Отчет

1. **Сортировка пузырьком**

**Идея:** Попарно сравниваются соседние элементы и меняются местами, если они стоят в неправильном порядке. Проход по массиву повторяется до тех пор, пока массив не будет отсортирован.

**Сложность:** O(n²) в худшем и среднем случае.

**Особенности:** Простая в реализации, но очень медленная на больших данных. Используется только в учебных целях.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Как работает?

Начальный массив: [4 2 1 5]

1 шаг: [2 4 1 5]

2 шаг: [2 1 4 5]

3 шаг: [2 1 4 5] ничего не изменилось, т.к 4 меньше 5

4 шаг: [1 2 4 5]

**2. Шейкерная сортировка**

**Идея:** Улучшение пузырьковой сортировки. Проходы выполняются сначала слева направо, затем справа налево.

**Сложность:** O(n²), но на практике немного быстрее пузырьковой.

**Особенности:** Немного эффективнее пузырьковой, но всё ещё медленная.

Изображение выглядит как текст, снимок экрана, Шрифт, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

Как работает?

Массив [5, 3, 8, 4, 2]

**Первый проход (слева направо)**:

Сравниваем 5 и 3, меняем: [3, 5, 8, 4, 2]

Сравниваем 5 и 8, не меняем: [3, 5, 8, 4, 2]

Сравниваем 8 и 4, меняем: [3, 5, 4, 8, 2]

Сравниваем 8 и 2, меняем: [3, 5, 4, 2, 8]

Правую границу уменьшаем: теперь массив выглядит как [3, 5, 4, 2, 8].

**Второй проход (справа налево)**:

Сравниваем 2 и 4, меняем: [3, 5, 2, 4, 8]

Сравниваем 2 и 5, меняем: [3, 2, 5, 4, 8]

Сравниваем 2 и 3, меняем: [2, 3, 5, 4, 8]

Левую границу увеличиваем: массив теперь [2, 3, 5, 4, 8].

**Третий проход (слева направо)**:

Сравниваем 2 и 3, не меняем: [2, 3, 5, 4, 8]

Сравниваем 3 и 5, не меняем: [2, 3, 5, 4, 8]

Сравниваем 5 и 4, меняем: [2, 3, 4, 5, 8]

Правую границу уменьшаем: массив [2, 3, 4, 5, 8].

**Четвертый проход (справа налево)**:

Сравниваем 4 и 5, не меняем: [2, 3, 4, 5, 8]

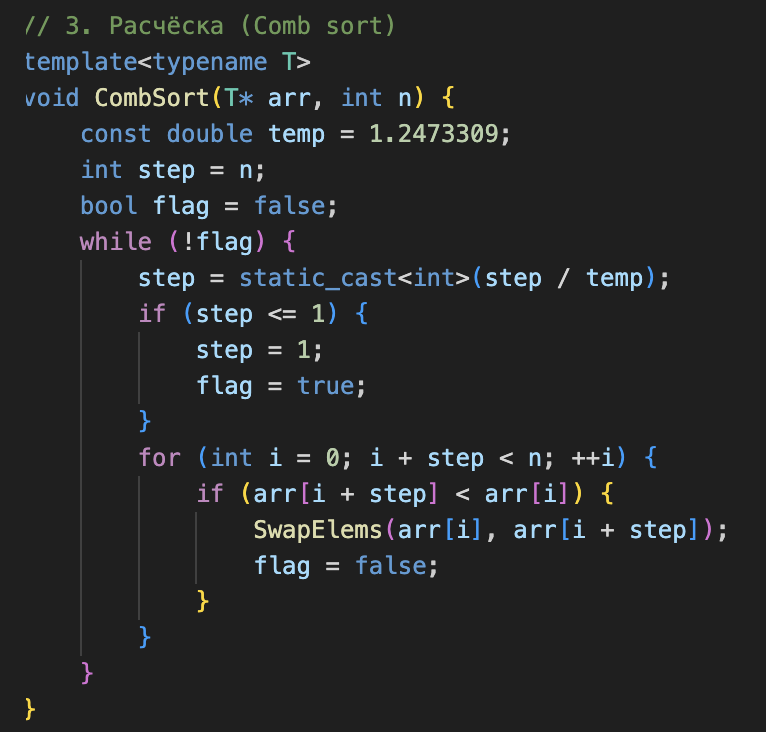
Левую границу увеличиваем: теперь границы пересеклись.

**3. Сортировка расческой**

**Идея:** Улучшение пузырьковой сортировки. Изначально сравниваются элементы на большом расстоянии (шаг вычисляется через "фактор уменьшения"), благодаря чему большие элементы становятся в конец массива(«устранить» элементы с небольшими значения в конце массива, которые замедляют работу алгоритма).

**Сложность:** В среднем O(n log n), в худшем случае O(n²).

**Особенности:** Устраняет главный недостаток пузырьковой сортировки — "черепах" (мелкие элементы в конце массива), поэтому работает намного быстрее.



**4. Сортировка вставками**

**Идея:** Массив условно делится на отсортированную и неотсортированную части. Элементы из неотсортированной части по одному вставляются на правильную позицию в отсортированной.

**Сложность:** O(n²) в худшем случае, O(n) — в лучшем (если массив уже отсортирован).

**Особенности:** Эффективна на небольших массивах и практически отсортированных данных.

Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

Пример:

5 2 4 3 1

2 5 4 3 1

2 4 5 3 1

2 3 4 5 1

1 2 3 4 5

**5. Сортировка выбором**



**Идея:** Массив делится на отсортированную и неотсортированную части. На каждом шаге в неотсортированной части ищется минимальный (или максимальный) элемент и помещается в конец отсортированной части.

**Сложность:** O(n²) для всех случаев.

**Особенности:** Простая, но неэффективная. Главное преимущество — минимум swap'ов (обменов) элементов (всего O(n)).

Пример:

4 3 1 5 2

1 3 4 5 2

1 2 4 5 3

1 2 3 4 5

**6. Быстрая сортировка**

**Идея:**

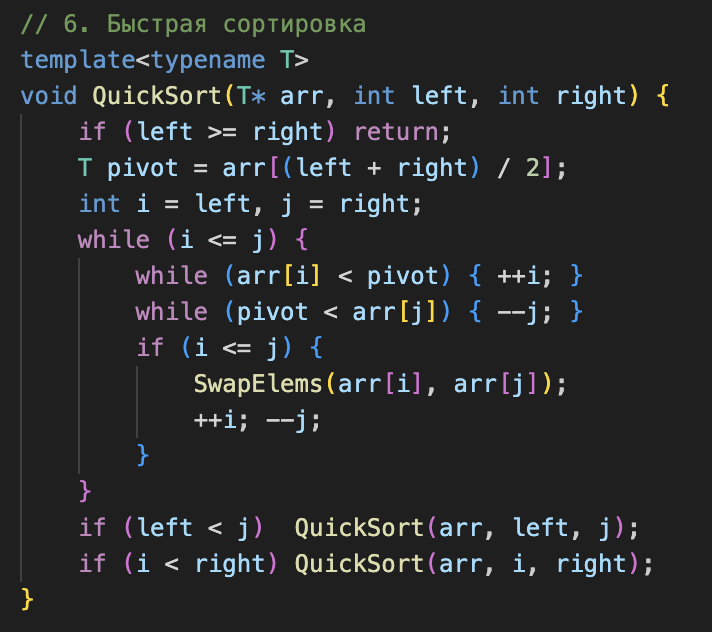
1. Выбирается **опорный элемент** (pivot).

2. Массив перераспределяется так, чтобы элементы меньше опорного оказались слева, а больше — справа.

3. Рекурсивно применяются первые два шага к двум подмассивам.

**Сложность:** O(n log n) в среднем, O(n²) в худшем (при неудачном выборе опорного элемента).

**Особенности:** Один из самых быстрых и распространённых алгоритмов на практике.



Пример:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 4 | 3 | 1 | 5 | 2 |  |
| <2 | **2** | >=2 |  | | |
| **1** |  | 4 | 3 | 5 |  |
|  | | | <5 | **5** | >=5 |
|  | 4 | 3 |  | | |
|  | <3 | **3** | >=3 |  | |
|  | | | **4** |  |  |

**7. Сортировка слиянием**

Изображение выглядит как текст, снимок экрана, программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Идея:** Алгоритм "разделяй и властвуй".

1. Массив рекурсивно разбивается на две половины до тех пор, пока не останутся подмассивы размером в один элемент.

2. Отсортированные подмассивы сливаются (merge) в один большой отсортированный массив.

**Сложность:** Всегда O(n log n).

**Особенности:** Стабильная, надёжная сортировка. Требует дополнительной памяти O(n).

Пример:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | | 4 | 3 | 1 | 5 | 2 |  | |
|  | | | | | | | | |
|  | 4 | 3 | 1 |  | | 5 | 2 |  |
|  | | | | | | | | |
| 4 |  | 3 |  | 1 |  | 5 |  | 2 |
|  | | | | | | | | |
|  | 3 | 4 |  | 1 |  | 2 | 5 |  |
|  | | | | | | | | |
|  | | 1 | 3 | 4 |  | 2 | 5 |  |
|  | | **1** | **2** | **3** | **4** | **5** |  | |

**8. Сортировка подсчётом**

**Идея:** Не основана на сравнениях. Работает с числами в заданном диапазоне.

1. Создаётся массив-счётчик, где индекс — число из исходного массива, а значение — количество его вхождений.

2. На основе массива-счётчика формируется результирующий отсортированный массив.

**Сложность:** O(n + k), где k — размер диапазона чисел.

**Особенности:** Очень быстрая, но только для целых чисел с небольшим диапазоном. Требует много памяти, если диапазон велик.

Пример:

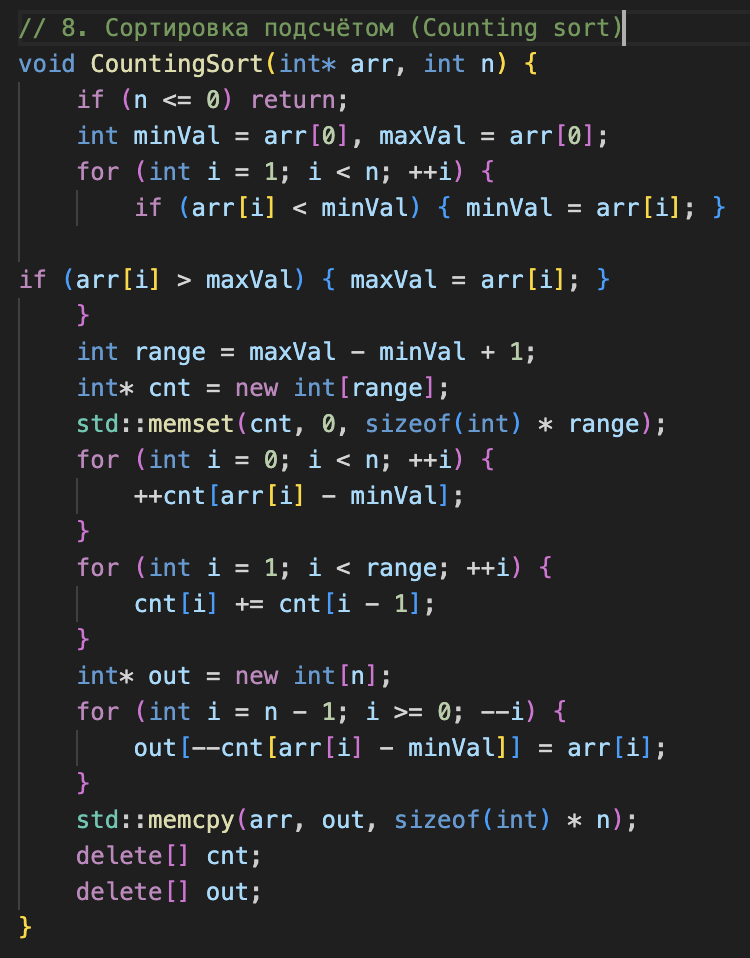
1 3 1 2 0 4 2 3

Кол-во = 8

К = 5

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 2 | 3 | 4 |
| 1 | 2 | 2 | 2 | 1 |

Получаем отсортированный массив 0 1 1 2 2 3 3 4

****

**9. BogoSort**

**Идея:**

1. Проверить, отсортирован ли массив.

2. Если нет — перемешать элементы случайным образом.

3. Повторять шаги 1 и 2 до тех пор, пока массив не окажется отсортирован.

**Сложность:** O((n+1)!) в среднем, бесконечность в худшем случае.

**Особенности:** Абсолютно не практичный алгоритм.

**Изображение выглядит как текст, снимок экрана, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.**