Рекурсия. Быстрые сортировки. Сортировка подсчетом

Denis Bakin

Рекурсия вокруг нас

- Рекурсия — ситуация, когда объект является частью себя

Фракталы как пример рекурсии

- Фракталы — самоподобные объекты
- Классический пример — **треугольник Серпинского**

Figure 2: Итерации треугольника Серпинского

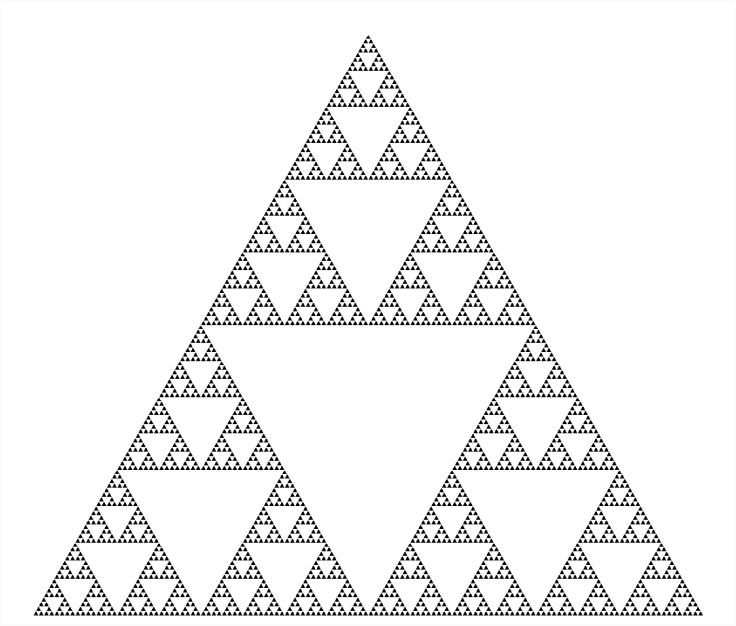


Figure 3: Треугольник Серпинского

Рекуррентные формулы

- Последовательности можно задавать через рекуррентное соотношение
- Пример — числа Фибоначчи:

$$\begin{cases} F_1 = 1 \\ F_2 = 1 \\ F_n = F_{n-1} + F_{n-2}, n > 2 \end{cases}$$

Рекурсия в программировании

- Рекурсия в коде — вызов функции из неё самой
- Применение:
  - обход графов (DFS)
  - генерация перестановок, слов
  - сортировки
  - вычислительная геометрия (например, выпуклая оболочка)

Иллюстрация DFS

Суть рекурсии

- Сведение большой задачи к нескольким подзадачам того же типа
- Решение подзадач $\rightarrow$ решение исходной задачи

Пример: сумма массива рекурсивно

Пример: сумма массива рекурсивно

- Сумма массива = первый элемент + сумма оставшихся

$$\text{sum}(i, j) = \begin{cases} a_i + \text{sum}(i+1, j), & i \neq j \\ a_i, & i = j \end{cases}$$

Альтернативное условие выхода

$$\mathrm{sum}(i, j) = \begin{cases} 0, & i > j \\ a_i + \mathrm{sum}(i+1, j), & \text{иначе} \end{cases}$$

Реализация в C++

```cpp
// args: {1, 2, 3, 4}, 0, 3
int recursive_sum(const std::vetor<int>& nums, int start, int stop) {
    if (start == stop) {
        return nums[start];
    }
    return nums[start] + recursive_sum(nums, start + 1, stop);
}
```

```
finding sum of 0 to 3
finding sum of 0 to 3 by summing 0 and sum of 1 to 3
finding sum of 1 to 3
finding sum of 1 to 3 by summing 1 and sum of 2 to 3
finding sum of 2 to 3
finding sum of 2 to 3 by summing 2 and sum of 3 to 3
finding sum of 3 to 3
found sum of 3 to 3: 4
10
```

Рекомендации по работе с рекурсией

1. Разбить задачу на меньшие подзадачи
2. Определить условие выхода
3. Проверить, что рекурсия всегда доходит до выхода
4. Думать о рекурсивной функции как о «чёрном ящике», который уже реализован

Пример: максимальная цифра числа

Пример: максимальная цифра числа

- Сведение: $\mathsf{max\_digit}(N) = \max(N\%10, \mathsf{max\_digit}(N/10))$
- Условие выхода: число из одной цифры
- $N < 10 \Rightarrow \mathsf{max\_digit}(N) = N$

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
- $$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \ : \ (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
- $$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \ : \ (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$
- Рекурсия:

$$\gcd(a, b) = \begin{cases} a, & b = 0 \\ \gcd(b, a \bmod b), & b > 0 \end{cases}$$

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
- $$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \ : \ (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$
- Рекурсия:
$$\gcd(a, b) = \begin{cases} a, & b = 0 \\ \gcd(b, a \bmod b), & b > 0 \end{cases}$$
- $a \leq b \Rightarrow a \bmod b \leq \frac{a}{2}$

Алгоритм Евклида

- $\gcd(a, b)$ — наибольший общий делитель (greatest common divisor)
- $$d = \gcd(a, b) \Rightarrow \begin{cases} a \vdots d \\ b \vdots d \end{cases} \Rightarrow \forall k \in \mathbb{Z} \ : \ (a - k \cdot b) \vdots d \Leftrightarrow (a \bmod b) \vdots d$$
- Рекурсия:
$$\gcd(a, b) = \begin{cases} a, & b = 0 \\ \gcd(b, a \bmod b), & b > 0 \end{cases}$$
- $a \leq b \Rightarrow a \bmod b \leq \frac{a}{2}$
- Работает за $O(\log \min(a, b))$

Реализация алгоритма Евклида

```
int gcd(int a, int b) {
    if (b == 0) {
        return a;
    }
    return gcd(b, a % b);
}
```

Нижняя граница сортировки сравнениями

- Дерево решений для сортировки:
  - $n!$ листьев (все перестановки)
  - высота $\geq n \log_2 n$
- Нижняя граница:

$$\Omega(n \log n)$$

Сортировка слиянием

Операция merge

Вводим базовое преобразование – слияние двух отсортированных массивов:

- Линейный проход по обоим массивам
- На каждом шаге выбираем меньший из текущих элементов
- Время работы: $O(n)$

Сортировка слиянием

Разделяй и властвуй

- Разбиваем массив на половины
- Рекурсивно сортируем
- Сливаем отсортированные половины

Сортировка слиянием

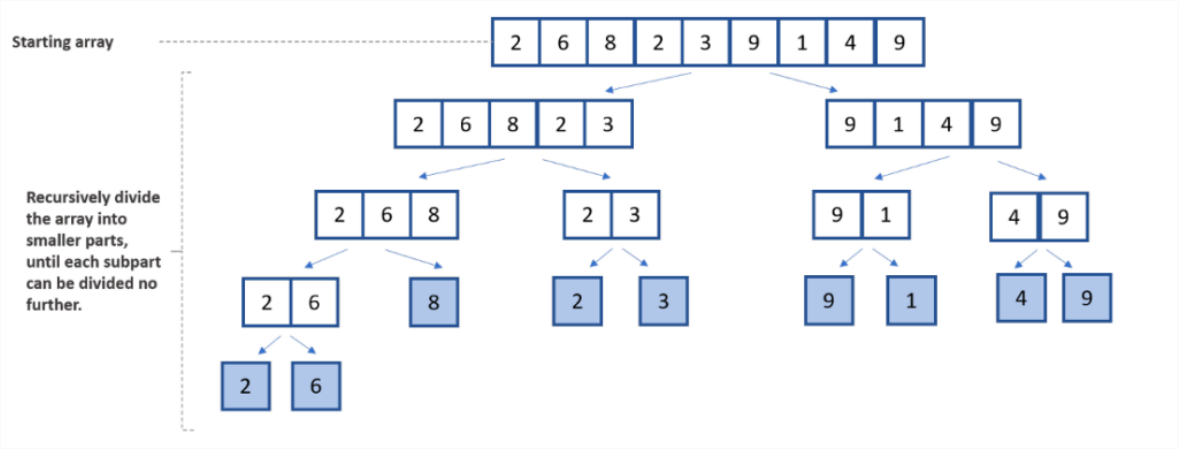


Figure 4: Разбиение

Сортировка слиянием

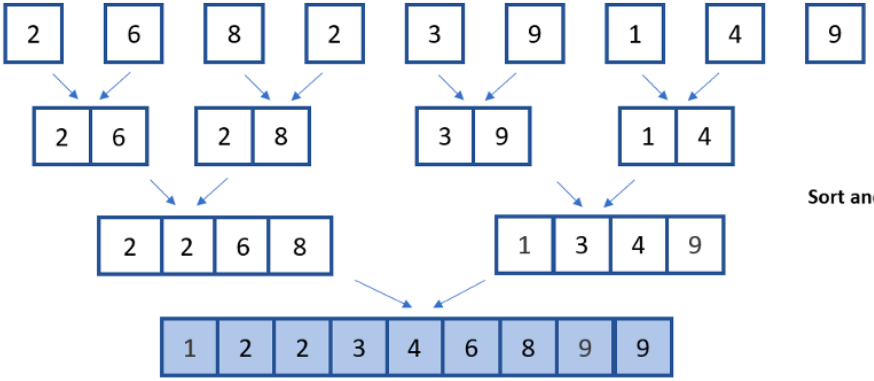


Figure 5: Слияние

Сложность merge sort

- Слияние — $O(n)$

Сложность merge sort

- Слияние — $O(n)$
- Глубина рекурсии — $\log n$, так как делим массив пополам

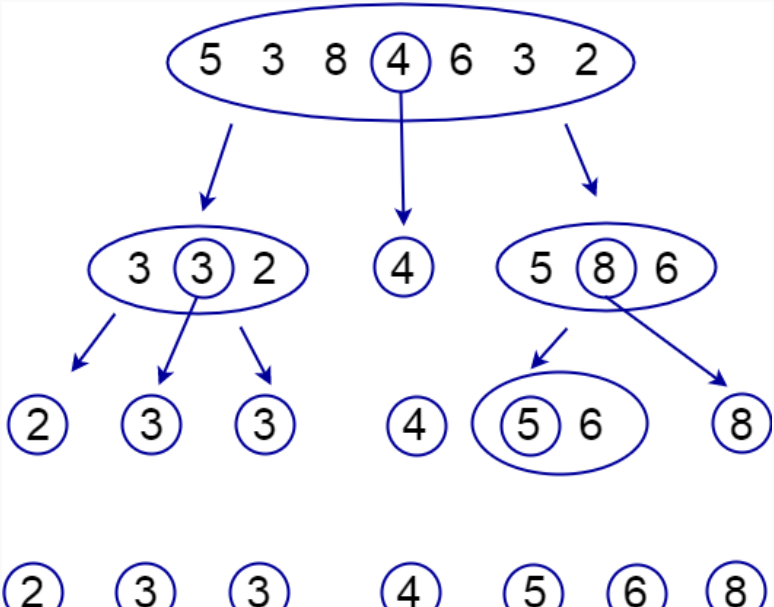
Сложность merge sort

- Слияние — $O(n)$
- Глубина рекурсии — $\log n$, так как делим массив пополам
- Итого: $O(n \log n)$

Быстрая сортировка: идея

- Алгоритм “разделяй и властвуй”
- Основная операция — partition (разбиение массива относительно опорного элемента)
- После partition:
  - слева — элементы $\leq$ pivot
  - справа — элементы $>$ pivot
- Затем рекурсивно сортируем каждую часть

Быстрая сортировка



Partition: основная операция

- Вход: массив $arr[l..r]$
- Выбираем опорный элемент (pivot)
- Двигаем два указателя:
  - i — от начала к середине (ищем элемент $>$ pivot)
  - j — от конца к середине (ищем элемент $<$ pivot)
- Если нашли “перепутанные” элементы — меняем их местами
- Повторяем, пока $i \leq j$

Partition: псевдокод (Hoare)

```
функция partition(arr, l, r):
    v = arr[(l + r) // 2]  # pivot
    i = l
    j = r
    пока i <= j:
        пока arr[i] < v:
            i = i + 1
        пока arr[j] > v:
            j = j - 1
        если i >= j:
            break
        поменять arr[i], arr[j]
        i = i + 1
        j = j - 1
    вернуть j
```

Быстрая сортировка: рекурсивный вызов

```
function quick_sort(a, l, r):
    если l < r:
        q = partition(a, l, r)
        quick_sort(a, l, q)       # сортируем левую часть
        quick_sort(a, q + 1, r)   # сортируем правую часть
```

- Условие выхода: массив из 0 или 1 элемента
- На каждом шаге массив делится на части, которые сортируются независимо

Временная сложность quick sort

- Средний случай: $O(n \log n)$
- Худший случай: $O(n^2)$ при неудачном разбиении: 1 элемент и $n - 1$ элементов

Как избежать худшего случая

- Если pivot всегда выбирается одинаково, можно построить худший вход
  - пример: pivot = первый элемент, вход = отсортированный массив
- Чтобы избежать квадратичной работы:
  - выбирать pivot случайным образом
  - выбирать медиану из трёх случайных элементов

Сортировка подсчётом

- Подходит, если элементы из небольшого диапазона
- Считаем частоты каждого значения
- Выводим элементы в порядке возрастания
- Время работы: $O(n + m)$, где m — количество возможных значений

Counting sort illustration