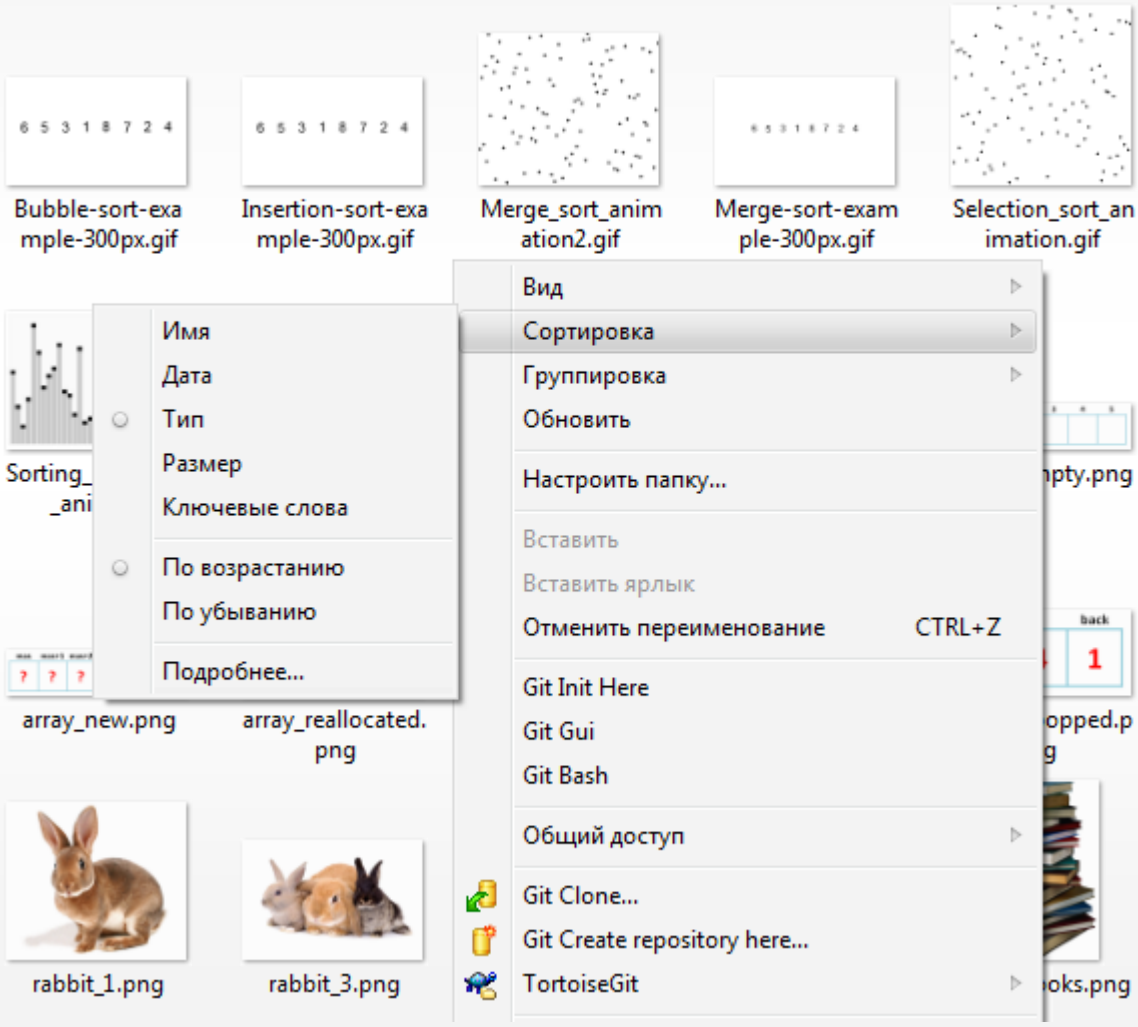
+ Сложность алгоритма

План лекции

- Зачем нужен поиск?
- Линейный поиск
- Бинарный (двоичный) поиск
- Тернарный (троичный) поиск

Зачем нужны сортировки?

Работать с упорядоченными данными гораздо удобнее, не так ли?

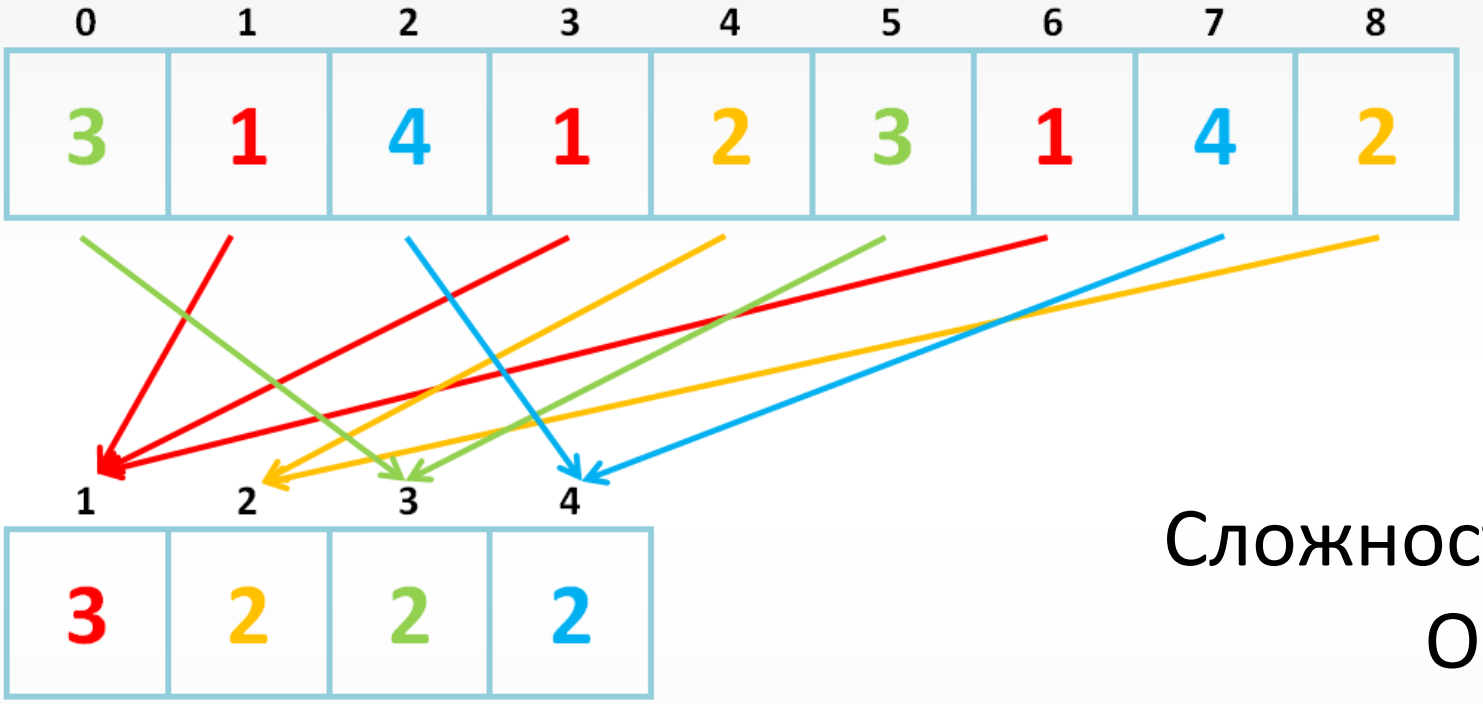


Сортировка подсчётом (Counting sort)

Самая простая по алгоритму сортировка, которая лучше всего работает, когда у нас достаточно большое количество одинаковых значений.

Смысл прост: посчитать, сколько каких значений имеется в последовательности, и вывести их по порядку именно столько раз.

Сортировка подсчётом



Сложность:
O(n)

0	1	2	3	4	5	6	7	8
1	1	1	2	2	3	3	4	4

Сложность алгоритма

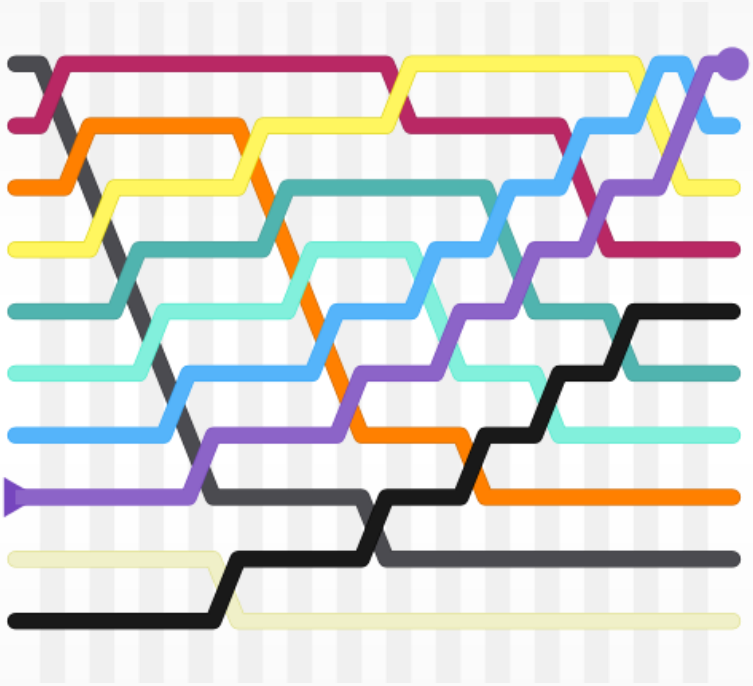
Вычислительная сложность алгоритма – это оценка объёма работы в зависимости от объёма входных данных.

Для чёткого описания сложности алгоритма используется асимптотическая сложность (асимптотика) – примерная оценка времени работы алгоритма.

Сложность алгоритма

Название	Запись	Время при n=100	Пример алгоритмов
Постоянное время (константа)	O(1)	1	Ввод/вывод числа, арифметическая операция
Логарифмическое время (логарифм)	O(log n)	7	Двоичный поиск
Линейное время (линия)	O(n)	100	Поиск минимума/максимума в неотсортированном массиве
Линейно-логарифмическое время (линия на логарифм)	O(n log n)	700	Быстрая сортировка, сортировка слиянием
Квадратичное время (квадрат)	$O(n^2)$	10 000	Сортировка пузырьком, сортировка вставками
Кубическое время (куб)	$O(n^3)$	1 000 000	Умножение матриц
Факториальное время (факториал)	O(n!)	933262154439441526816992388562667004907159682643816214685929638952175999932299156089461463976156518286253697920827223758251185210916864000000000000000000000	Решение задачи коммивояжёра полным перебором

Сортировка пузырьком (Bubble sort)



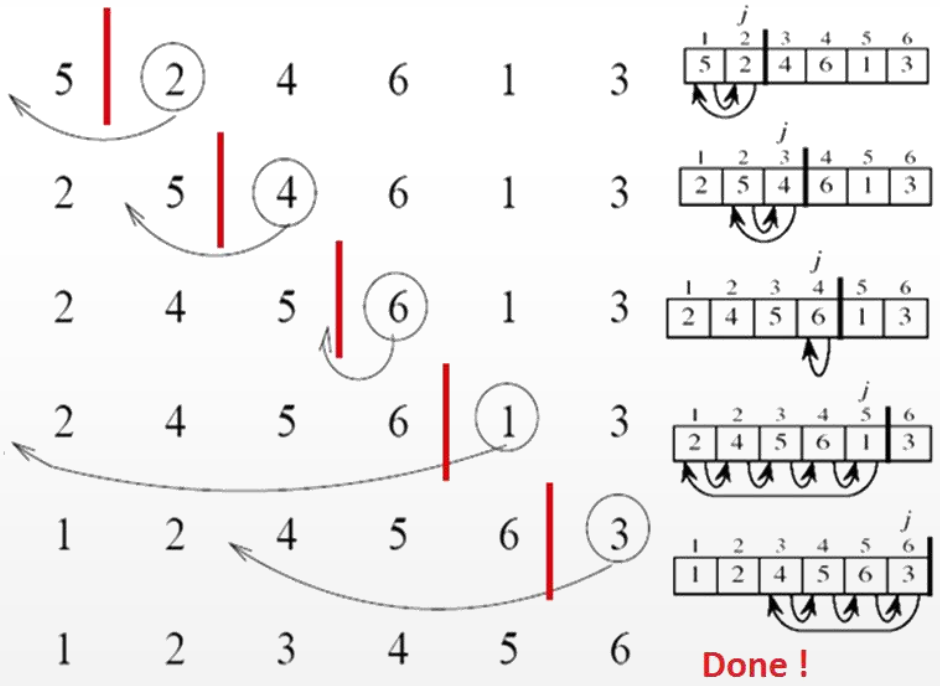
Элементы сортируемого массива «всплывают», как пузырьки воздуха в воде.

Алгоритм проходит по заданному массиву множество раз, каждый раз сравнивая пару соседних друг с другом чисел. Если числа стоят не в том порядке, в котором должны, он меняет их местами.

Сложность: $O(n^2)$

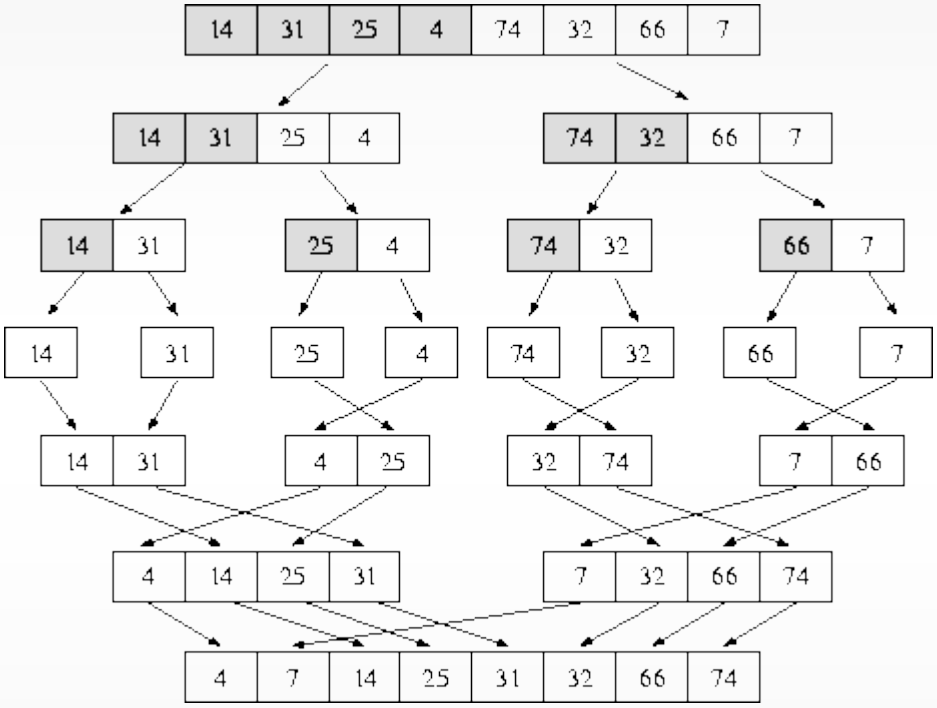
Сортировка вставками (Insertion sort)

Каждый элемент массива вставляется на правильное (относительно остальных) место в новом, отсортированном массиве.



Так как кроме обхода всех элементов массива, для каждого элемента нужно найти место в новом массиве и сдвинуть некоторые элементы (после вставки, но до старого расположения элемента), сложность задачи – $O(n^2)$

Сортировка слиянием (Merge sort)



Главный принцип этого алгоритма: массив из одного элемента уже отсортирован.

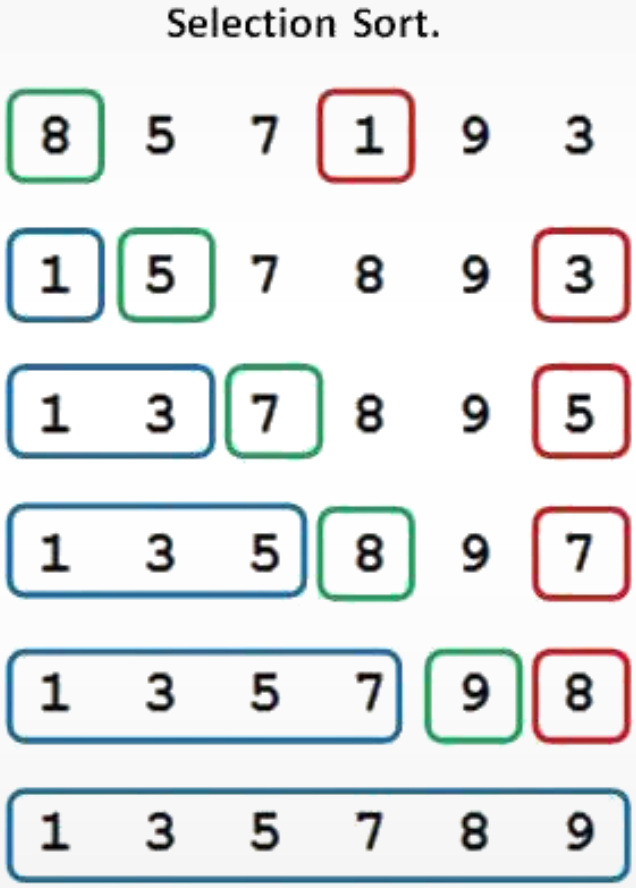
Сортируемый массив разбивается на две части, каждая из которых сортируется отдельно (возможно, этим же самым алгоритмом), а потом результаты сортировок объединяются в один. Это – рекурсивный алгоритм.

Сложность: O(n log n)

Сортировка выбором (Selection sort)

Алгоритм ищет минимальный (или максимальный) элемент сортируемого массива и меняет его местами с первым несортированным элементом. Стандартный поиск минимума для каждого из n элементов – это квадрат.

Сложность: $O(n^2)$



Быстрая сортировка (Quick sort)

Ещё один рекурсивный алгоритм. Сначала проводится грубая оценка массива на основе некоторого элемента, называемого опорным. Все элементы, если сортируем по возрастанию, большие опорного, перекидываются направо от него, а все меньшие – налево. Массив делится на две части, каждая из которых сортируется отдельно.

Разработана англичанином Чарльзом Хоаром в МГУ в 1960 г. как оптимизация сортировки ПУЗЫРЬКОМ!

Сложность: O(n log n)

Зачем нужен поиск?

Серьёзно?

Линейный поиск (Linear search)

(Последовательный поиск, полный перебор, брутфорс)

Последовательный просмотр каждого элемента последовательности, пока не найден нужный. Выполняется за один проход цикла, сложность, соответственно – O(n).

Двоичный поиск (Binary search)

(Бинарный поиск, метод деления пополам)

Поиск элемента в отсортированном массиве. Суть алгоритма – последовательные запросы с предлагаемым вероятным ответом. Ответ на запрос может быть «нужный элемент меньше», «нужный элемент больше» или «это ответ!». Первым предлагается серединный элемент массива. После получения ответа алгоритм отбрасывает половину массива, содержащую неподходящие значения, и начинает поиск на оставшейся половине.

Троичный поиск (Ternary search)

(Тернарный поиск)

Используется для поиска максимума функции, которая строго возрастает, а потом убывает, либо наоборот.

Если известно, что ответ лежит между точками A и B, на этом отрезке выбираются некоторые точки a и b (обычно делящие отрезок на три равные части). Проверяют, на какой из этих двух точек функция принимает меньшее значение. Крайняя треть, отделяемая той точкой, отбрасывается, и поиск продолжается на двух оставшихся третях.

Вопросы?