Сортировки

Булгаков Илья,Гусев Илья

Московский физико-технический институт

Москва, 2020

Содержание

Задача сортировки

Пусть требуется упорядочить N элементов: $R_1, R_2, \ldots, R_n$.
K - K - ключ сортировки, $\forall j \in 1 \ldots n, K_j \in R_j$
$\forall a, b, c \in K \to (a < b) \vee (b > a) \vee (a = b)$
$\forall a, b, c \in K \to (a < b) \wedge (b < c) \Rightarrow (a < c)$
Найти $p(1)p(2)\ldots p(n)$ т.ч. $K_{p(1)} \leq K_{p(2)} \leq \cdots \leq K_{p(n)}$
Устойчивая (стабильная) перестановка: $\forall i < j, K_{p(i)} = K_{p(j)} \to p(i) < p(j)$

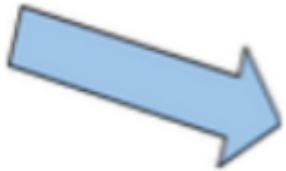
Быстрая сортировка (QuickSort)

Устойчивая сортировка

Пример устойчивой сортировки. Вы сначала отсортировали по фамилии, а потом хотите отсортировать по грейдам, но **сохранить порядок по фамилиям**

An Example, Code, and a Demo

BEFORE	
Name	**Grade**
Dave	C
Earl	B
Fabian	B
Gill	B
Greg	A
Harry	A



AFTER	
Name	**Grade**
Greg	A
Harry	A
Earl	B
Fabian	B
Gill	B
Dave	C

Быстрая сортировка (QuickSort)

Устойчивая сортировка

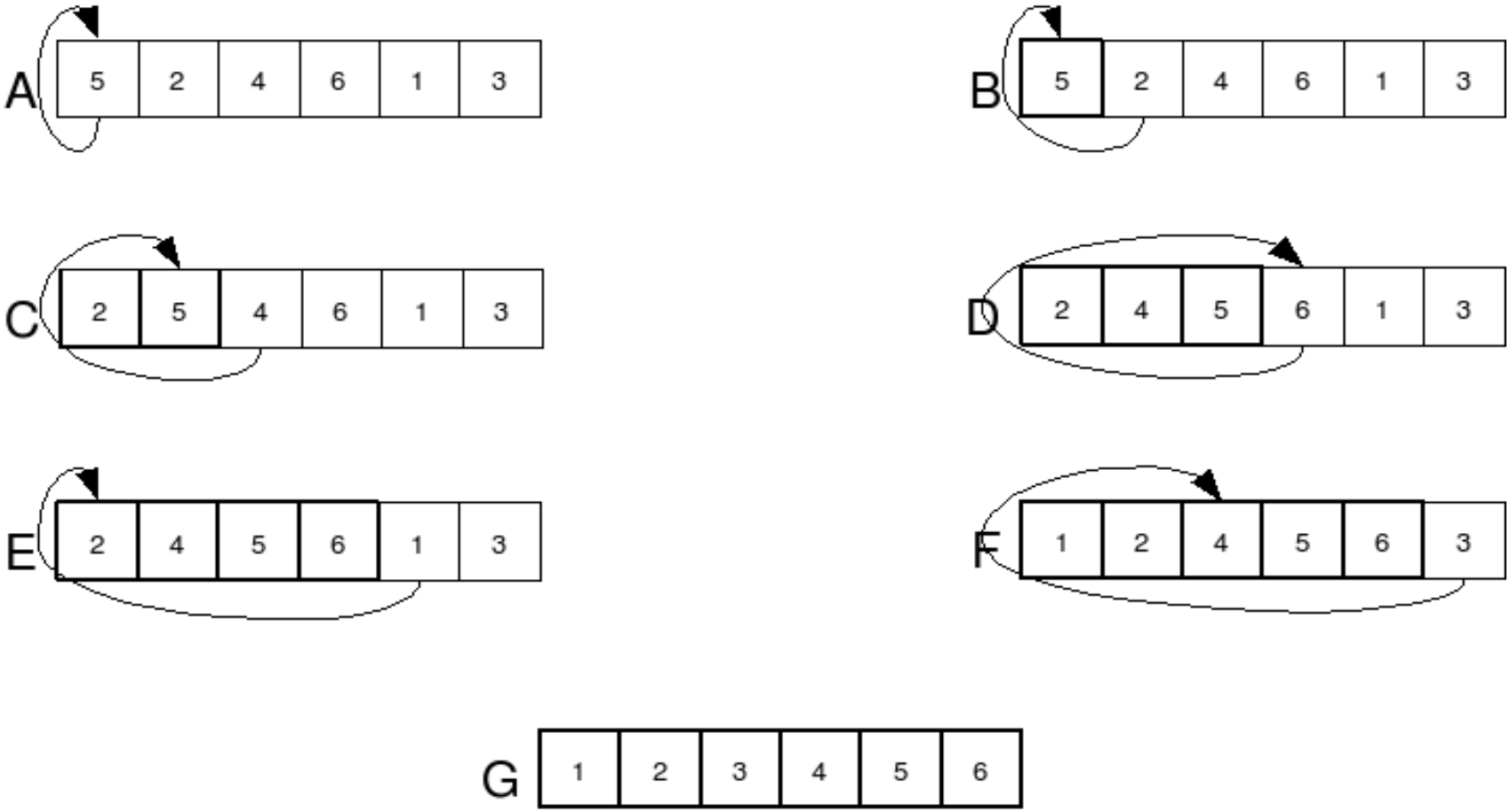
Пример неустойчивой сортировки

BEFORE	
Name	Grade
Dave	C
Earl	B
Fabian	B
Gill	B
Greg	A
Harry	A

AFTER	
Name	Grade
Greg	A
Harry	A
Gill	B
Fabian	B
Earl	B
Dave	C

Чем плохо - вы проделали часть работы по сортировке по одному ключу, а сортировка по другому ключу все испортила

Сортировка вставками



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

Сортировка вставками (Insertion Sort)

Код

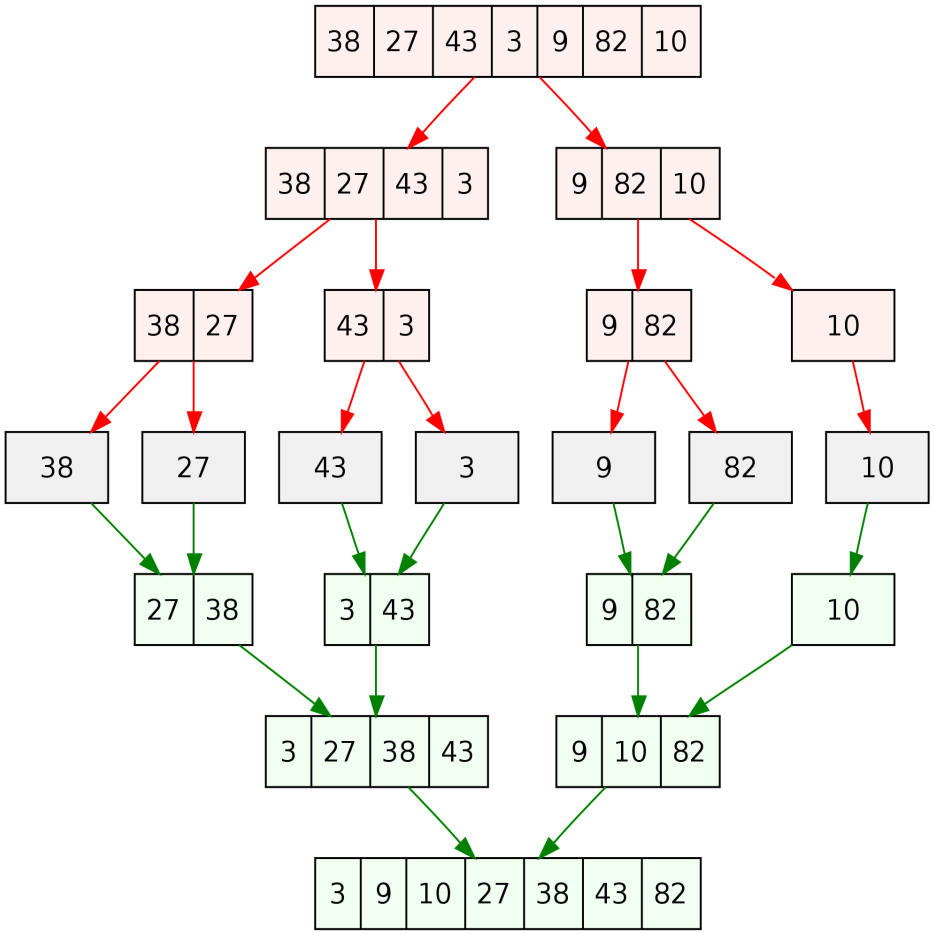
```
template <class T>
void insertion_sort(std::vector<T>& collection) {
    for (size_t i = 1; i < collection.size(); i++) {
        T key = collection[i];
        size_t j = i - 1;
        while (j >= 0 && collection[j] > key) {
            collection[j + 1] = collection[j];
            j -= 1;
  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

Сортировка вставками

- Квадратичное время в худшем случае
- Линейное на уже отсортированном массиве
- Устойчива
- Не требует доп.памяти

Сортировка слиянием (MergeSort)

Recursive

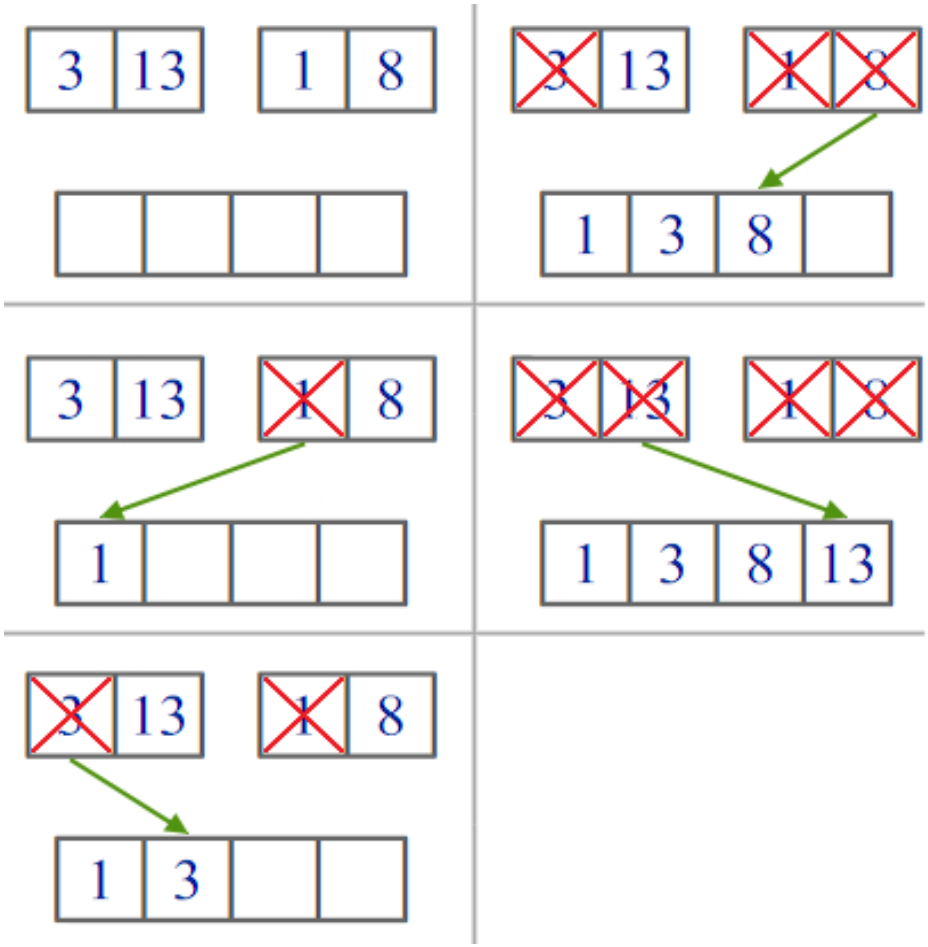


Рекурсивный вариант

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)

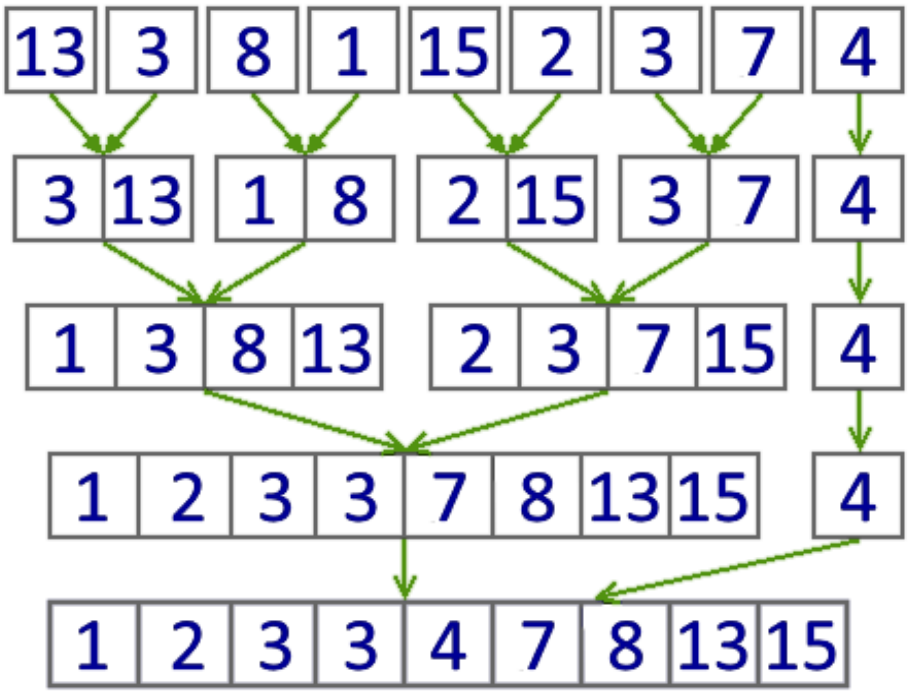
Merge



- 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



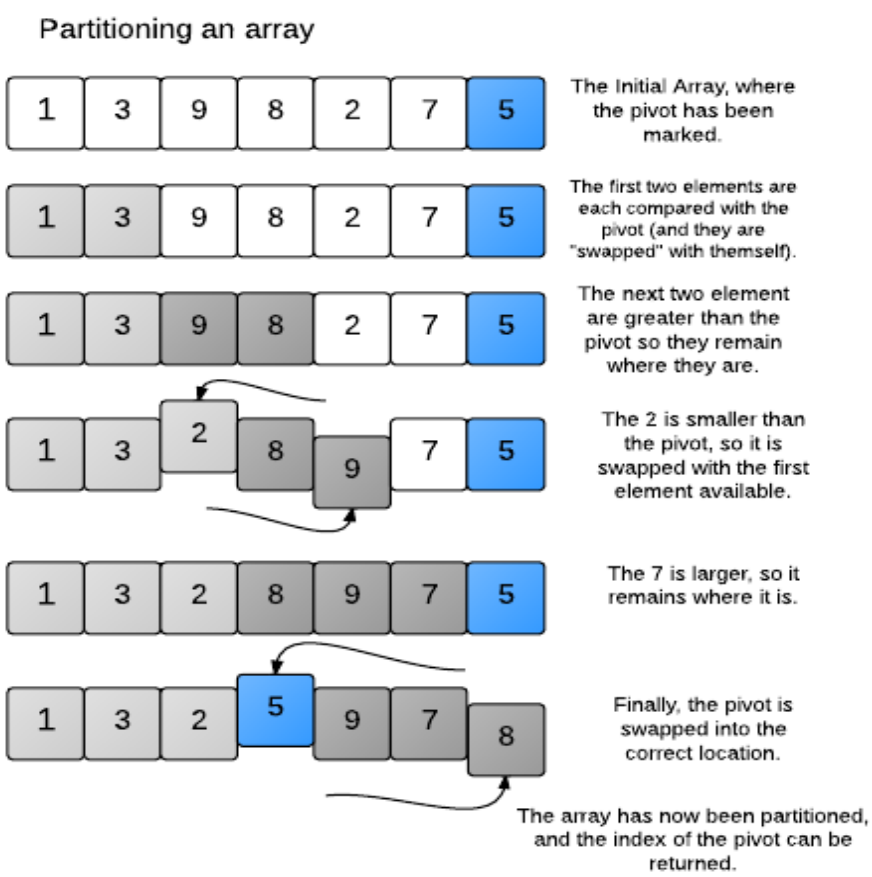
- Альтернатива: итеративный алгоритм
- Сложность? Устойчивость? Доп. память? Сложность на уже сортированных массивах?

Сортировка слиянием (MergeSort)

- $O(n * logn)$ в худшем случае
- $O(n * logn)$ на уже отсортированном массиве
- Устойчива
- Требует доп.памяти - по размеру исходного массива

Быстрая сортировка (QuickSort)

Partition

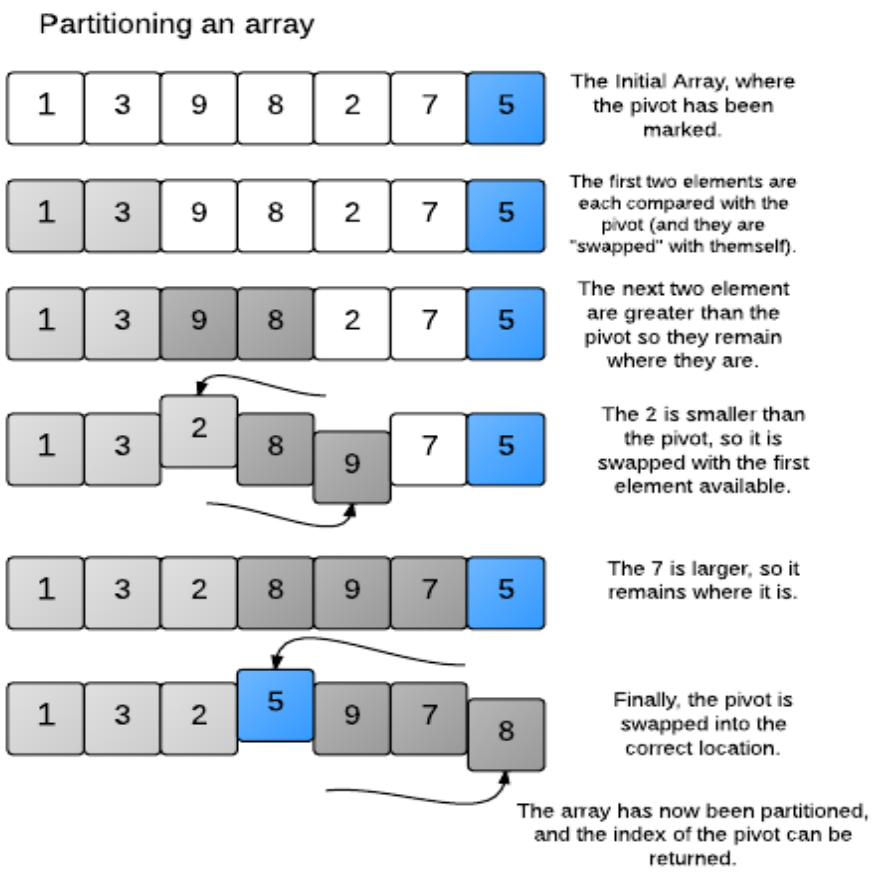


Идея - выбираем элемент, упорядочивая элементы массива относительно него

После этого мы имеем элемент на своем месте и нам остается отсортировать две подчасти.

Быстрая сортировка (QuickSort)

Partition

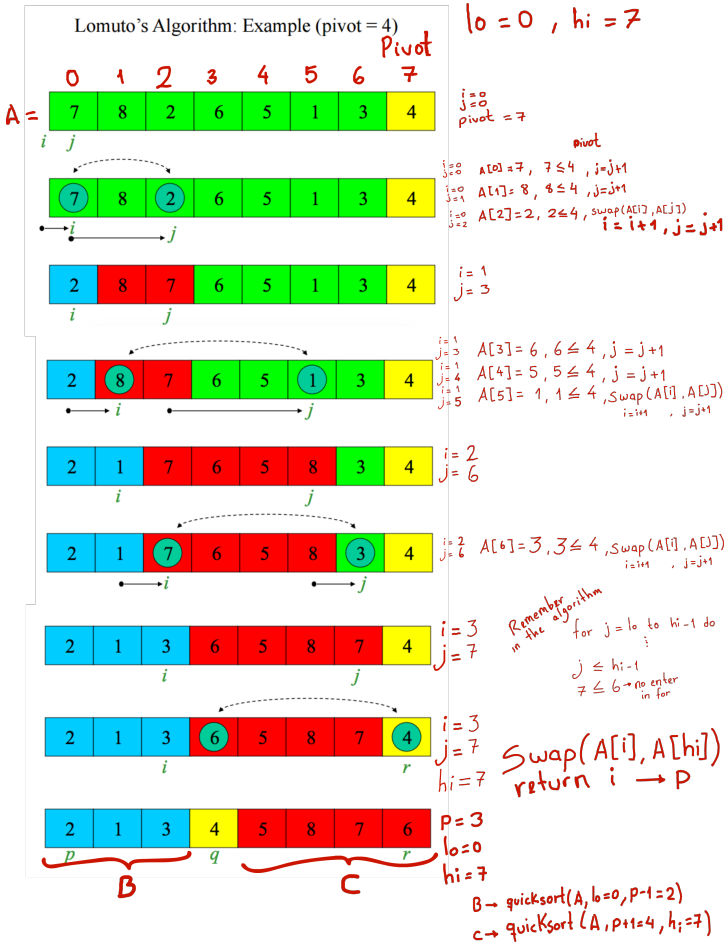


Операция Partition - выбираем опорный элемент, а остальные элементы делим на две части: меньше опорного и больше или равные опорному. $\Theta(n)$ Для обоих частей рекурсивно выполняем Partition

Основые тип операции Partition: Хоара или Ломуто

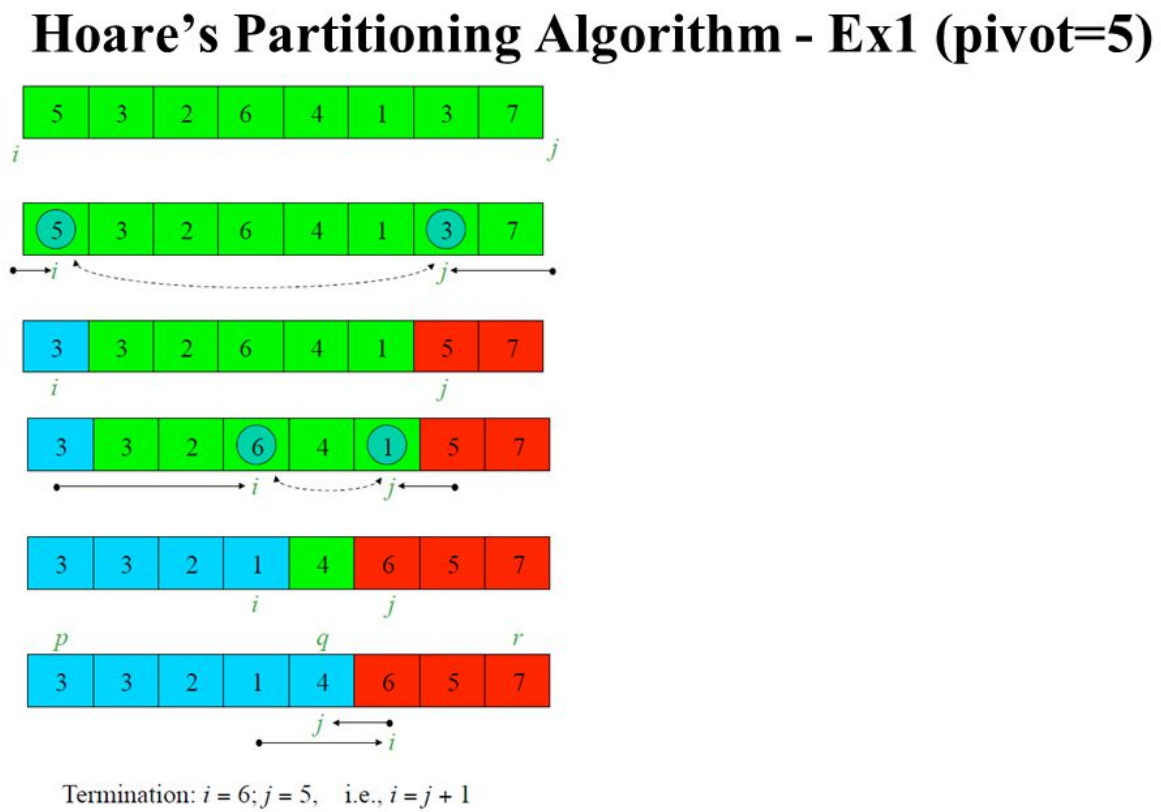
Быстрая сортировка (QuickSort)

Partition



Быстрая сортировка (QuickSort)

Partition



Быстрая сортировка (QuickSort)

Partition

BASIS	LOMUTO'S	HOARE'S
Implementation	Easier to implement	Comparatively harder to implement
Efficiency	Less efficient	Comparatively more efficient
Number of swaps and partitioning	Three times more swaps compared to Hoare's	Three times fewer swaps and creates efficient partitions
Time complexity when all elements are equal	$O(n^2)$	O(nlogn)
Time complexity when the array is already sorted	$O(n^2)$	$O(n^2)$

Быстрая сортировка (QuickSort)

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

Быстрая сортировка (QuickSort)

- $O(n^2)$ в худшем случае
- $O(n * logn)$ в среднем
- $O(n^2)$ на уже отсортированном массиве
- Неустойчива
- Не требует доп.памяти

Быстрая сортировка (QuickSort)

Модификации:

- Устойчивости можно добиться, но обычно это не делают. Можно через выделение дополнительной памяти, собирая элементы больше и меньше в отдельные массивы
- Выбор опорного элемента: первый, последний, средний, медианный из 3, случайный

Быстрая сортировка (QuickSort)

Оптимизации при реализации

Возможные оптимизации

1. Оптимизация выбора опорного элемента
   Медиана из трёх, например.
2. Оптимизация Partition
   Использовать разбиение Хоара (два индекса, которые приближаются друг к другу) вместо разбиения Ломуто
3. Написать с одной ветвью рекурсии
   Рекурсивный вызов вызывается для меньшего куска после разделения, 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 сортирову 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	$\Theta(nlog(n))$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	Нет	$\Theta(1)$
Merge	$\Theta(nlog(n))$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	Да	$\Theta(n)$
Quick	$\Theta(n^2)$	$\Theta(nlog(n))$	$\Theta(nlog(n))$	$\Theta(n^2)$	Нет	$\Theta(1)$

Библиотечные функции сортировки

std::sort

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

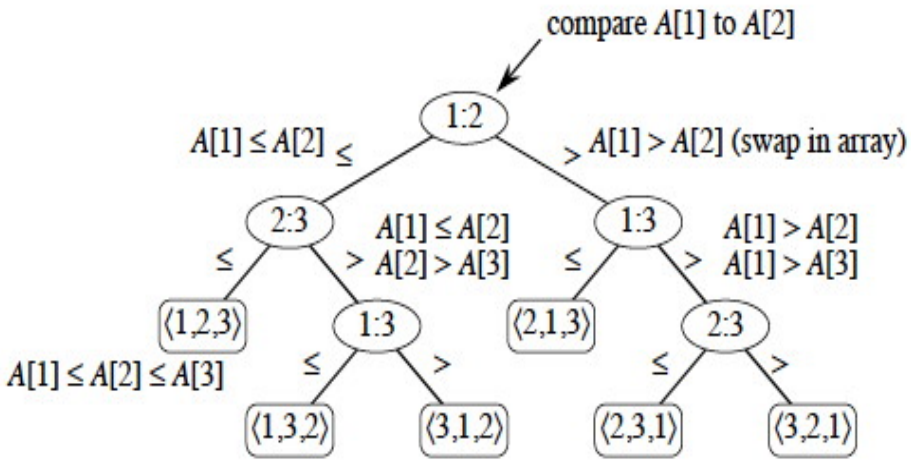
1. std::ios_base::sync_with_stdio(false); - отключение синхронизации C++ потоков и вывода C. Быстрее, но может приводить к перемешиванию вывода. Если включено, но нет буфферизации, и код потоко-безопасный.

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

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

2. 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 $\Omega(nlog(n))$ для сортировок сравнениями



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

Сортировки

Задача 1

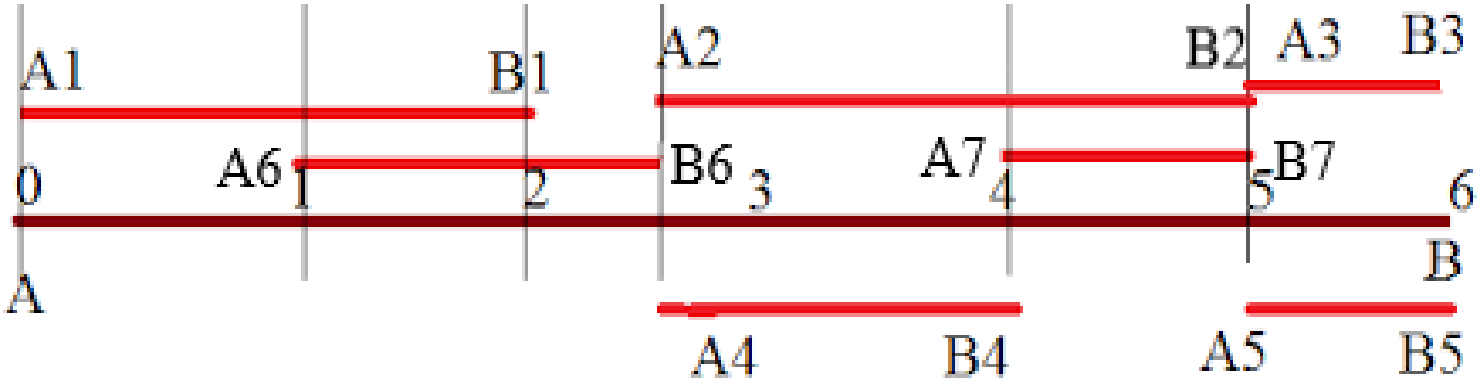
На прямой заданы n отрезков координатами своих концов $[a_i, b_i]$. Найдите

1. длину их объединения
2. длину их пересечения
3. максимальное количество отрезков, которое можно выбрать так, чтобы выбранные отрезки попарно не пересекались

Сортировки

Задача 1

1. Длина объединения



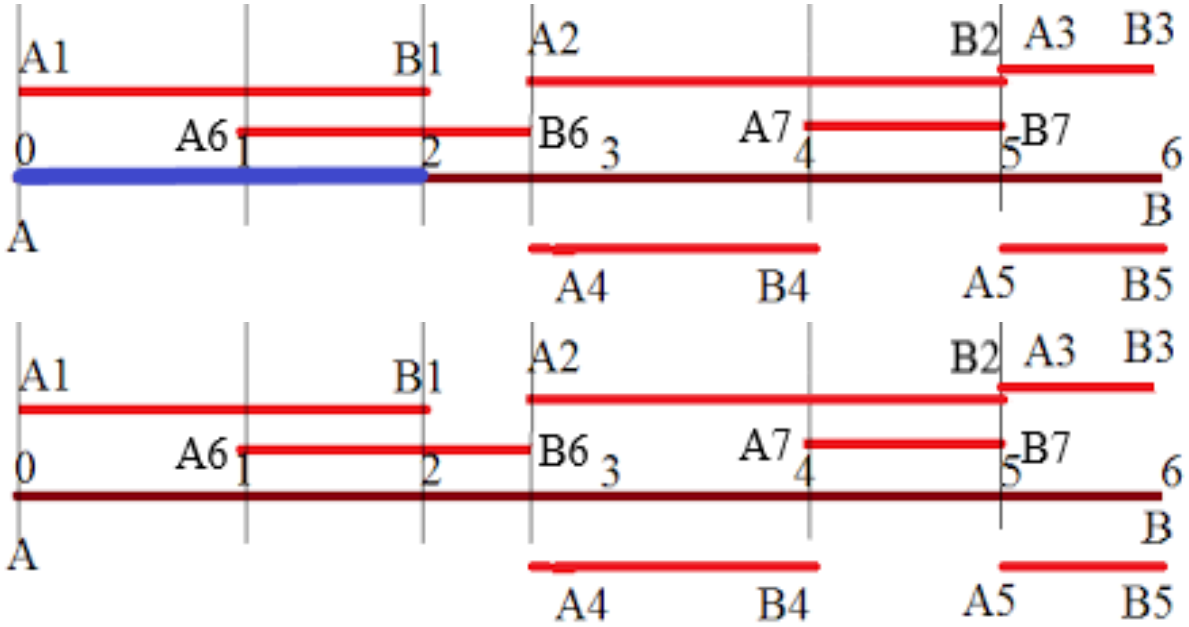

```
struct Point {
    double x;
    bool is_end;
} \\ (1, false) > (1, true)

std::vector<Point> points = transform(segments); // std::vector<Segment>
sort(points);
double sum = 0;
for (auto point: points)
    if (point.is_end)
        sum += point.x - prev_point.x;
    else
        if !(prev_point.is_end)
            sum += point.x - prev_point.x;
```

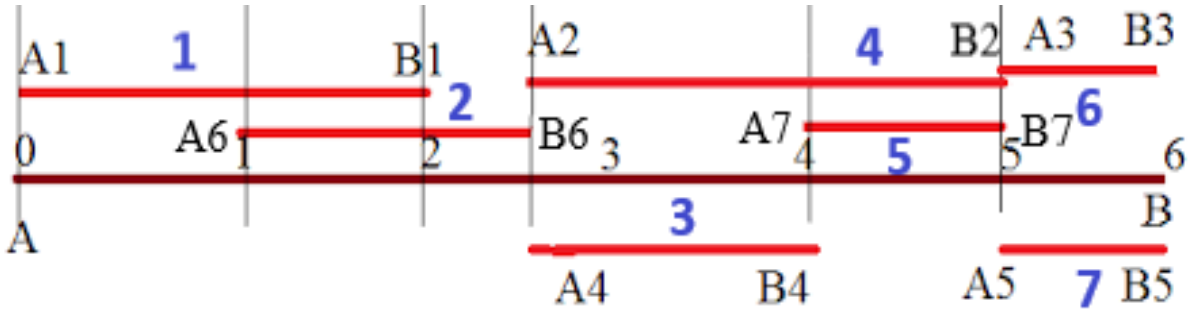
Сортировки

Задача 1

2. Длина пересечения



3. Длина пересечения



Быстрая сортировка (QuickSort)

Задача 2

Как реализовать Partition(A, x) с привлечением O(1) дополнительной памяти?

Быстрая сортировка (QuickSort)

Задача 2

```
void Partition(A, x)
{
    int i = 0, j = n - 1;
    while (i <= j) {
        while (A[i] < x) ++i;
        while (A[j] > x) --j;
        if (i <= j) {
            std::swap(A[i], A[j]);
            ++i;
            --j;
        }
    }
}
```

Полезные ссылки I

ICS 311 #10: Theoretical Limits of Sorting, O(n) Sorts
https://www2.hawaii.edu/ñodari/teaching/s17/Notes/Topic-10.html

Wiki - Sorting algorithm
https://en.wikipedia.org/wiki/Sorting_algorithm

Викиконспекты: Сортировка слиянием
https://bit.ly/2DH6XmF