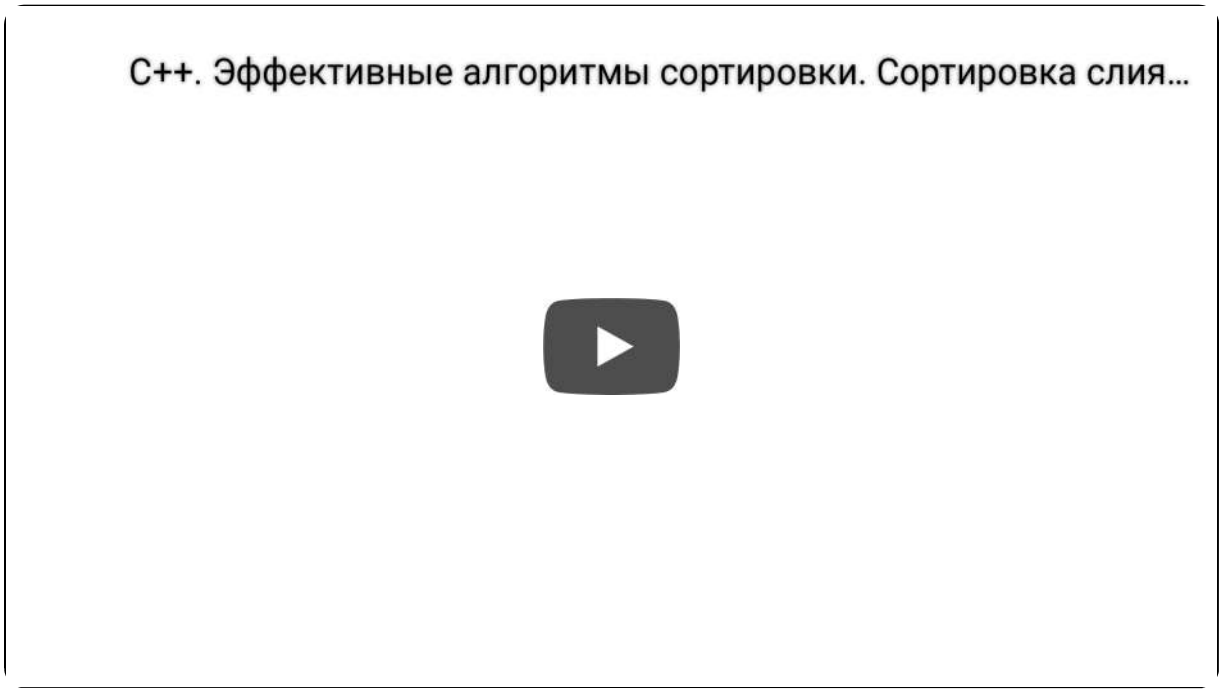
Сортировка слиянием



Кроме квадратичных универсальных сортировок существуют и более эффективные сортировки со сложностью $O(n \log n)$, где n — количество элементов в массиве.

Рассмотрим один из таких алгоритмов — **сортировку слиянием**.

Идея заключается в следующем: рассмотрим сортируемый список. Если он состоит из одного элемента, то он уже отсортирован. В противном случае разобьём его на две примерно равные (с точностью до единицы) части и отсортируем их, рекурсивно применив для каждой из частей аналогичную сортировку. После этого начнём «сливать» отсортированные части. Для этого будем хранить индексы начала двух отсортированных частей. Новый список будет создаваться следующим образом: мы будем выбирать меньший из двух элементов, на которые указывают индексы, и дописывать его в новый список. После этого сдвинем индекс, элемент которого мы взяли, на следующий элемент этого списка. Когда у нас закончится одна из сливаемых частей, то элементы второй мы просто добавим в конец итогового списка. В итоге получим отсортированный список, который надо поместить на место сливаемых частей.

Рассмотрим работу процедуры "слияния" двух отсортированных частей на

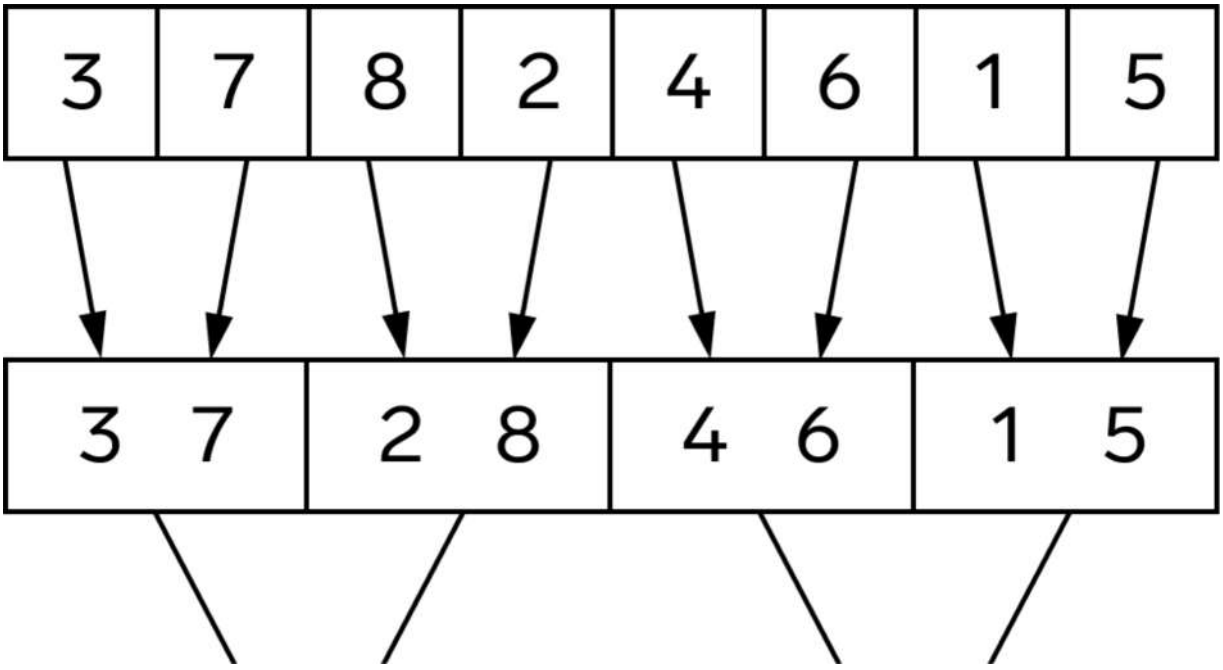
примере:

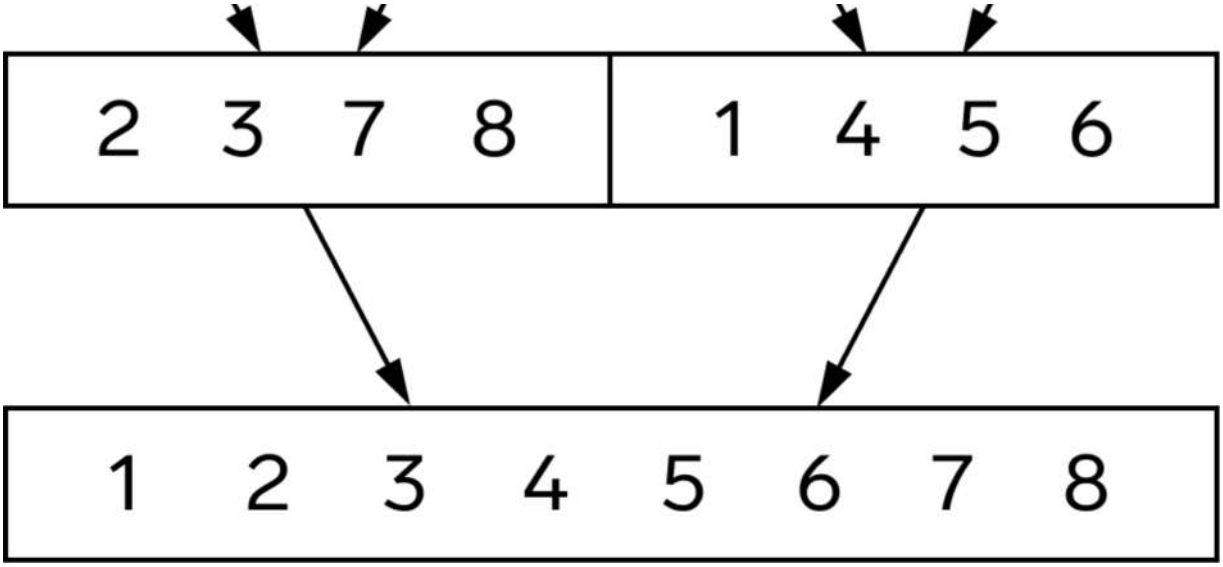
$$[4, 5, 5, 7, 8] \text{ и } [2, 4, 6, 8, 9]$$

Части массива		Результат
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2, 4]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2, 4, 4]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2, 4, 4, 5]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2, 4, 4, 5, 5]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2, 4, 4, 5, 5, 6]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2, 4, 4, 5, 5, 6, 7]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	[2, 4, 4, 5, 5, 6, 7, 8]
[4, 5, 5, 7, 8]	[2, 4, 6, 8, 9]	**[2, 4, 4, 5, 5, 6, 7, 8, 8, 9]**

Осталось только разобраться, как упорядочить каждую из двух половинок исходного массива. Сделаем это также при помощи сортировки слиянием. Разобьём эти части на две, упорядочим их и сольём результат вместе. Меньшие части в свою очередь также разбиваются на две ещё меньшие части, которые упорядочиваются тем же алгоритмом и затем сливаются.

Проиллюстрируем эту идею:





Таким образом, сортировка слиянием — это рекурсивный алгоритм, который будет разбивать массив на две части и упорядочивать две меньшие части тем же самым алгоритмом, а затем сливать результат вместе.

Реализация

```
vector<int> merge(vector<int> a, vector<int> b) {
    vector<int> res;
    int i = 0;
    int j = 0;
    while (i < a.size() && j < b.size()) {
        if (a[i] <= b[j])
            res.push_back(a[i++]);
        else
           res.push_back(b[j++]);
    }
    while (i < a.size())
        res.push_back(a[i++]);
    while (j < b.size())
        res.push_back(b[j++]);
    return res;
}

vector<int> msort(vector<int> a) {
    if (a.size() <= 1)
        return a;
```

```
    int k = a.size() / 2;
    return merge(
        msort(vector<int>(a.begin(), a.begin() + k)),
        msort(vector<int>(a.begin() + k, a.end())));
}
```

Оценим сложность работы алгоритма сортировки слиянием. Сложность алгоритмов сортировки обычно измеряют в количестве наиболее типичных операций — сравнений элементов массива.

Изучим ход нашего алгоритма от самого внешнего рекурсивного вызова. Мы разбили массив из n элементов на два массива из $n/2$ элементов, а затем их слили. Слияние двух массивов выполняется за время, пропорциональное сумме их длин, то есть у нас будет не более n операций сравнения.

На следущем уровне рекурсии мы делим массивы длиной $n/2$ на массивы длиной $n/4$ и затем объединяем 4 массива длиной $n/4$ попарно в два массива длиной $n/2$. Для этого понадобится также число сравнений, равное суммарной длине исходных массивов, то есть тоже n операций сравнения.

На следующем уровне рекурсии массивы размером $n/8$ объединяются попарно в массивы размером $n/4$, и для этого тоже понадобится суммарно n операций сравнения.

На каждом уровне рекурсии общее число операций равно n. А чтобы определить количество уровней рекурсии, нужно заметить, что размер одного кусочка на следующем шаге сокращается в два раза и рекурсия остановится, когда размер кусочка станет равен 1. Это произойдёт, когда будет выполнено не более, чем $\lceil \log_2 n \rceil$ шагов. То есть общее число операций будет равно $O(n \log(n))$.

Это гораздо лучше, чем рассмотренные раньше квадратичные алгоритмы сортировки, сложность которых равна $O(n^2)$. Можно доказать, что не существует универсального алгоритма сортировки, который требует меньше, чем $O(n \log(n))$ сравнений. Под универсальным алгоритмом подразумевается алгоритм, который только сравнивает элементы массива между собой и не использует иной информации о свойствах элементов массива.

Может показаться, что сортировка подсчётом имеет меньшую сложность, то есть работает быстрее, но при этом сортировка подсчётом не является универсальным алгоритмом сортировки, поскольку применима только для массивов, набор значений в которых ограничен.