Билет 13. Сортировка Шелла

22 октября 2025 г.

Билет 13. Сортировка Шелла

Основная идея: Вспомним алгоритм сортировки вставками. Попробуем начать с большого шага, уменьшая его до одного.

Простая аналогия: Сортировка книг на полке

Представьте, что у вас есть книги, разбросанные на длинной полке:

Шаг 1: Большой промежуток (Сортировка издалека)

- Смотрим на книги, находящиеся на 8 позиций друг от друга, и меняем их местами при необходимости
- Это быстро распределяет книги по правильным "окрестностям"

Шаг 2: Средний промежуток (Сортировка со среднего расстояния)

- Смотрим на книги, находящиеся на 4 позиции друг от друга
- Уточняем порядок внутри каждой окрестности

Шаг 3: Маленький промежуток (Сортировка ближайших соседей)

- Смотрим на книги, находящиеся на 2 позиции друг от друга
- Массив почти отсортирован!

Шаг 4: Без промежутка (Финальная проверка)

- Смотрим на непосредственных соседей (промежуток = 1)
- Это обычная **сортировка вставками** на почти отсортированном массиве

Пошаговый пример

Сортируем массив: $[8, 3, 5, 1, 4, 2]$

Начальное состояние:

`8, 3, 5, 1, 4, 2`

Проход 1: Промежуток = 3

Сравниваем элементы на расстоянии 3:

- Сравниваем `8  1` → Меняем местами: `1, 3, 5, 8, 4, 2`
- Сравниваем `3  4` → Не меняем
- Сравниваем `5  2` → Меняем местами: `1, 3, 2, 8, 4, 5`

Проход 2: Промежуток = 1 (обычная сортировка вставками)

Теперь массив: `[1, 3, 2, 8, 4, 5]` - уже частично отсортирован!

Финальный проход сортирует соседей: **[1, 2, 3, 4, 5, 8]**

Время работы (б/д)

- **Зависит от выбора последовательности промежутков**
- **Лучший случай:** $O(n \log n)$
- **Средний случай:** зависит от последовательности промежутков
- **Худший случай:** $O(n^2)$ для некоторых последовательностей
- **На практике:** работает значительно быстрее, чем $O(n^2)$

```java
public static void shellSort(int[] arr) {  1 usage
  int n = arr.length;

  for (int gap = n / 2; gap > 0; gap /= 2) {
    for (int i = gap; i < n; i++) {
      int temp = arr[i];
       int j;
      for (j = i; j - gap >= 0 && arr[j - gap] >= temp; j -= gap) {
        arr[j] = arr[j - gap];
      }

      arr[j] = temp;
    }
  }
}
```

Рис. 1: ShellSort

Лучший случай

Условие: Массив уже отсортирован или почти отсортирован
Пример: $[1, 2, 3, 4, 5, 6, 7, 8]$
Последовательность промежутков: 4, 2, 1

- **Промежуток 4:** Сравнения, но нет перестановок
- **Промежуток 2:** Сравнения, но нет перестановок
- **Промежуток 1:** Только проверка соседей, нет перестановок

Время работы: $O(n \log n)$ - каждый проход работает за $O(n)$

Худший случай

Условие: Массив специально подобран так, чтобы каждый промежуток делал минимальный прогресс
Пример: $[8, 1, 4, 2, 3, 5, 7, 6]$ с последовательностью промежутков 4, 2, 1

- **Промежуток 4:**
  - Группы: [8,3], [1,5], [4,7], [2,6]
  - Результат: [3,1,4,2,8,5,7,6] - почти без улучшений
- **Промежуток 2:**
  - Группы: [3,4,8,7], [1,2,5,6]
  - Результат: [3,1,4,2,7,5,8,6] - минимальные улучшения
- **Промежуток 1:** Фактически выполняет полную сортировку вставками на плохо отсортированном массиве

Время работы: $O(n^2)$ - вырождается в сортировку вставками наихудшего случая