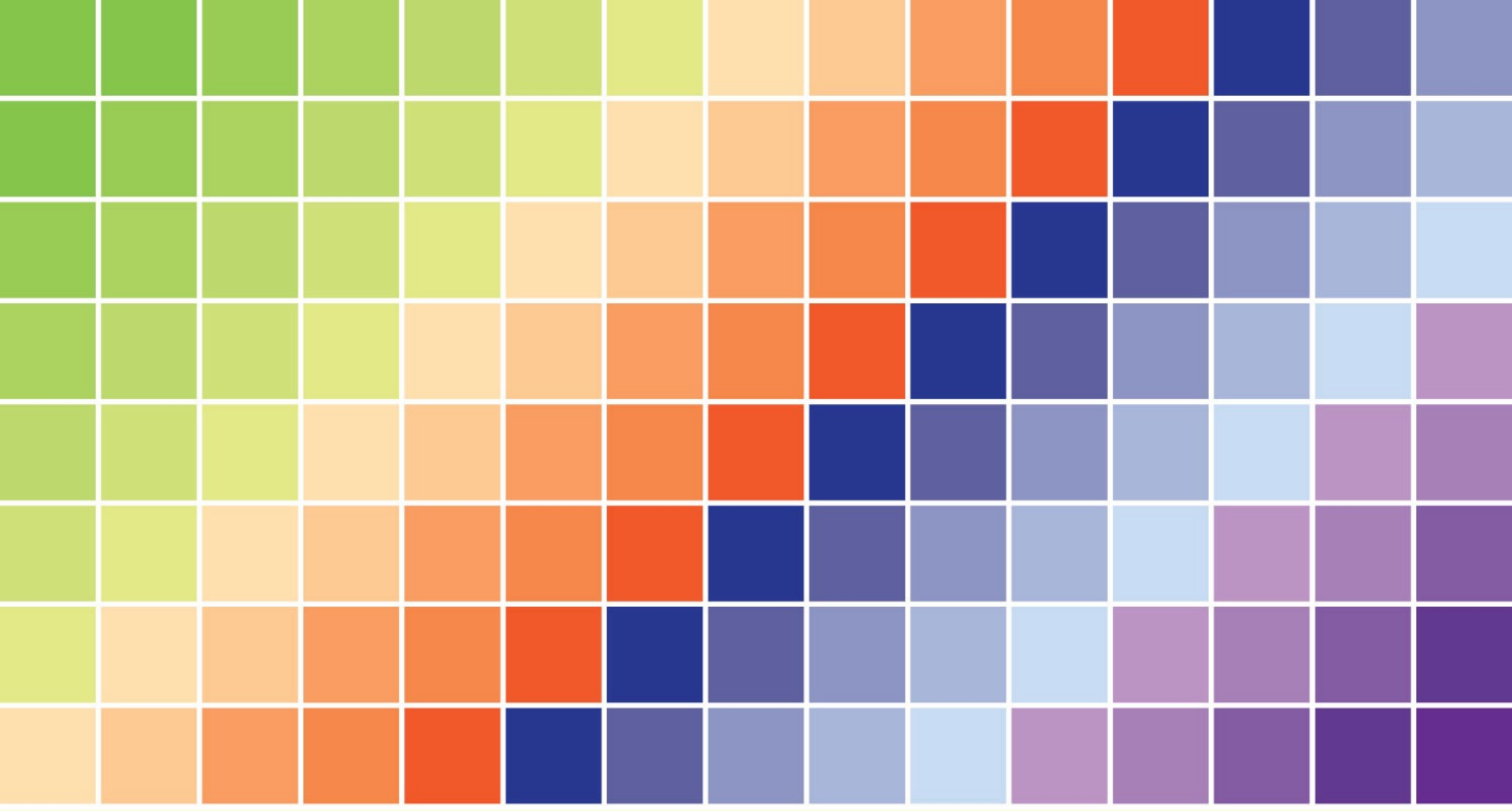


СОРТИРОВКИ

School of Computer Science

ПРОГРАММИРОВАНИЕ НА PYTHON

Лекции для IT-школы

ВОПРОСЫ ПО ПРОШЛОМУ ЗАНЯТИЮ

1. Вычислительная сложность алгоритма это?
2. Какие типовые группы сложностей алгоритмов вы знаете?
3. Какую сложность дают алгоритмы:
   a) Перебора всех элементов списка в цикле
   b) Перебора всех элементов списка в 2-ух вложенных циклах?
4. Если функция вызывает другую функцию, то как посчитать их совместную сложность?
5. А если функции вызываются поочередно?

ВОПРОСЫ ПО ПРОШЛОМУ ЗАНЯТИЮ

6. Программа отрабатывает последовательно алгоритмы сложности $N^3 + N^2$. Какова итоговая вычислительная сложность этой программы?

7. На каком объеме данных оценивается сложность алгоритма:
   a) 1-2 элемента
   b) 10-20 элементов списка
   c) Как можно больше

8. Какой вариант работы алгоритма учитывается при оценке сложности:
   a) Лучший
   b) Средний
   c) Худший

СОРТИРОВКА ВЫБОРОМ МАКСИМУМА
РЕАЛИЗАЦИЯ

- Первый шаг (пример в sel_max_step.py):
  - Находим индекс максимального значения в неотсортированной части списка
  - Помещаем максимальный элемент в конец списка
  - В то место, где был найден максимум, помещаем последний элемент списка

- Продолжение:
  - продолжаем сортировку в цикле, исключая из рассмотрения уже отсортированные элементы
  - **реализуйте сортировку в sel_max_task.py**

СОРТИРОВКА ВЫБОРОМ МАКСИМУМА

ШАГИ

№ итерации	Несортированная часть	Индекс обмена	Сортированная часть
0	[3, 9, 7, 2, 5]	–	–
1	[3, 5, 7, 2,	4	9]
2	[3, 5, 2,	3	7, 9]
3	[3, 2,	2	5, 7, 9]
4	[2,	1	3, 5, 7, 9]
–	–	–	[2, 3, 5, 7, 9]

СОРТИРОВКА ВЫБОРОМ
ИССЛЕДОВАНИЕ АЛГОРИТМА

- В sel_max_time.py найдите:
  1. время сортировки 1000 элементов?
  2. во сколько раз увеличится время работы алгоритма при увеличении длины списка в N раз?
  3. какова, доказанная на практике, сложность этого алгоритма сортировки?
  4. список какой длины этот алгоритм отсортирует за одну секунду?
- Используя шаблон compare_max.py определите будут ли эффективны предложенные оптимизации

СОРТИРОВКА ВЫБОРОМ
ПЕРЕДЕЛКА НА ВЫБОР МИНИМУМА

- По аналогии с реализованным алгоритмом сортировки выбором максимума, **реализуйте алгоритм *сортировки выбором минимума***

- Два алгоритма сортировки выбором (максимума и минимума) сохраните в отдельном модуле в виде функций

- Для каждой функции опишите doctest, включающие автономные тесты

СОРТИРОВКА ВЫБОРОМ МИНИМУМА

ШАГИ

№ итера-ции	Сортированная часть	Несортированная часть
–	–	[3, 7, 2, 9, 5]
0	[2,	7, 3, 9, 5]
1	[2, 3,	7, 9, 5]
2	[2, 3, 5,	9, 7]
3	[2, 3, 5, 7	9]
4	[2, 3, 5, 7, 9]	–

СОРТИРОВКА ВЫБОРОМ МИНИМУМА

ВОПРОС №1

- Три прохода по списку завершено и текущий список выглядит так:
  [3, 4, 5, 7, 8, 10, 6]

- Каким будет список после еще одного прохода:
  1) [3, 4, 5, 7, 8, 6, 10]
  2) [3, 4, 5, 6, 7, 8, 10]
  3) [3, 4, 5, 6, 8, 10, 7]
  4) [3, 4, 5, 6, 7, 10, 8]

???

СОРТИРОВКА ВЫБОРОМ МИНИМУМА
ВОПРОС №2

- Рассмотрите этот список:
  [3, 3, 6, 7, 9, 4, 5]

- Какое максимальное количество проходов выбора минимума уже состоялось:
  1) один
  2) два
  3) три
  4) четыре

???

СОРТИРОВКА ОБМЕНОМ
МЕТОД «ПУЗЫРЬКА»

- Массив упорядочен по возрастанию, если меньшие числа в нем стоят раньше больших
- При последовательном рассмотрении всех пар соседних чисел, нужно для каждой пары:
  - если большее число находится левее меньшего – поменять их местами
  - при выполнении правила для каждой пары чисел весь массив окажется отсортированным
- После полного прохода по массиву со сравнением соседних пар самое большое число будет поставлено на свое место

СОРТИРОВКА ОБМЕНОМ
ОДИН ПРОХОД АЛГОРИТМА

- Дан массив [4, **8**, 3, 5]
- Шаг 1: сравним 4 и 8, список не меняется
- Шаг 2: сравним 8 и 3
  Новый список [4, 3, **8**, 5]
- Шаг 3: сравним 8 и 5
  Новый список [4, 3, 5, **8**]
- См. скрипт bubble_step.py

СОРТИРОВКА ОБМЕНОМ ЗНАЧЕНИЙ
ПРАКТИКУМ

- Реализуйте алгоритм в bubble_task.py
- Определите сложность сортировки методом «пузырька»
- Напишите еще одну функцию сортировки методом обмена с «опусканием» наименьшего элемента в начало списка
- Для новой функции напишите doctest, включающий автономные тесты

СОРТИРОВКА ПО МЕТОДУ «ПУЗЫРЬКА»
ВОПРОС №1

- Два прохода по списку завершены и текущий список выглядит так:
  [2, 3, 2, 4, 1, 5, 7]

- Каким будет список после еще одного прохода сортировки:
  1) [1, 2, 2, 3, 4, 5, 7]
  2) [2, 3, 2, 1, 4, 5, 7]
  3) [2, 2, 3, 4, 1, 5, 7]
  4) [2, 2, 3, 1, 4, 5, 7]

???

СОРТИРОВКА ПО МЕТОДУ «ПУЗЫРЬКА»
ВОПРОС №2

- Рассмотрите этот список:
  [5, 6, 4, 3, 7, 8, 10]

- Какое максимальное количество проходов уже состоялось:
  1) один
  2) два
  3) три
  4) четыре

???

СОРТИРОВКА ВСТАВКАМИ
АЛГОРИТМ

- Первые элементы в левой части списка уже отсортированы
- Берем очередной i-ый элемент в правой части списка и вставляем его в нужное место в сортированной части
- Все элементы с индексом < i, но с большим значением сдвигаются вправо
- **Допишите алгоритм в insertion_task.py**

СОРТИРОВКА ВСТАВКАМИ
ШАГИ

№ итера-ции	Сортированная часть	Несортированная часть
–	–	[7, 3, 5, 2]
0	[7,	3, 5, 2]
1	[3, 7,	5, 2]
2	[3, 5, 7,	2]
3	[2, 3, 5, 7]	–

СОРТИРОВКА ВСТАВКАМИ
ОСОБЕННОСТИ

- Online-сортировка – размещение вновь добавляемых элементов **в уже отсортированный список с линейной сложностью**

- Хорошо работает на частично отсортированных массивах данных

- Используется в реализации стандартных функций Python sort() и sorted() наряду с другими алгоритмами

СОРТИРОВКА ВСТАВКАМИ
ВОПРОС №1

- Три прохода по списку завершены и текущий список выглядит так:
  [2, 4, 5, 1, 3, 1]

- Каким будет список после еще одного прохода сортировки:
  1) [1, 2, 4, 5, 3, 1]
  2) [1, 4, 5, 2, 3, 1]
  3) [1, 1, 2, 4, 5, 3]
  4) [1, 2, 3, 4, 5, 1]

???

СОРТИРОВКА ВСТАВКАМИ
ВОПРОС №2

- Рассмотрите этот список:
  [3, 6, 7, 4, 9, 5]

- Какое максимальное количество проходов уже состоялось:
  1) один
  2) два
  3) три
  4) четыре

???

СОРТИРОВКА ШЕЛЛА
АЛГОРИТМ

1. Зафиксируем некоторое расстояние inc.
2. Разобьём элементы массива на группы – в одну группу попадут элементы, расстояние между которыми inc.
3. Отсортируем вставками каждую группу
4. Уменьшим inc
5. Повторим шаги 1–4, пока inc не будет равным 1.

Реализация алгоритма в shell_sort.py
Сложность этого алгоритма: $O(n) – O(n^2)$

СОРТИРОВКА СЛИЯНИЕМ
АЛГОРИТМ

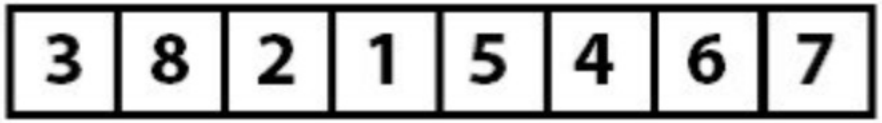
1. Если в рассматриваемом массиве один элемент, или массив пуст, то он уже отсортирован — алгоритм завершает работу

2. Иначе массив разбивается на две части, которые сортируются рекурсивно

3. После сортировки двух частей массива к ним применяется процедура слияния, которая по двум отсортированным частям получает исходный отсортированный массив

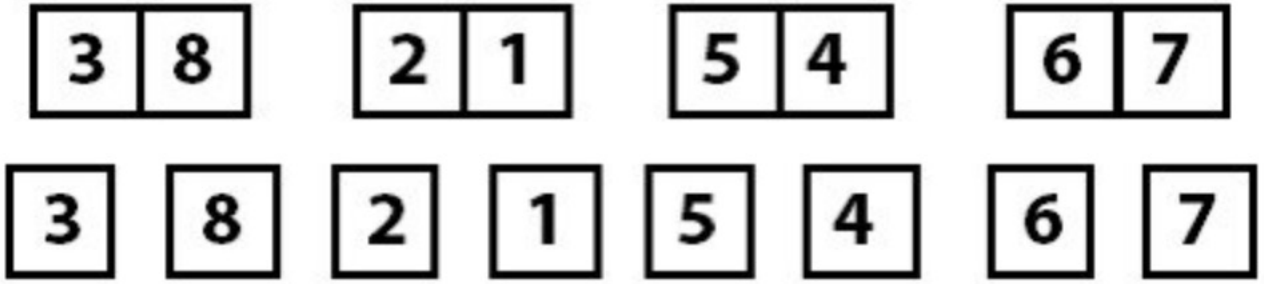
СОРТИРОВКА СЛИЯНИЕМ
РАЗБИЕНИЕ

Давайте посмотрим на такой массив:



Разделим его пополам:

И будем делить каждую часть пополам, пока не останутся части с одним элементом:

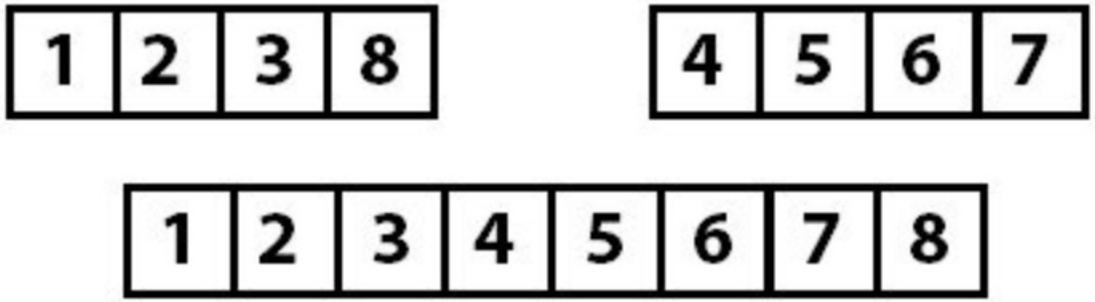


СОРТИРОВКА СЛИЯНИЕМ

СЛИЯНИЕ

Теперь, когда мы разделили массив на максимально короткие участки, мы сливаем их в правильном порядке.

Сначала мы получаем группы по два отсортированных элемента, потом «собираем» их в группы по четыре элемента и в конце собираем все вместе в отсортированный массив.



СОРТИРОВКА СЛИЯНИЕМ
ДОСТОИНСТВА И НЕДОСТАТКИ

Достоинства:

1. Стабилен, т.е не зависит от перестановок элементов в массиве
2. Удобен для структур с последовательным доступом к элементам (файлы)

Недостатки:

1. Требует дополнительной памяти по объёму равной объёму сортируемого файла

СОРТИРОВКА СЛИЯНИЕМ
ОЦЕНКА СЛОЖНОСТИ

Пусть массив содержит n элементов. Тогда за O(n) его можно разделить на две части и после сортировки слить их вместе. Процесс деления продолжается столько раз, сколько раз можно число n делить на 2, до тех пор, пока размер части не станет равен 1, то есть $\log_2 n$. Итого, общая сложность этого алгоритма равна $O(n \log_2 n)$

СОРТИРОВКА СЛИЯНИЕМ
ВОПРОС

Дан следующий список чисел:

[21, 1, 26, 45, 29, 28, 2, 9, 16, 49, 39, 27, 43, 34, 46, 40].

Какой ответ иллюстрирует первые два списка для слияния?

a) [21, 1] и [26, 45]

b) [1, 2, 9, 21, 26, 28, 29, 45] и [16, 27, 34, 39, 40, 43, 46, 49]

c) [21] и [1]

d) [9] и [16]

ПРАКТИЧЕСКОЕ ЗАДАНИЕ

В файле *merge_sort.py* допишите функцию *merge*, сливающую два списка в один по следующему алгоритму:

1. Возьмите элементы в началах обоих списков и сравните их
2. Меньший элемент добавьте в результирующий список и сдвиньте указатель в исходном для элемента списке на следующий элемент
3. Если вы достигли конца одного из списков, добавьте в результирующий все оставшиеся элементы другого списка
4. Верните результирующий список

БЫСТРАЯ СОРТИРОВКА
АЛГОРИТМ

1. Выбрать ключевой индекс и разделить по нему массив на две части

2. Переместить все элементы больше ключевого в правую часть массива, а все элементы меньше ключевого — в левую. Теперь ключевой элемент находится в правильной позиции — он больше любого элемента слева и меньше любого элемента справа

3. Повторить первые два шага, пока массив не будет полностью отсортирован

Пример в *quick_sort.py*

БЫСТРАЯ СОРТИРОВКА ПО ШАГАМ

1. Случайным образом выбираем ключевой элемент

2. Переносим значения в массиве. в процессе переноса значений индекс ключевого элемента может измениться

3. Рекурсивно повторяем для левой и правой частей

БЫСТРАЯ СОРТИРОВКА ПО ШАГАМ

4. И так далее

…

До тех пор, пока не останется одно неотсортированное значение, а, поскольку мы знаем, что все остальное уже отсортировано, алгоритм завершает работу

БЫСТРАЯ СОРТИРОВКА
ОЦЕНКА СЛОЖНОСТИ

В лучшем случае: $O(n \log n)$
В среднем: $O(n \log n)$
В худшем случае: $O(n^2)$

В алгоритме быстрой сортировки, как правило, в качестве барьерного элемента выбирается случайный элемент списка. Тогда алгоритм становится вероятностным — время его работы зависит от того, каким будет случайно выбранный элемент

ВСТРОЕННЫЕ ФУНКЦИИ СОРТИРОВКИ
ХАРАКТЕРИСТИКИ

- Алгоритм функций sort() и sorted() в Python:
  - Адаптивный, комбинированный алгоритм сортировки вставками со слиянием (timsort)
  - Имеет логарифмическую сложность
  - Хорошо подходит для сортировки частично упорядоченных последовательностей
- Смотрите примеры использования стандартных сортировок в sort_example.py
- Сравните скорость работы и объем данных для сортировки в одну секунду для sort() и любого алгоритма сортировки, реализованного нами ранее

СОРТИРОВКА С ДЕКОРИРОВАНИЕМ
SORT / SORTED (KEY =…)

- DSU-сортировка – Decorate, Sort, Undecorate

- Нужно отсортировать массив без учета регистра символов:

```
>>> names = ["Петя", "вася", "Женя", "ДЖЕК", "жучка"]
```

- Делаем это одной строкой кода:

```
>>> names.sort(key=str.lower)
>>> names
['вася', 'ДЖЕК', 'Женя', 'жучка', 'Петя']
```

СОРТИРОВКА С ДЕКОРИРОВАНИЕМ
АЛГОРИТМИЧЕСКАЯ РЕАЛИЗАЦИЯ

DSU-сортировка – явный алгоритм:

```
>>> names = ["Петя", "вася", "Женя", "ДЖЕК", "жучка"]
>>>
>>> temp = []
>>> for item in names:
        temp.append((item.lower(), item))


>>> names = []
>>> for key, value in sorted(temp):
        names.append(value)


>>> names
['вася', 'ДЖЕК', 'Женя', 'жучка', 'Петя']
```

АЛГОРИТМЫ СОРТИРОВОК
ПОЛЕЗНЫЕ ССЫЛКИ

- Курс Learn to Program: Crafting Quality Code. University of Toronto (english): https://www.coursera.org/learn/program-code/home/week/3

- Статья с объяснением алгоритмов сортировок и примерами на Python: https://tproger.ru/translations/sorting-algorithms-in-python

- The Sound of Sorting - "Audibilization" and Visualization of Sorting Algorithms: http://panthema.net/2013/sound-of-sorting

СПАСИБО ЗА ВНИМАНИЕ !

ВОПРОСЫ ?

School of Computer Science