Введение в Python

Упорядоченные списки и сортировки.

Алексей Сорокин

спецкурс, ОТИПЛ МГУ,
осенний семестр 2017–2018 учебного года
24 октября 2017 г.

Упорядоченные списки

- Во многих задачах данные удобно представлять в отсортированном виде:
  - Списки слов, упорядоченные по частотности/ в алфавитном порядке (`http://ruscorpora.ru/corpora-freq.html`):

и	6829968
в	5190215
не	3164900

Упорядоченные списки

- Во многих задачах данные удобно представлять в отсортированном виде:
  - Списки слов, упорядоченные по частотности/ в алфавитном порядке (`http://ruscorpora.ru/corpora-freq.html`):

и	6829968
в	5190215
не	3164900

  - Списки гипотез по оптимальности.

большой дом	
big house	0.92
large house	0.44
big home	0.02

Упорядоченные списки

- Во многих задачах данные удобно представлять в отсортированном виде:
  - Списки слов, упорядоченные по частотности/ в алфавитном порядке (`http://ruscorpora.ru/corpora-freq.html`):

и	6829968
в	5190215
не	3164900

  - Списки гипотез по оптимальности.

большой дом	
big house	0.92
large house	0.44
big home	0.02

  - Расписание заданий по приоритету.

пообедать	2
сделать домашнюю	4
помыть полы	5

Упорядоченные списки

- Нужно уметь сортировать произвольные массивы (списки) данных.

Упорядоченные списки

- Нужно уметь сортировать произвольные массивы (списки) данных.
- Кроме того, в отсортированном списке гораздо легче искать.
- А что значит легче?

Упорядоченные списки

- Нужно уметь сортировать произвольные массивы (списки) данных.
- Кроме того, в отсортированном списке гораздо легче искать.
- А что значит легче?

Угадай число

Загадано число от 1 до 100, угадайте это число за 7 вопросов “да/нет”.

- Давайте поиграем...
- Пусть загадано число 39.

Упорядоченные списки

- Нужно уметь сортировать произвольные массивы (списки) данных.
- Кроме того, в отсортированном списке гораздо легче искать.
- А что значит легче?

Угадай число

Загадано число от 1 до 100, угадайте это число за 7 вопросов “да/нет”.

- Давайте поиграем...
- Пусть загадано число 39.
- Будем задавать только вопросы вида “больше/меньше ли загаданное число, чем x?"

Угадай число

- Загадано число больше 50?

Угадай число

- Загадано число больше 50? Нет.
  Теперь нас интересуют только числа от 1 до 50.

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38?

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44?

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44? Нет.

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44? Нет.
- Загадано число больше 41?

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44? Нет.
- Загадано число больше 41? Нет.

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44? Нет.
- Загадано число больше 41? Нет.
- Загадано число больше 40?

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44? Нет.
- Загадано число больше 41? Нет.
- Загадано число больше 40? Нет.

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44? Нет.
- Загадано число больше 41? Нет.
- Загадано число больше 40? Нет.
- Загадано число больше 39?

Это число 39!

Угадай число

- Загадано число больше 50? Нет.

  Теперь нас интересуют только числа от 1 до 50.
- Загадано число больше 25? Да.

  Теперь знаем, что число из отрезка $[26; 50]$.
- Загадано число больше 38? Да.
- Загадано число больше 44? Нет.
- Загадано число больше 41? Нет.
- Загадано число больше 40? Нет.
- Загадано число больше 39? Нет.

Это число 39!

Анализ задачи

- Можно считать, что мы искали элемент в упорядоченном списке от 1 до 100.

Анализ задачи

- Можно считать, что мы искали элемент в упорядоченном списке от 1 до 100.
- Каждый раз мы делили список пополам, проверяя, в какую половину попадает нужное число.

Анализ задачи

- Можно считать, что мы искали элемент в упорядоченном списке от 1 до 100.
- Каждый раз мы делили список пополам, проверяя, в какую половину попадает нужное число.
- Почему получилось 7 операций?

Анализ задачи

- Можно считать, что мы искали элемент в упорядоченном списке от 1 до 100.
- Каждый раз мы делили список пополам, проверяя, в какую половину попадает нужное число.
- Почему получилось 7 операций?
- Например, для 1000 чисел хватит 10 попыток. А для N?

Анализ задачи

- Можно считать, что мы искали элемент в упорядоченном списке от 1 до 100.
- Каждый раз мы делили список пополам, проверяя, в какую половину попадает нужное число.
- Почему получилось 7 операций?
- Например, для 1000 чисел хватит 10 попыток. А для N?
- Это типичная проблема сложности алгоритмов: сколько операций нужно для выполнения какой-либо задачи для объекта размера N.

Сложность поиска в упорядоченном массиве

Постановка задачи

Дан упорядоченный список длины N, содержащий элементы $x_1 \leqslant x_2 \ldots \leqslant x_N$. Определить, есть ли в нём число x.

Сложность поиска в упорядоченном массиве

Постановка задачи

Дан упорядоченный список длины N, содержащий элементы $x_1 \leqslant x_2 \ldots \leqslant x_N$. Определить, есть ли в нём число x.

Алгоритм решения

- Найти серединный элемент m текущего списка и сравнить x с ним.

Сложность поиска в упорядоченном массиве

Постановка задачи

Дан упорядоченный список длины N, содержащий элементы $x_1 \leqslant x_2 \ldots \leqslant x_N$. Определить, есть ли в нём число x.

Алгоритм решения

- Найти серединный элемент m текущего списка и сравнить x с ним.
- Если $x = m$, то элемент найден.

Сложность поиска в упорядоченном массиве

Постановка задачи

Дан упорядоченный список длины N, содержащий элементы $x_1 \leqslant x_2 \ldots \leqslant x_N$. Определить, есть ли в нём число x.

Алгоритм решения

- Найти серединный элемент m текущего списка и сравнить x с ним.
- Если $x = m$, то элемент найден.
- Если $x > m$, то искать в правой половине, иначе – в левой.

Сложность поиска в упорядоченном массиве

Постановка задачи

Дан упорядоченный список длины N, содержащий элементы $x_1 \leqslant x_2 \ldots \leqslant x_N$. Определить, есть ли в нём число x.

Алгоритм решения

- Найти серединный элемент m текущего списка и сравнить x с ним.
- Если $x = m$, то элемент найден.
- Если $x > m$, то искать в правой половине, иначе – в левой.

Теорема

Данный алгоритм требует не более $\lceil \log_2 N \rceil + 1$ сравнений, где $\lceil \cdot \rceil$ — верхняя целая часть числа.

Сложность поиска в упорядоченном массиве

Теорема (переформулировка)

Если в массиве менее 2^m элементов, то двоичный поиск осуществим за не более, чем $m + 1$ сравнение.

Сложность поиска в упорядоченном массиве

Теорема (переформулировка)

Если в массиве менее 2^m элементов, то двоичный поиск осуществим за не более, чем $m + 1$ сравнение.

Доказательство

- Применим индукцию по m.
- База: $m \leqslant 2$ — поиск занимает не более 2 сравнений.

Сложность поиска в упорядоченном массиве

Теорема (переформулировка)

Если в массиве менее 2^m элементов, то двоичный поиск осуществим за не более, чем $m + 1$ сравнение.

Доказательство

- Применим индукцию по m.
- База: $m \leqslant 2$ — поиск занимает не более 2 сравнений.
- Шаг: если в массиве не более 2^m элементов, то в каждой из его половин не более 2^{m-1}. Соответственно, поиск в каждой из них требует не более m сравнений.

Сложность поиска в упорядоченном массиве

Теорема (переформулировка)

Если в массиве менее 2^m элементов, то двоичный поиск осуществим за не более, чем $m + 1$ сравнение.

Доказательство

- Применим индукцию по m.
- База: $m \leqslant 2$ — поиск занимает не более 2 сравнений.
- Шаг: если в массиве не более 2^m элементов, то в каждой из его половин не более 2^{m-1}. Соответственно, поиск в каждой из них требует не более m сравнений.
- Дополнительно нужно одно сравнение с серединным элементов, то есть всего не более $m + 1$ сравнения.

Сложность поиска в упорядоченном массиве

Теорема (переформулировка)

Если в массиве менее 2^m элементов, то двоичный поиск осуществим за не более, чем $m + 1$ сравнение.

Доказательство

- Применим индукцию по m.
- База: $m \leqslant 2$ — поиск занимает не более 2 сравнений.
- Шаг: если в массиве не более 2^m элементов, то в каждой из его половин не более 2^{m-1}. Соответственно, поиск в каждой из них требует не более m сравнений.
- Дополнительно нужно одно сравнение с серединным элементов, то есть всего не более $m + 1$ сравнения.

Теорема (переформулировка)

Алгоритм двоичного поиска в массиве длины N требует не более $C\lceil \log_2 N \rceil$ элементарных арифметических операций.

Реализация алгоритма

- Ищется индекс элемента x в упорядоченном массиве a.

Реализация алгоритма

- Ищется индекс элемента x в упорядоченном массиве a.
- Поддерживаем два индекса i, j, означающие, что поиск производится в массиве $a[i : j]$.

Реализация алгоритма

- Ищется индекс элемента x в упорядоченном массиве a.
- Поддерживаем два индекса i, j, означающие, что поиск производится в массиве $a[i : j]$.
- На каждом шаге находим индекс середины $m = i + (j - i)//2$.

Реализация алгоритма

- Ищется индекс элемента x в упорядоченном массиве a.
- Поддерживаем два индекса i, j, означающие, что поиск производится в массиве $a[i : j]$.
- На каждом шаге находим индекс середины $m = i + (j - i)//2$.
- Если $a[m] == x$, то выдаём m.

Реализация алгоритма

- Ищется индекс элемента x в упорядоченном массиве a.
- Поддерживаем два индекса i, j, означающие, что поиск производится в массиве $a[i : j]$.
- На каждом шаге находим индекс середины $m = i + (j - i)//2$.
- Если $a[m] == x$, то выдаём m.
- Если $a[m] < x$, то $i = m + 1$.
- Если $a[m] > x$, то $j = m$.

Реализация алгоритма

- Ищется индекс элемента x в упорядоченном массиве a.
- Поддерживаем два индекса i, j, означающие, что поиск производится в массиве $a[i : j]$.
- На каждом шаге находим индекс середины $m = i + (j - i)//2$.
- Если $a[m] == x$, то выдаём m.
- Если $a[m] < x$, то $i = m + 1$.
- Если $a[m] > x$, то $j = m$.
- Завершаем работу, если $j = i$.

Варианты алгоритма

- bisect_left(a, x) — находит индекс i, такой что

$$j < i \quad \Leftrightarrow \quad a[j] < x$$
$$j \geqslant i \quad \Leftrightarrow \quad a[j] \geqslant x$$

Варианты алгоритма

- bisect_left(a, x) — находит индекс i, такой что

$$j < i \quad \Leftrightarrow \quad a[j] < x$$
$$j \geqslant i \quad \Leftrightarrow \quad a[j] \geqslant x$$

- bisect_right(a, x) — находит индекс i, такой что

$$j < i \quad \Leftrightarrow \quad a[j] \leqslant x$$
$$j \geqslant i \quad \Leftrightarrow \quad a[j] > x$$

Варианты алгоритма

- bisect_left(a, x) — находит индекс i, такой что
$$j < i \Leftrightarrow a[j] < x$$
$$j \geqslant i \Leftrightarrow a[j] \geqslant x$$
- bisect_right(a, x) — находит индекс i, такой что
$$j < i \Leftrightarrow a[j] \leqslant x$$
$$j \geqslant i \Leftrightarrow a[j] > x$$
- Реализовано в модуле bisect (`https://docs.python.org/3.1/library/bisect.html`).

Постановка задачи

- Иногда бывает нужно объединить два уже отсортированных списка в новый, сохранив сортировку.

Постановка задачи

- Иногда бывает нужно объединить два уже отсортированных списка в новый, сохранив сортировку.
- Кажется, что это должно быть проще, чем отсортировать список (частичная упорядоченность уже есть).

Постановка задачи

- Иногда бывает нужно объединить два уже отсортированных списка в новый, сохранив сортировку.
- Кажется, что это должно быть проще, чем отсортировать список (частичная упорядоченность уже есть).
- Общая идея:
  - если один из списков пуст, вернуть второй.

Постановка задачи

- Иногда бывает нужно объединить два уже отсортированных списка в новый, сохранив сортировку.
- Кажется, что это должно быть проще, чем отсортировать список (частичная упорядоченность уже есть).
- Общая идея:
  - если один из списков пуст, вернуть второй.
  - выделить минимальный элемент (это один из двух первых элементов $\mathrm{a}[0]$ и $\mathrm{b}[0]$),

Постановка задачи

- Иногда бывает нужно объединить два уже отсортированных списка в новый, сохранив сортировку.
- Кажется, что это должно быть проще, чем отсортировать список (частичная упорядоченность уже есть).
- Общая идея:
  - если один из списков пуст, вернуть второй.
  - выделить минимальный элемент (это один из двух первых элементов $\mathrm{a}[0]$ и $\mathrm{b}[0]$),
  - присоединить к нему справа ответ для $\mathrm{a}[1{:}]$ и b (если минимум выбран из a) или для a и $\mathrm{b}[1{:}]$.

Постановка задачи

- Иногда бывает нужно объединить два уже отсортированных списка в новый, сохранив сортировку.
- Кажется, что это должно быть проще, чем отсортировать список (частичная упорядоченность уже есть).
- Общая идея:
  - если один из списков пуст, вернуть второй.
  - выделить минимальный элемент (это один из двух первых элементов $\mathrm{a}[0]$ и $\mathrm{b}[0]$),
  - присоединить к нему справа ответ для $\mathrm{a}[1{:}]$ и b (если минимум выбран из a) или для a и $\mathrm{b}[1{:}]$.
- На “функциональном” языке:

$$\begin{aligned}\mathrm{merge}(\mathrm{a},\mathrm{b}) \;=\; & \mathrm{b}\ \textbf{if}\ \textbf{len}(\mathrm{a}) == 0\ \textbf{else}\ \mathrm{a}\ \textbf{if}\ \textbf{len}(\mathrm{b}) == 0\ \textbf{else}\\ & \mathrm{a}[0] + \mathrm{merge}(\mathrm{a}[1:],\mathrm{b})\ \textbf{if}\ \mathrm{a}[0] \leqslant \mathrm{b}[0]\ \textbf{else}\\ & \mathrm{b}[0] + \mathrm{merge}(\mathrm{a},\mathrm{b}[1:])\end{aligned}$$

Алгоритм слияния списков

- Можно переписать “функциональный” вариант с помощью рекурсии, но это неэффективно.

Алгоритм слияния списков

- Можно переписать “функциональный” вариант с помощью рекурсии, но это неэффективно.
- Более традиционный алгоритм:
  - Завести два индекса i и j — текущие позиции в обоих списках,

Алгоритм слияния списков

- Можно переписать “функциональный” вариант с помощью рекурсии, но это неэффективно.
- Более традиционный алгоритм:
  - Завести два индекса i и j — текущие позиции в обоих списках,
  - На каждом шаге сравнивать $a[i]$ и $b[j]$.

Алгоритм слияния списков

- Можно переписать “функциональный” вариант с помощью рекурсии, но это неэффективно.
- Более традиционный алгоритм:
  - Завести два индекса i и j — текущие позиции в обоих списках,
  - На каждом шаге сравнивать $a[i]$ и $b[j]$.
    - Если $a[i] \leqslant b[j]$, добавить в ответ $a[i]$, увеличить i;

Алгоритм слияния списков

- Можно переписать “функциональный” вариант с помощью рекурсии, но это неэффективно.
- Более традиционный алгоритм:
  - Завести два индекса i и j — текущие позиции в обоих списках,
  - На каждом шаге сравнивать $a[i]$ и $b[j]$.
    - Если $a[i] \leqslant b[j]$, добавить в ответ $a[i]$, увеличить i;
    - Иначе добавить в ответ $b[j]$, увеличить j.

Алгоритм слияния списков

- Можно переписать “функциональный” вариант с помощью рекурсии, но это неэффективно.
- Более традиционный алгоритм:
  - Завести два индекса i и j — текущие позиции в обоих списках,
  - На каждом шаге сравнивать $a[i]$ и $b[j]$.
    - Если $a[i] \leqslant b[j]$, добавить в ответ $a[i]$, увеличить i;
    - Иначе добавить в ответ $b[j]$, увеличить j.
  - Если один из списков закончился, добавить в ответ остаток второго.

Алгоритм слияния списков

- Можно переписать "функциональный" вариант с помощью рекурсии, но это неэффективно.
- Более традиционный алгоритм:
  - Завести два индекса i и j — текущие позиции в обоих списках,
  - На каждом шаге сравнивать $a[i]$ и $b[j]$.
    - Если $a[i] \leqslant b[j]$, добавить в ответ $a[i]$, увеличить i;
    - Иначе добавить в ответ $b[j]$, увеличить j.
  - Если один из списков закончился, добавить в ответ остаток второго.
- Пример выполнения алгоритма:

$$a = [2, 3, 6, 9] \quad b = [1, 5, 7, 8] \quad i = 0 \quad j = 0 \quad answer = []$$
$$a = [2, 3, 6, 9] \quad b = [1, 5, 7, 8] \quad i = 0 \quad j = 1 \quad answer = [1]$$
$$a = [2, 3, 6, 9] \quad b = [1, 5, 7, 8] \quad i = 1 \quad j = 1 \quad answer = [1, 2]$$
$$a = [2, 3, 6, 6, 9] \quad b = [1, 5, 7, 8] \quad i = 2 \quad j = 1 \quad answer = [1, 2, 3]$$
$$\ldots$$
$$answer = [1, 2, 3, 5, 6, 7, 8, 9]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $a_1 \leqslant \ldots \leqslant a_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $i = 0$.
  - найти j — индекс минимального элемента в $a[i:]$.

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $a_1 \leqslant \ldots \leqslant a_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $i = 0$.
  - найти j — индекс минимального элемента в $a[i:]$.
  - переставить $a[i]$ и $a[j]$ местами.

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $a_1 \leqslant \ldots \leqslant a_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $i = 0$.
  - найти j — индекс минимального элемента в $a[i:]$.
  - переставить $a[i]$ и $a[j]$ местами.
  - увеличить i на 1.

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$
$$[2, 3, 4, 6, 10, 7, 5]$$

Сортировка выбором

- Дан список a, его элементы надо упорядочить в порядке возрастания $\mathrm{a}_1 \leqslant \ldots \leqslant \mathrm{a}_n$.
- Простейший вариант: сортировка выбором
  - i — длина уже отсортированного участка, вначале $\mathrm{i} = 0$.
  - найти j — индекс минимального элемента в $\mathrm{a}[\mathrm{i}:]$.
  - переставить $\mathrm{a}[\mathrm{i}]$ и $\mathrm{a}[\mathrm{j}]$ местами.
  - увеличить i на 1.
  - если $i = \mathbf{len}(\mathrm{a})$, завершить работу.

$$[3, 2, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$
$$[2, 3, 6, 4, 10, 7, 5]$$
$$[2, 3, 4, 5, 6, 7, 10]$$

Сложность сортировки выбором

Утверждение

На поиск максимума в массиве длины n нужно $n - 1$ сравнение.

Сложность сортировки выбором

Утверждение

На поиск максимума в массиве длины n нужно $n-1$ сравнение.

Утверждение

На сортировку массива длины n требуется не более $\frac{n(n-1)}{2}$ сравнений и $(n-1)$ перестановок.

Сложность сортировки выбором

Утверждение

На поиск максимума в массиве длины n нужно $n-1$ сравнение.

Утверждение

На сортировку массива длины n требуется не более $\frac{n(n-1)}{2}$ сравнений и $(n-1)$ перестановок.

- Индукция по n, для массива длины 1 очевидно.

Сложность сортировки выбором

Утверждение

На поиск максимума в массиве длины n нужно $n-1$ сравнение.

Утверждение

На сортировку массива длины n требуется не более $\frac{n(n-1)}{2}$ сравнений и $(n-1)$ перестановок.

- Индукция по n, для массива длины 1 очевидно.
- Сортировка массива длины n занимает $n-1$ итераций, на каждой из которых требуется 1 перестановка для помещения минимального элемента в начало $\Leftrightarrow$ всего $(n-1)$ присваиваний.

Сложность сортировки выбором

Утверждение

На поиск максимума в массиве длины n нужно $n-1$ сравнение.

Утверждение

На сортировку массива длины n требуется не более $\frac{n(n-1)}{2}$ сравнений и $(n-1)$ перестановок.

- Индукция по n, для массива длины 1 очевидно.
- Сортировка массива длины n занимает $n-1$ итераций, на каждой из которых требуется 1 перестановка для помещения минимального элемента в начало $\Leftrightarrow$ всего $(n-1)$ присваиваний.
- Пусть на сортировку массива длины n нужно $S(n)$ сравнений, тогда $S(n) = S(n-1) + (n-1)$.

Сложность сортировки выбором

Утверждение

На поиск максимума в массиве длины n нужно $n-1$ сравнение.

Утверждение

На сортировку массива длины n требуется не более $\frac{n(n-1)}{2}$ сравнений и $(n-1)$ перестановок.

- Индукция по n, для массива длины 1 очевидно.
- Сортировка массива длины n занимает $n-1$ итераций, на каждой из которых требуется 1 перестановка для помещения минимального элемента в начало $\Leftrightarrow$ всего $(n-1)$ присваиваний.
- Пусть на сортировку массива длины n нужно $S(n)$ сравнений, тогда $S(n) = S(n-1) + (n-1)$.
- $S(n) = S(n-1) + (n-1) \leqslant \frac{(n-1)(n-2)}{2} + (n-1) = \frac{(n-1)(n)}{2}$.