

Сортировки

Булгаков Илья, Гусев Илья

Московский физико-технический институт

Москва, 2020

Содержание

- 1 Задача сортировки
- 2 Виды сортировок
 - Сортировка вставками
 - Пирамидальная сортировка (HeapSort)
 - Сортировка слиянием (MergeSort)
 - Быстрая сортировка (QuickSort)
- 3 Сравнение сортировок
- 4 Библиотечные функции сортировки
- 5 Особенности реализации
- 6 Доказательство $\Omega(n \log(n))$ для сортировок сравнениями

Задача сортировки

Пусть требуется упорядочить N элементов: R_1, R_2, \dots, R_n .

K - K - ключ сортировки, $\forall j \in 1 \dots n, K_j \in R_j$

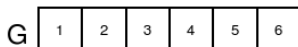
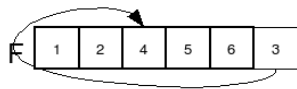
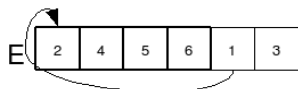
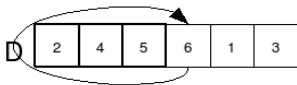
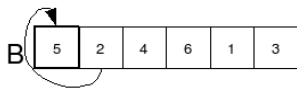
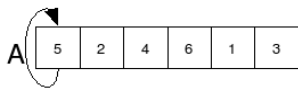
$\forall a, b, c \in K \rightarrow (a < b) \vee (b > a) \vee (a = b)$

$\forall a, b, c \in K \rightarrow (a < b) \wedge (b < c) \Rightarrow (a < c)$

Найти $p(1)p(2) \dots p(n)$ т.ч. $K_{p(1)} \leq K_{p(2)} \leq \dots \leq K_{p(n)}$

Устойчивая (стабильная) перестановка: $\forall i < j, K_{p(i)} = K_{p(j)} \rightarrow p(i) < p(j)$

Сортировка вставками



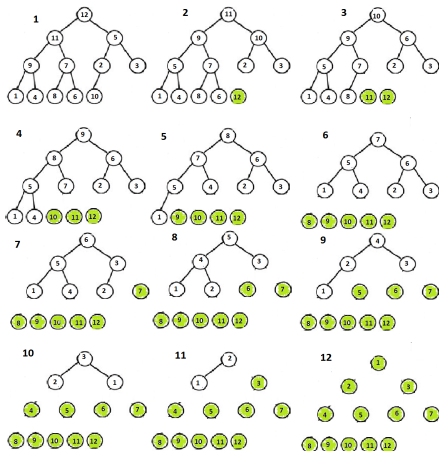
Сложность? Устойчивость? Доп. память? Сложность на уже сортированных массивах?

Сортировка вставками (Insertion Sort)

Код

```
template <class T>
void insertion_sort(std::vector<T>& collection) {
    for (size_t i = 1; i < collection.size(); i++) {
        T key = collection[i];
        size_t j = i - 1;
        while (j >= 0 && collection[j] > key) {
            collection[j + 1] = collection[j];
            j -= 1;
        }
        A[j+1] = key;
    }
}
```

Пирамидальная сортировка (HeapSort)

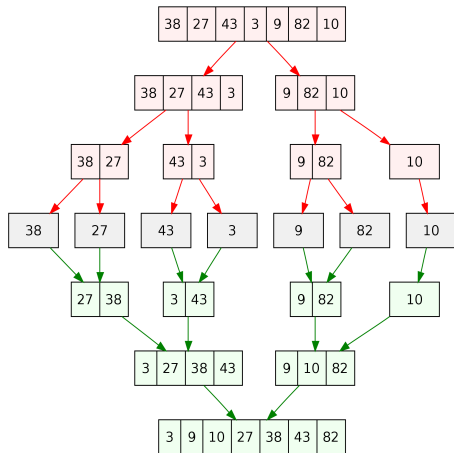


- 1 Строим над коллекцией кучу
- 2 Делаем ExtractMin n раз
(минимум перемещаем в конец)
- 3 ...
- 4 PROFIT!

Сложность? Устойчивость? Доп.
память? Сложность на уже
сортированных массивах?

Сортировка слиянием (MergeSort)

Recursive

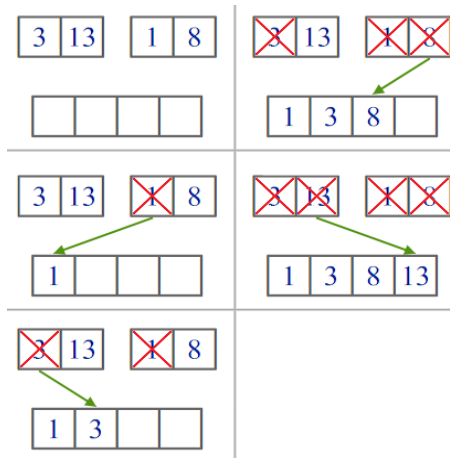


Рекурсивный вариант

- 1 Сортируемый массив разбивается на две части примерно одинакового размера
- 2 Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом
- 3 Два упорядоченных массива половинного размера соединяются в один

Сортировка слиянием (MergeSort)

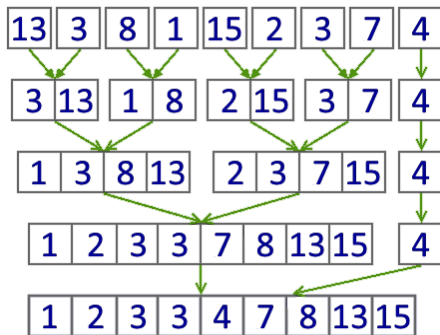
Merge



- Время работы процедуры: $\Theta(m)$, где m - суммарное количество входных данных
- Суммарно для всех вызовов на одном уровне: $\Theta(n)$, где n - количество элементов коллекции

Сортировка слиянием (MergeSort)

Iterative

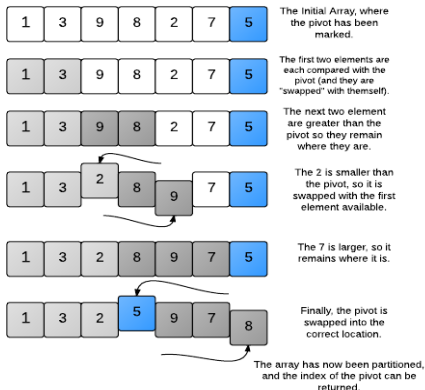


- Альтернатива: итеративный алгоритм
- Сложность? Устойчивость? Доп. память? Сложность на уже сортированных массивах?

Быстрая сортировка (QuickSort)

Partition

Partitioning an array



- Сделать Partition коллекции
- Partition - берём опорный элемент, а остальные элементы делим на две части: меньше опорного и больше или равные опорному. $\Theta(n)$
- Для обеих частей рекурсивно выполняем Partition
- Тип Partition: Хоара или Ломута

Быстрая сортировка (QuickSort)

Сложность? Устойчивость? Доп. память? Сложность на уже сортированных массивах?

Модификации:

- Устойчивость через второй ключ-индекс
- Выбор опорного элемента: первый, последний, средний, медианный из 3, случайный
- Разбиение на 3 части

Быстрая сортировка (QuickSort)

Оптимизации при реализации

- 1 Оптимизация выбора опорного элемента
Медиана из трёх, например.
- 2 Оптимизация Partition
Использовать разбиение Хоара (два индекса, которые приближаются друг к другу) вместо разбиения Ломута
- 3 Написать с одной ветвью рекурсии
Рекурсивный вызов вызывается для меньшего куска после деления, больший остаётся сортироваться в теле функции
- 4 Написать без рекурсии
Рекурсивных вызовов не остается вообще
- 5 Оптимизация концевой рекурсии
Когда в отрезке остается мало элементов, то перестаем использовать алгоритм быстрой сортировки и переходим на сортировку вставками

Сравнение сортировок

Алгоритм	Худшее	Лучшее	В среднем	Sorted	Уст.	+ память
Insertion	$\Theta(n^2)$	$\Theta(n)$	$\mathcal{O}(n^2)$	$\Theta(n)$	Да	$\Theta(1)$
Heap	$\Theta(n \log(n))$	$\Theta(n \log(n))$	$\Theta(n \log(n))$	$\Theta(n \log(n))$	Нет	$\Theta(1)$
Merge	$\Theta(n \log(n))$	$\Theta(n \log(n))$	$\Theta(n \log(n))$	$\Theta(n \log(n))$	Да	$\Theta(n)$
Quick	$\Theta(n^2)$	$\Theta(n \log(n))$	$\Theta(n \log(n))$	$\Theta(n^2)$	Нет	$\Theta(1)$

Библиотечные функции сортировки

std::sort

- 1 std::sort - функция сортировки на [first, last).
Объявлена в заголовочном файле <algorithm>
Сложность $O(N \cdot \log(N))$

```
#include <algorithm>
int main() {
    int a[5] = { 4, 2, 7, 9, 6 };
    std::sort(a, a + 5);
}
```

- 2 std::stable_sort - функция устойчивой сортировки на [first, last).

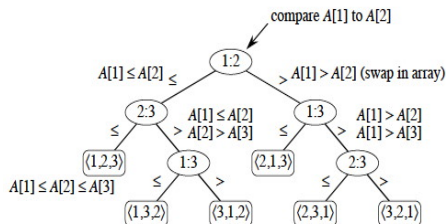
Оптимизация ввода/вывода

- ❶ `std::ios_base::sync_with_stdio(false);` - отключение синхронизации C++ потоков и вывода C. Быстрее, но может приводить к перемешиванию вывода. Если включено, но нет буфферизации, и код потоко-безопасный.

```
std::ios::sync_with_stdio(false);  
std::cout << "a\n";  
std::printf("b\n");  
std::cout << "c\n";
```

```
// Возможный вывод:  
// b  
// a  
// c
```

- ❷ `cin.tie(0)` и `cout.tie(0)`. По умолчанию `cin` привязан к `cout`, что означает, что перед каждой операцией над `cin` сначала сбрасывается буфер `cout`. Если отключить, то в интерактивных программах пользователь может не получить вывод своевременно.

Доказательство $\Omega(n \log(n))$ для сортировок сравнениями

- $\text{count}(\text{leaves}) = l \geq n!$
- $l \leq 2^h$
- $n! \leq l \leq 2^h \Rightarrow \log_2(n!) \leq h$
- $n! > \left(\frac{n}{e}\right)^n$
 - $1 > \frac{1}{e}$
 - $\left(\frac{n+1}{e}\right)^{n+1} = \left(\frac{n}{e}\right)^n \frac{(n+1)^{n+1}}{n^ne} = \left(\frac{n}{e}\right)^n \left(1 + \frac{1}{n}\right)^n \frac{n+1}{e} > n!(n+1) = (n+1)!$
- $h \geq \log_2\left(\frac{n}{e}\right)^n = n \cdot \log_2\left(\frac{n}{e}\right) = \Omega(n \cdot \log(n))$

Полезные ссылки I



ICS 311 #10: Theoretical Limits of Sorting, $O(n)$ Sorts

<https://www2.hawaii.edu/~nodari/teaching/s17/Notes/Topic-10.html>



Wiki - Sorting algorithm

https://en.wikipedia.org/wiki/Sorting_algorithm



Викиконспекты: Сортировка слиянием

<https://bit.ly/2DH6XmF>