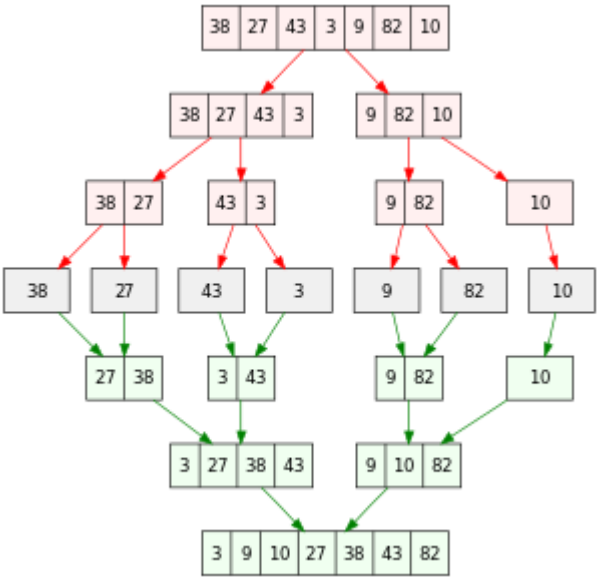
Содержание

merge sort

Сортировка слиянием (Merge Sort) — классический алгоритм, часто упоминаемый в связке с Quick Sort. Он также использует подход «разделяй и властвуй», но с акцентом на объединение (слияние) отсортированных подмассивов в итоговую структуру.

Алгоритм делит массив на две части, рекурсивно сортирует каждую из них, а затем объединяет их в один отсортированный массив.



Как выглядит алгоритм:

1. Делим массив на две равные части, пока каждая не станет одноэлементной.
2. Сравниваем элементы каждой пары и сливаем их в один отсортированный подмассив.
3. Повторяем шаг 2, пока не получим один полностью отсортированный массив.

Эта структура делает алгоритм понятным и легко реализуемым, особенно в языках с поддержкой рекурсии: Python и Java. Однако операции требуют дополнительной памяти для хранения временных подмассивов. Это один из основных недостатков Merge Sort.

Merge Sort — один из немногих алгоритмов, которые гарантируют временную сложность O(n log n) в худшем, среднем и лучшем случаях. Все это — благодаря рекурсивному делению массива и упорядочиванию уже отсортированных подмассивов.

Пространственная сложность Merge Sort составляет O(n). Требуется дополнительная память для хранения временных массивов, используемых на каждом этапе рекурсии.

Таким образом, в ситуациях, где важна постоянная скорость выполнения независимо от распределения данных, лучше выбирать Merge Sort. Quick Sort может страдать от квадратичной сложности в худших случаях.

Пример реализации

```
#include <iostream>
```

```cpp
#include <vector>

// Рекурсивная реализация merge (слияния) двух отсортированных векторов
std::vector<int> merge(const std::vector<int>& left, const std::vector<int>&
right) {
  std::vector<int> sorted_list;
  sorted_list.reserve(left.size() + right.size());

  size_t i = 0, j = 0;
  // Сравниваем элементы левой и правой части и сливаем их в один вектор
  while (i < left.size() && j < right.size()) {
    if (left[i] < right[j]) {
      sorted_list.push_back(left[i]);
      ++i;
    } else {
      sorted_list.push_back(right[j]);
      ++j;
    }
  }

  // Добавляем оставшиеся элементы (одна из частей уже пустая)
  while (i < left.size()) {
    sorted_list.push_back(left[i]);
    ++i;
  }
  while (j < right.size()) {
    sorted_list.push_back(right[j]);
    ++j;
  }

  return sorted_list;
}

// Рекурсивная реализация merge sort, возвращающая новый вектор
std::vector<int> merge_sort(const std::vector<int>& arr) {
  if (arr.size() <= 1) return arr; // базовый случай: 0 или 1 элемент

  size_t mid = arr.size() / 2;
  std::vector<int> left(arr.begin(), arr.begin() + mid);
  std::vector<int> right(arr.begin() + mid, arr.end());

  // Рекурсивная сортировка левой и правой половин
  std::vector<int> sorted_left = merge_sort(left);
  std::vector<int> sorted_right = merge_sort(right);

  // Слияние отсортированных половин
  return merge(sorted_left, sorted_right);
}

int main() {
  std::vector<int> array = {12, 11, 13, 5, 6, 7};
```

```
  std::vector<int> sorted_array = merge_sort(array);

  std::cout << "Отсортированный массив: ";
  for (size_t i = 0; i < sorted_array.size(); ++i) {
    if (i) std::cout << ", ";
    std::cout << sorted_array[i];
  }
  std::cout << std::endl;
  return 0;
}
```