Алгоритмы и структуры данных

Нижняя оценка алгоритмов сортировки, сортировки за линейное время

Перед тем, как перейдем в высшую лигу

Перед тем, как перейдем в высшую лигу

Вопрос: можем ли мы вообще отсортировать массив быстрее, чем за O(N*logN)?

Перед тем, как перейдем в высшую лигу

Вопрос: можем ли мы вообще отсортировать массив быстрее, чем за O(N*logN)?

Ответ: НЕТ

Перед тем, как перейдем в высшую лигу

Вопрос: можем ли мы вообще отсортировать массив быстрее, чем за O(N*logN)?

Ответ: НЕТ (для сортировок основанных на сравнении элементов)

Перед тем, как перейдем в высшую лигу

Вопрос: можем ли мы вообще отсортировать массив быстрее, чем за O(N*logN)?

Ответ: НЕТ (для сортировок основанных на сравнении элементов)

Т.е. у нас нет никакой дополнительной информации о свойствах элементов, мы просто раз за разом сравниваем их на >, < или ==

Нижняя оценка сложности сортировок сравнения

Пусть есть массив, содержащий числа 1,…,N в некотором порядке.

Нижняя оценка сложности сортировок сравнения

Пусть есть массив, содержащий числа 1,…,N в некотором порядке.

Корректная сортировка приведет такую последовательность к отсортированному виду, получив соответствующую перестановку

Нижняя оценка сложности сортировок сравнения

Пусть есть массив, содержащий числа 1,…,N в некотором порядке.

Корректная сортировка приведет такую последовательность к отсортированному виду, получив соответствующую перестановку

6	7	1	2	4	3	8	5
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

$\Longrightarrow$

1	2	3	4	5	6	7	8
a_2	a_3	a_5	a_4	a_7	a_0	a_1	a_6

Нижняя оценка сложности сортировок сравнения

Пусть есть массив, содержащий числа 1,…,N в некотором порядке.

Корректная сортировка приведет такую последовательность к отсортированному виду, получив соответствующую перестановку

6	7	1	2	4	3	8	5
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

$\Rightarrow$

1	2	3	4	5	6	7	8
a_2	a_3	a_5	a_4	a_7	a_0	a_1	a_6

$$< 2, 3, 5, 4, 7, 0, 1, 6 >$$

Нижняя оценка сложности сортировок сравнения

Пусть есть массив, содержащий числа 1,…,N в некотором порядке.

Корректная сортировка приведет такую последовательность к отсортированному виду, получив соответствующую перестановку

6	7	1	2	4	3	8	5
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

1	2	3	4	5	6	7	8
a_2	a_3	a_5	a_4	a_7	a_0	a_1	a_6

$$< 2, 3, 5, 4, 7, 0, 1, 6 >$$

Сколько таких перестановок может быть?

Нижняя оценка сложности сортировок сравнения

Пусть есть массив, содержащий числа 1,…,N в некотором порядке.

Корректная сортировка приведет такую последовательность к отсортированному виду, получив соответствующую перестановку

6	7	1	2	4	3	8	5
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

1	2	3	4	5	6	7	8
a_2	a_3	a_5	a_4	a_7	a_0	a_1	a_6

$$< 2, 3, 5, 4, 7, 0, 1, 6 >$$

Сколько таких перестановок может быть? **N!**

Дерево решений

На каждом шаге сортировки принимаем решение, сравнивая два элемента.

Дерево решений

На каждом шаге сортировки принимаем решение, сравнивая два элемента.

Пусть массив: $[a_0, a_1]$

Дерево решений

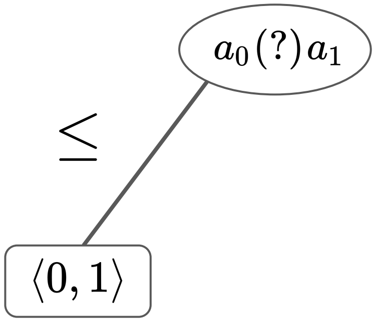
На каждом шаге сортировки принимаем решение, сравнивая два элемента.

$a_0 (?) a_1$

Пусть массив: $[a_0, a_1]$

Дерево решений

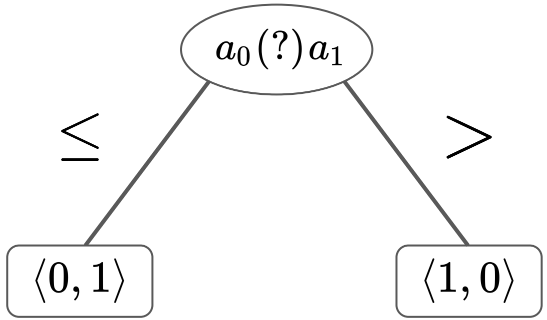
На каждом шаге сортировки принимаем решение, сравнивая два элемента.



Пусть массив: $[a_0, a_1]$

Дерево решений

На каждом шаге сортировки принимаем решение, сравнивая два элемента.

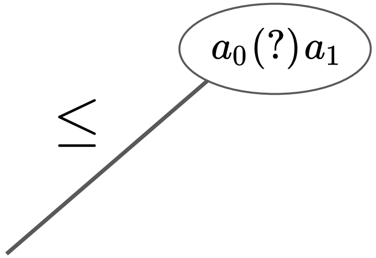


Пусть массив: $[a_0, a_1]$

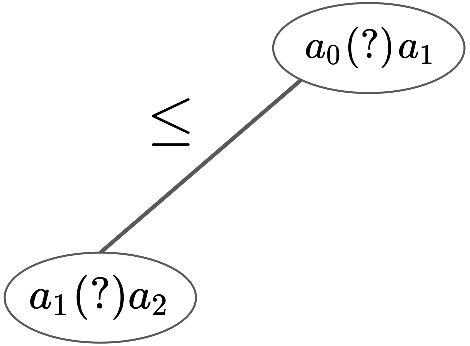
Пусть массив: $[a_0, a_1, a_2]$

$a_0 (?) a_1$

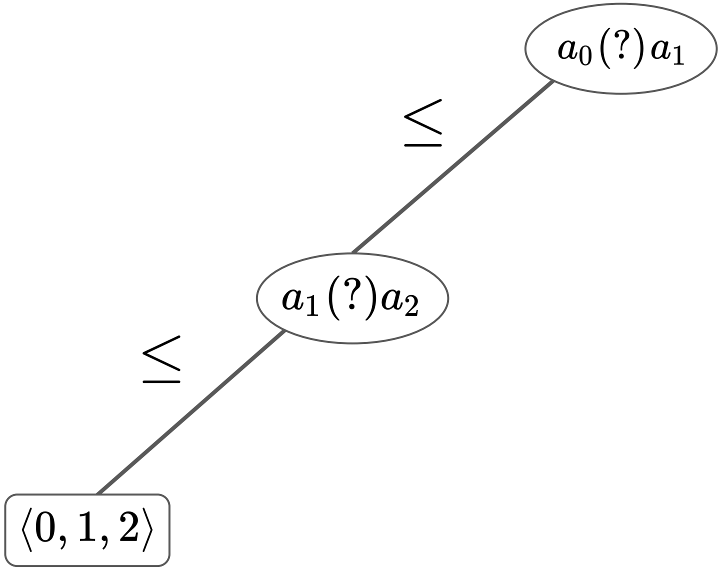
Пусть массив: $[a_0, a_1, a_2]$



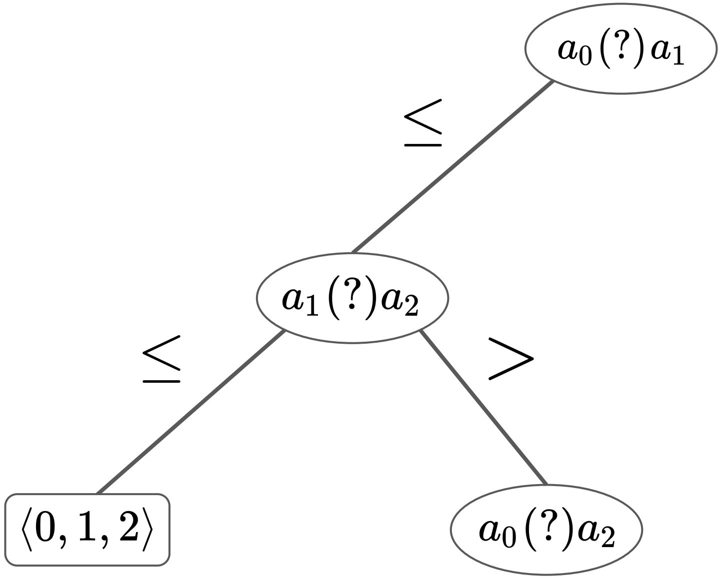
Пусть массив: $[a_0, a_1, a_2]$



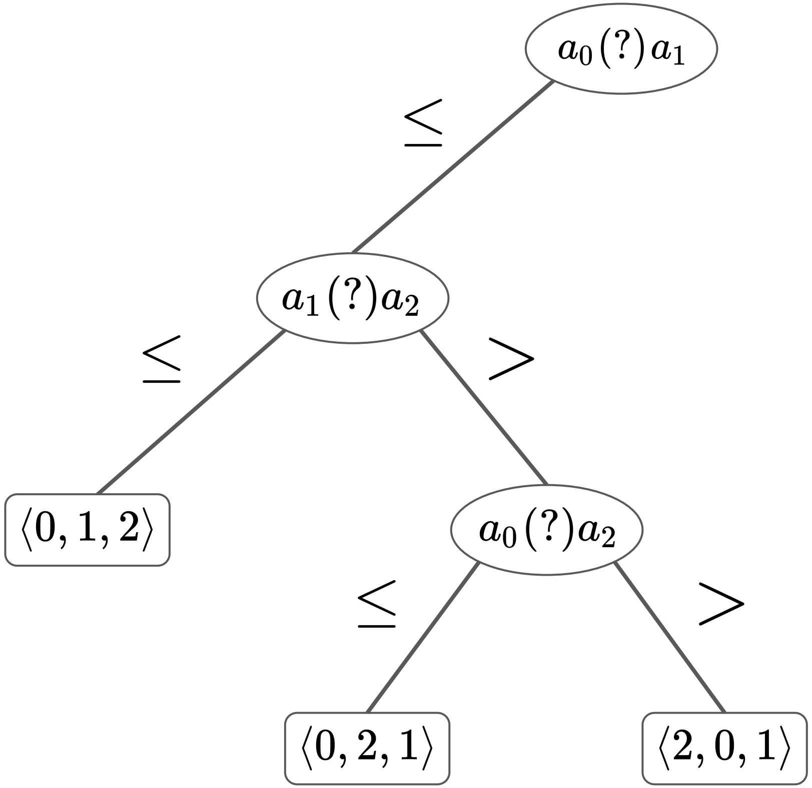
Пусть массив: $[a_0, a_1, a_2]$



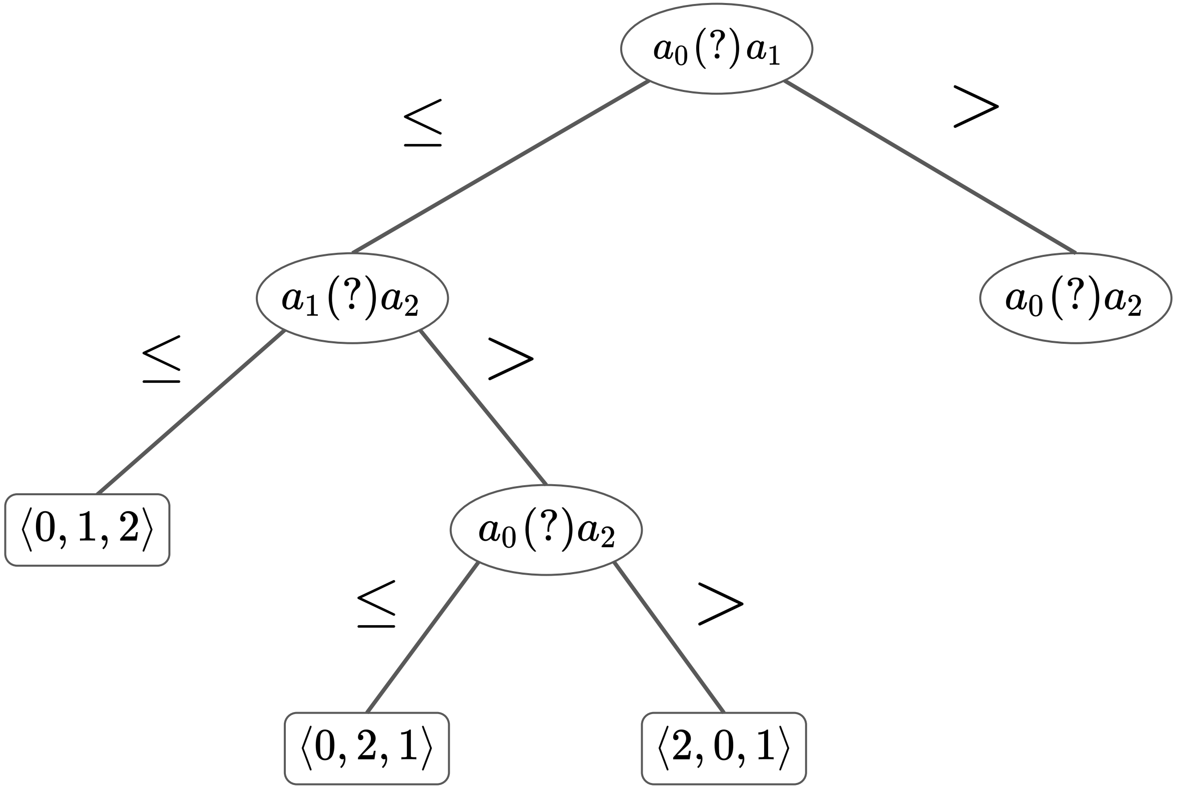
Пусть массив: $[a_0, a_1, a_2]$



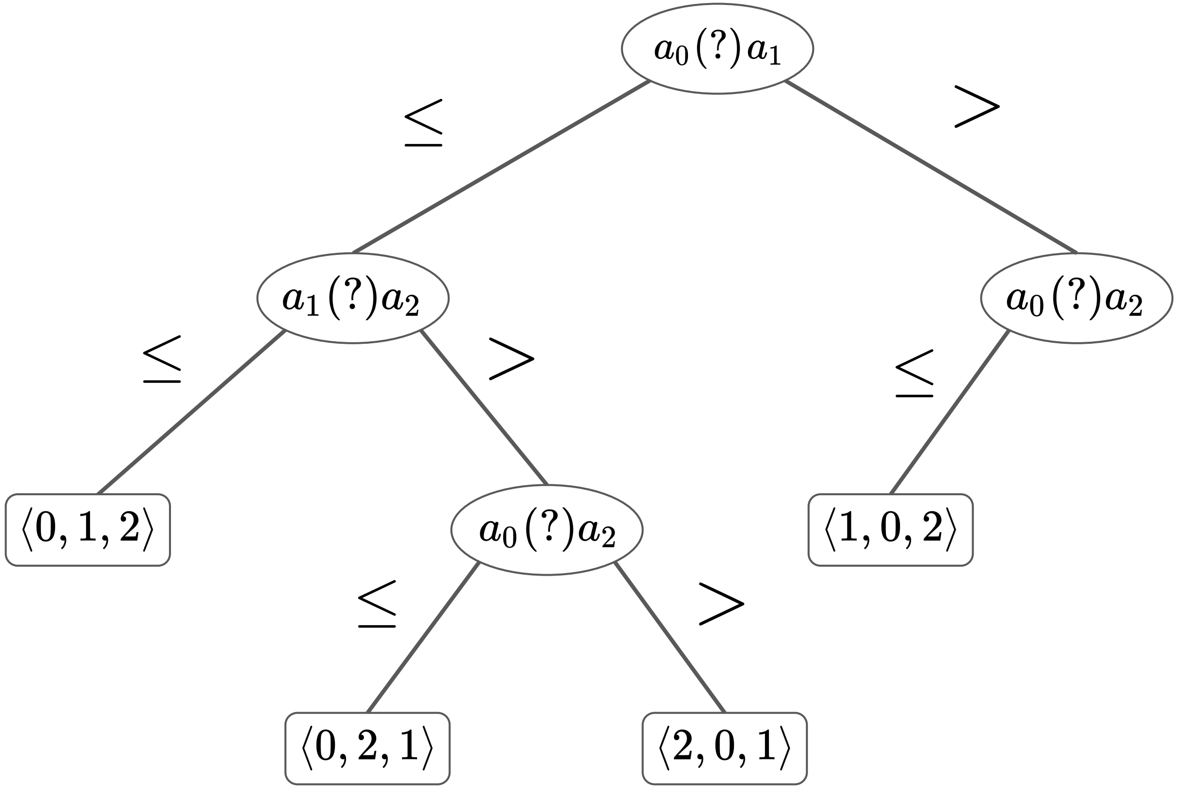
Пусть массив: $[a_0, a_1, a_2]$



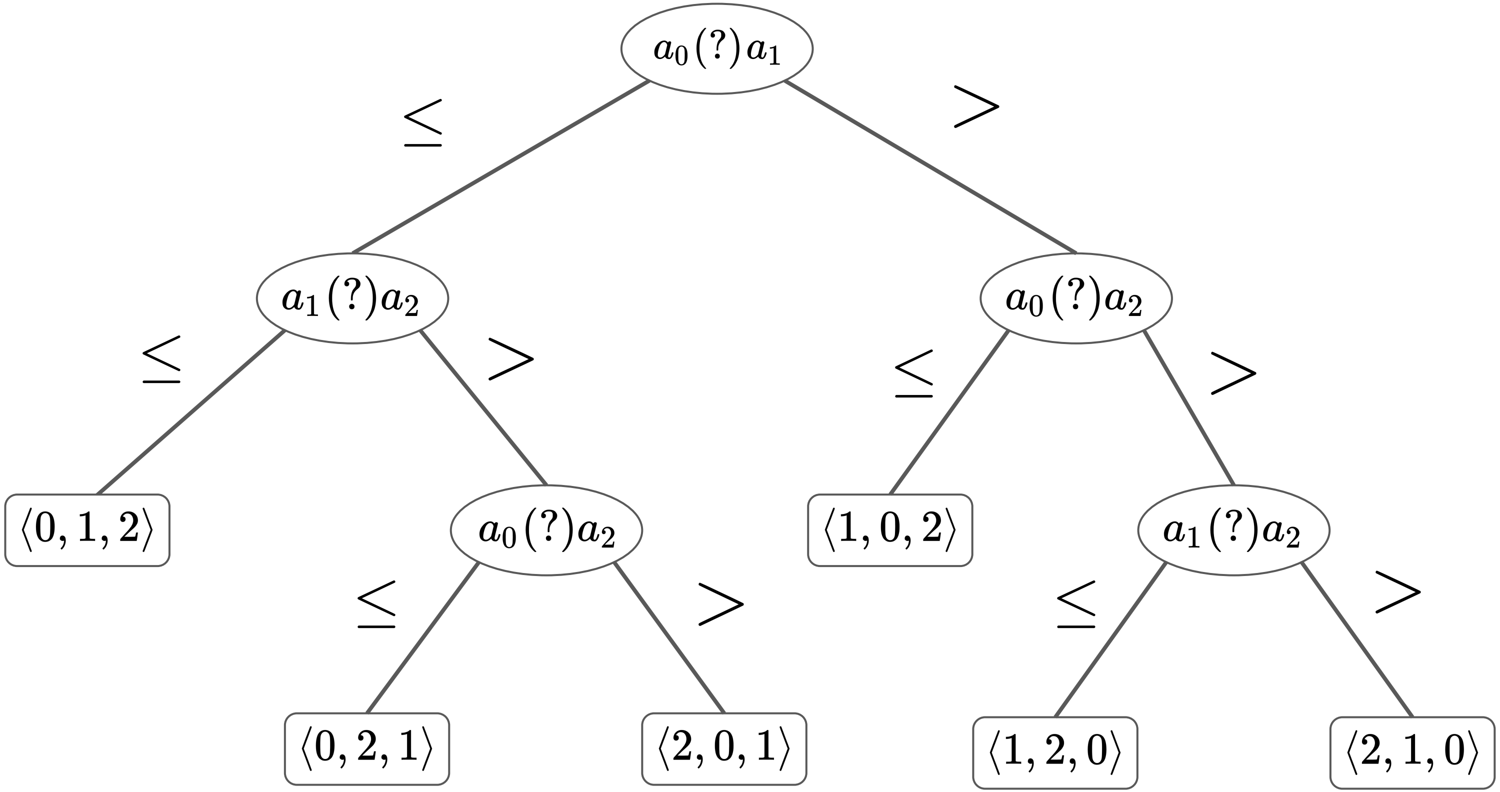
Пусть массив: $[a_0, a_1, a_2]$



Пусть массив: $[a_0, a_1, a_2]$

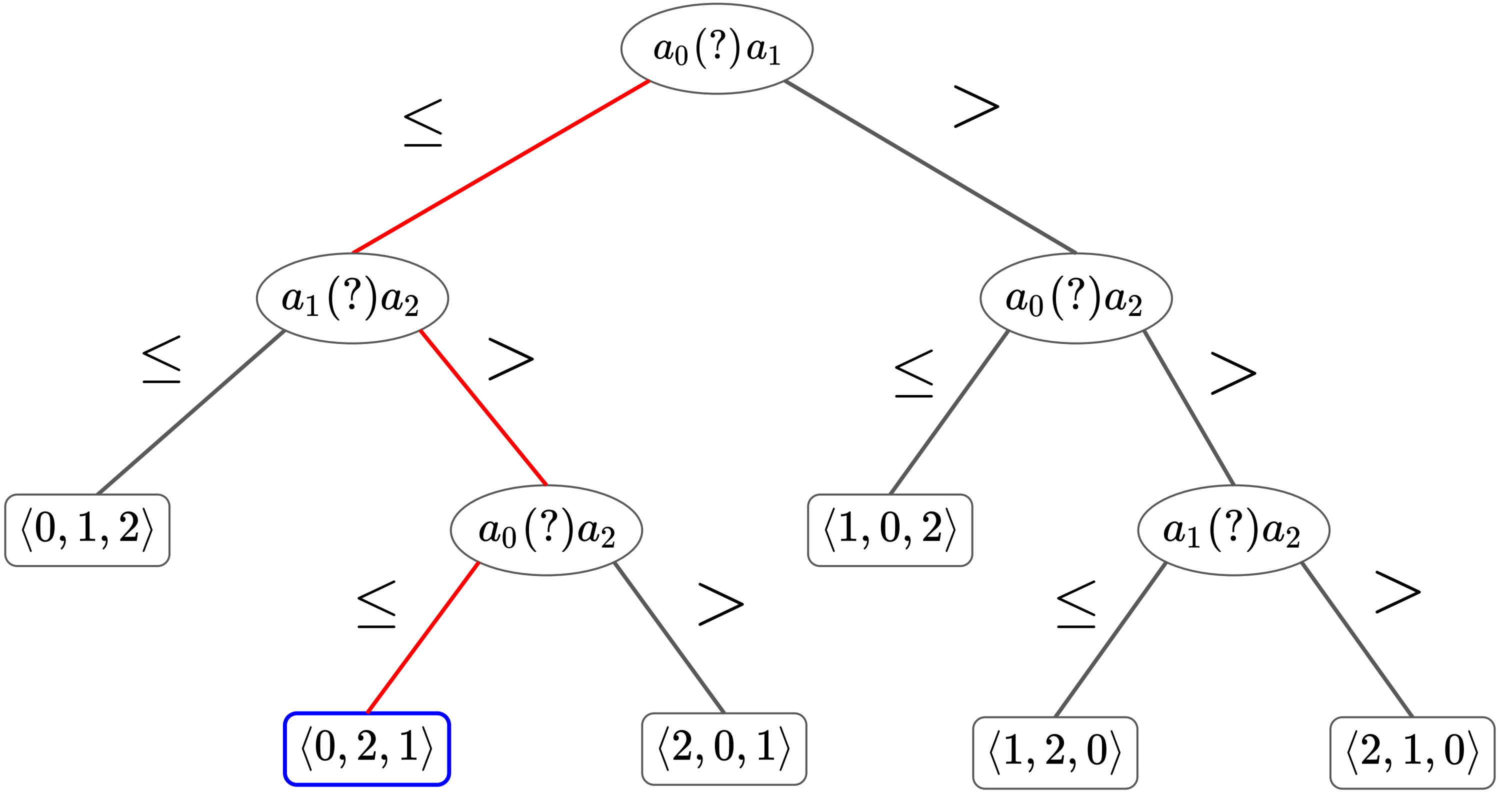


Пусть массив: $[a_0, a_1, a_2]$

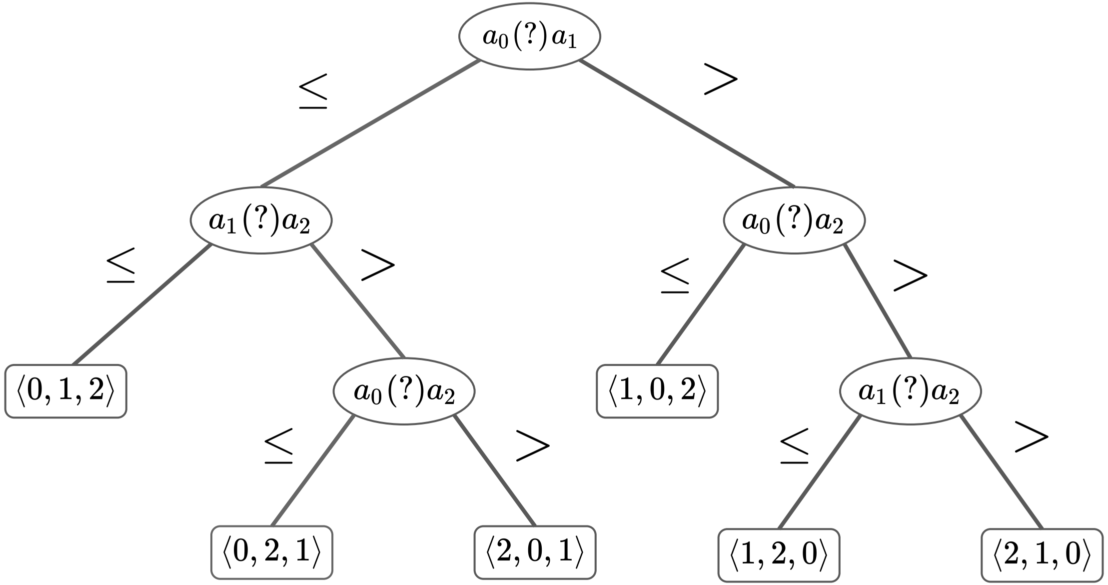


Пусть массив: $[a_0, a_1, a_2]$

$[16, 29, 18]$

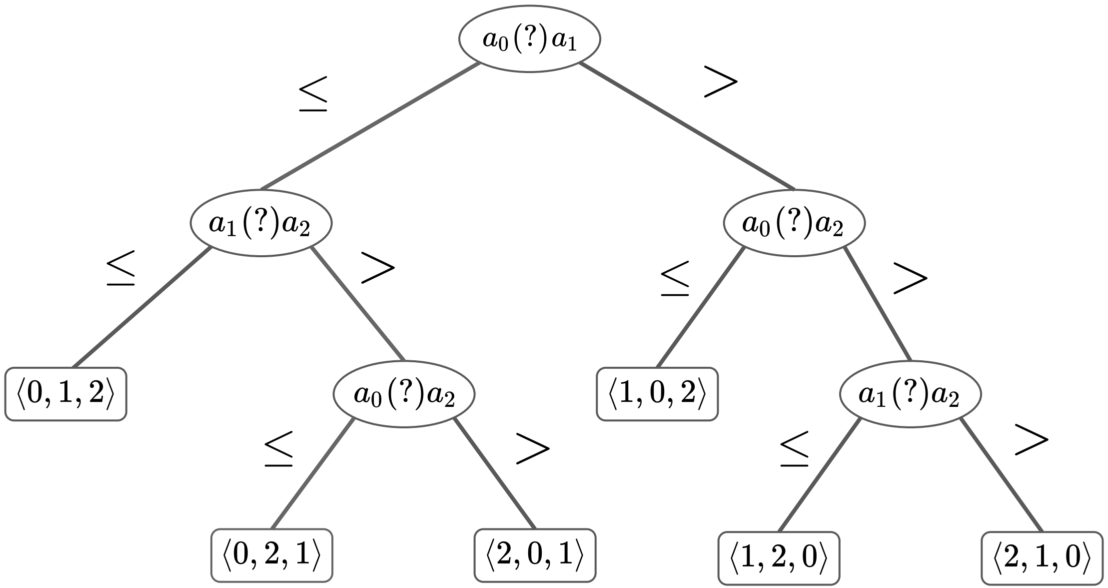


Дерево решений



Все возможные варианты работы сортировки сравнений образуют дерево решений.

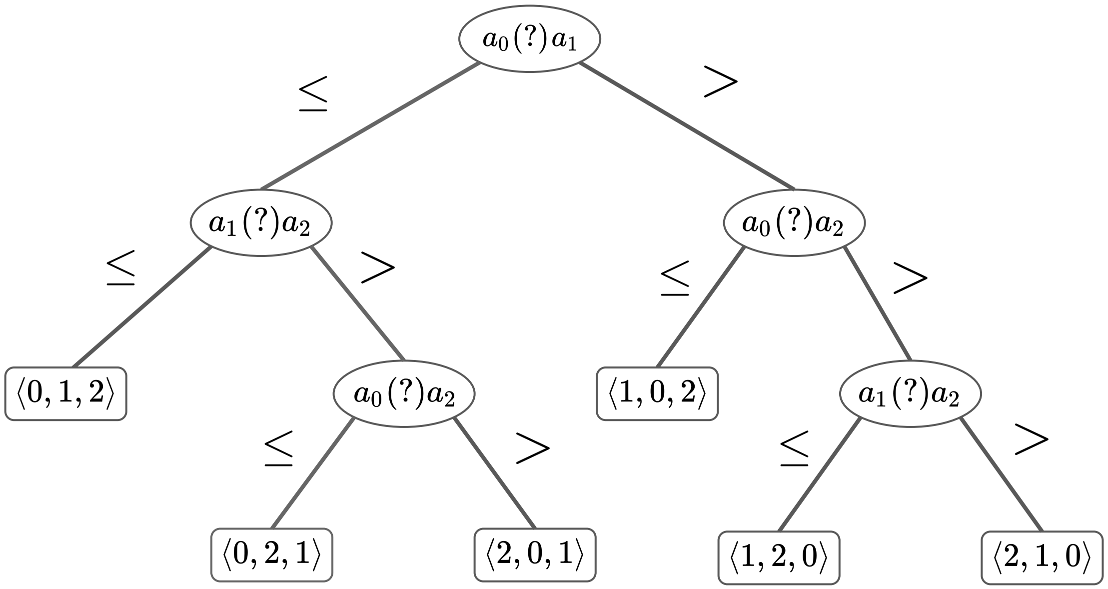
Дерево решений



Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

Дерево решений

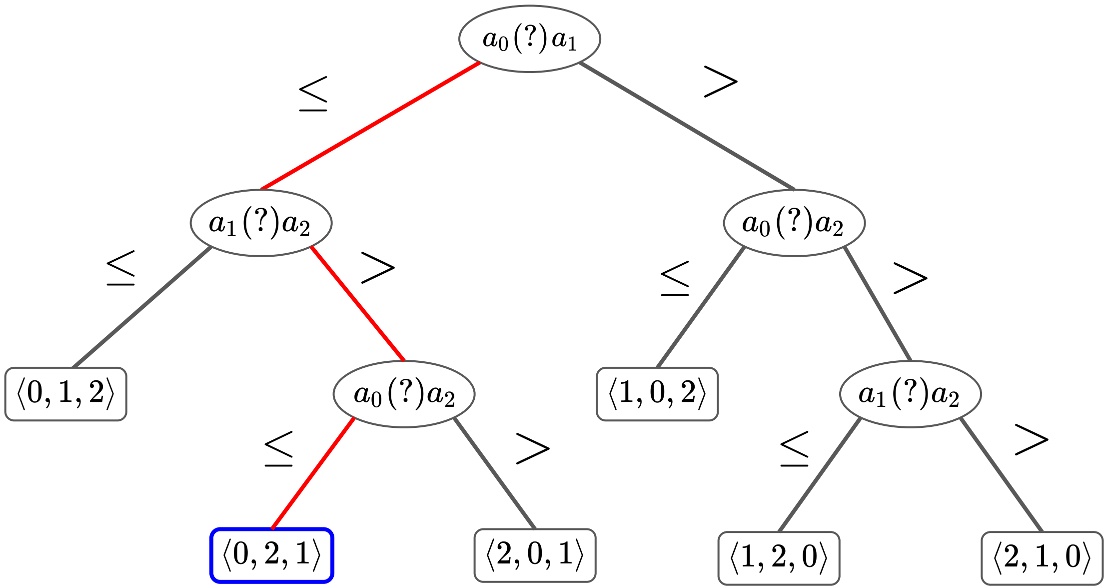


Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

Какой худший случай у алгоритма сортировки?

Дерево решений



Все возможные варианты работы сортировки сравнений образуют дерево решений.

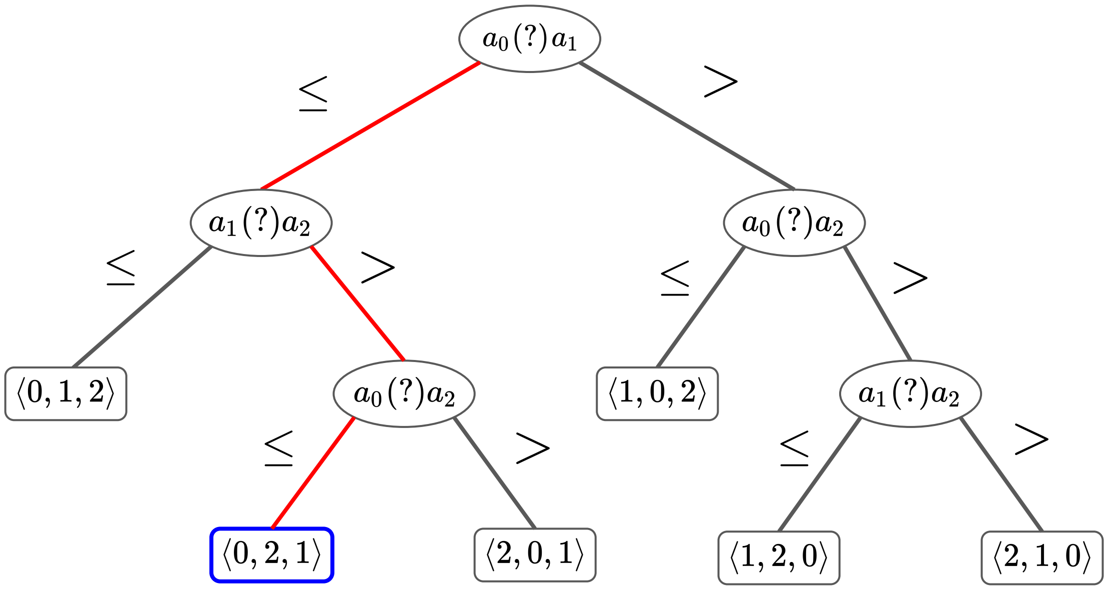
Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

Какой худший случай у алгоритма сортировки?

Прошли до самого нижнего уровня!

(сделали максимальное количество сравнений - k)

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

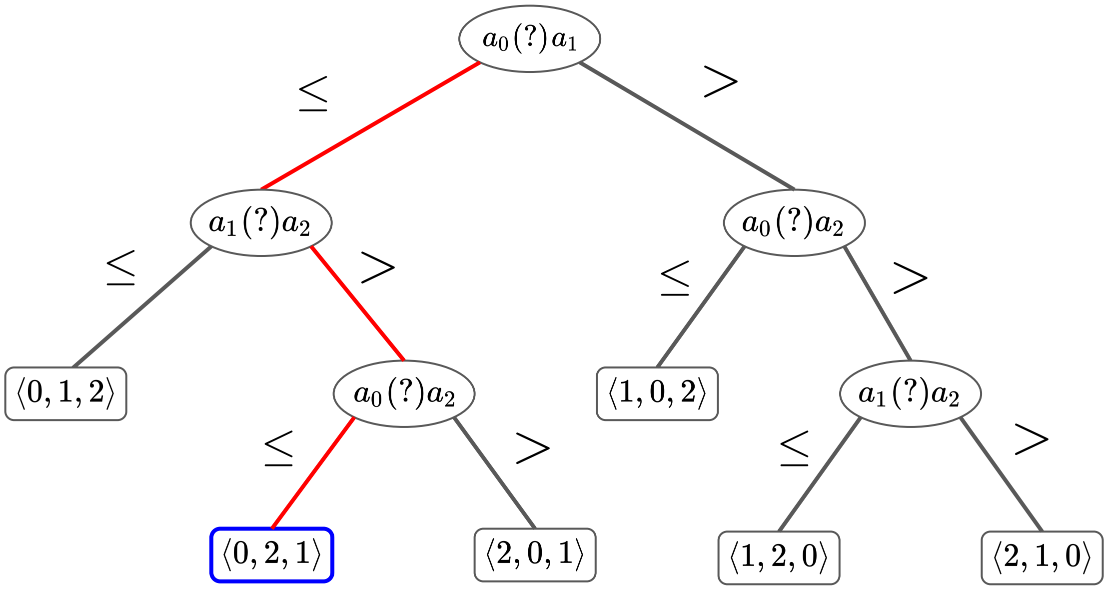
Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

Какой худший случай у алгоритма сортировки?

Прошли до самого нижнего уровня!

(сделали максимальное количество сравнений - k)

Дерево решений



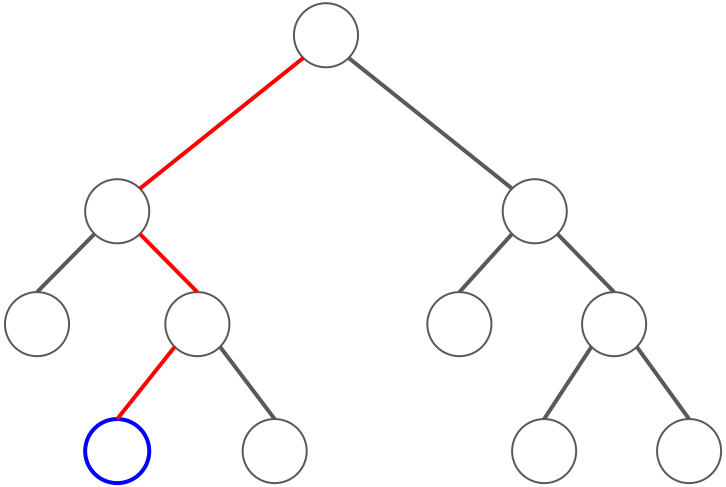
Максимальное количество сравнений - k (совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

А сколько у бинарного дерева высоты k может быть листьев?

Дерево решений



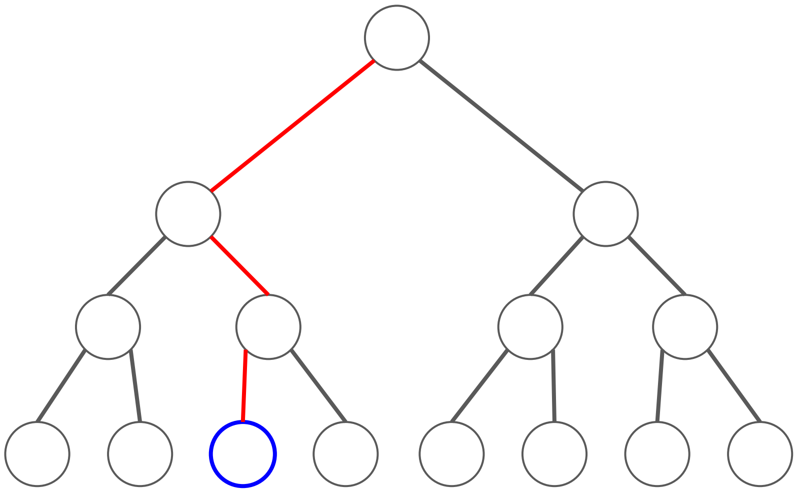
Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

А сколько у бинарного дерева высоты k может быть листьев?

Дерево решений



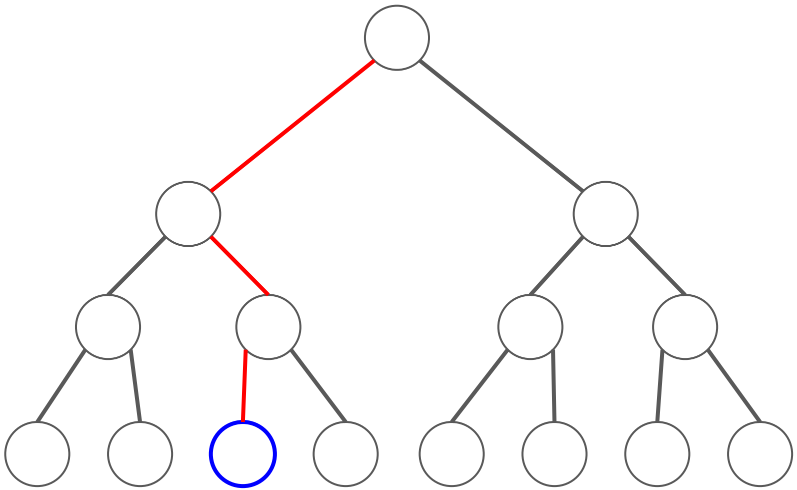
Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

А сколько у бинарного дерева высоты k может быть листьев?

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

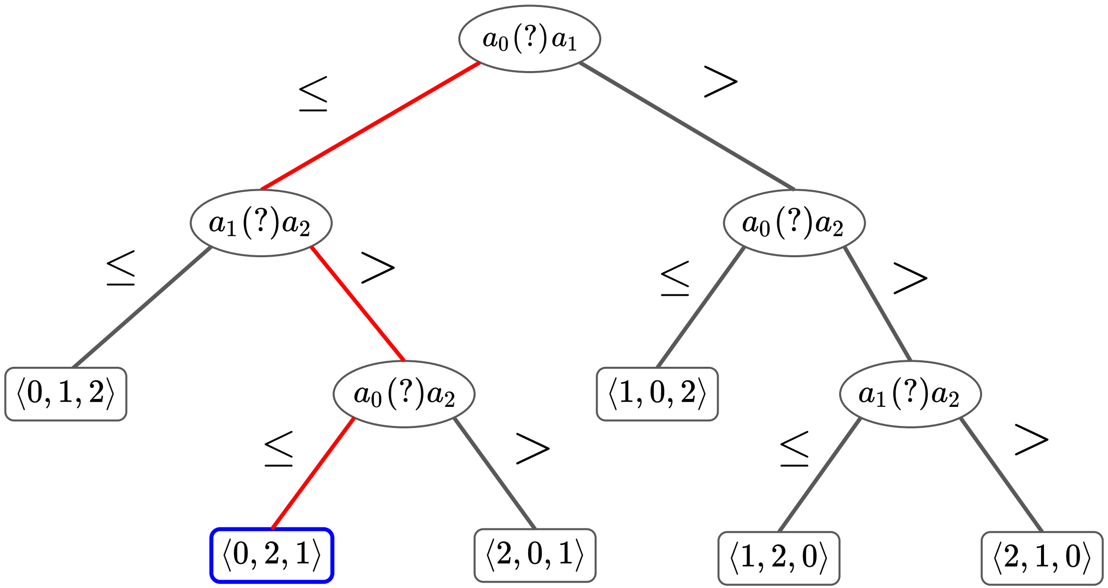
Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

А сколько у бинарного дерева высоты k может быть листьев?

#(листьев) $\leq 2^k$

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

А сколько у бинарного дерева высоты k может быть листьев?

#(листьев) $\leq 2^k$

Нижняя оценка сложности сортировок сравнения

Пусть есть массив, содержащий числа 1,…,N в некотором порядке.

Корректная сортировка приведет такую последовательность к отсортированному виду, получив соответствующую перестановку

6	7	1	2	4	3	8	5
a_0	a_1	a_2	a_3	a_4	a_5	a_6	a_7

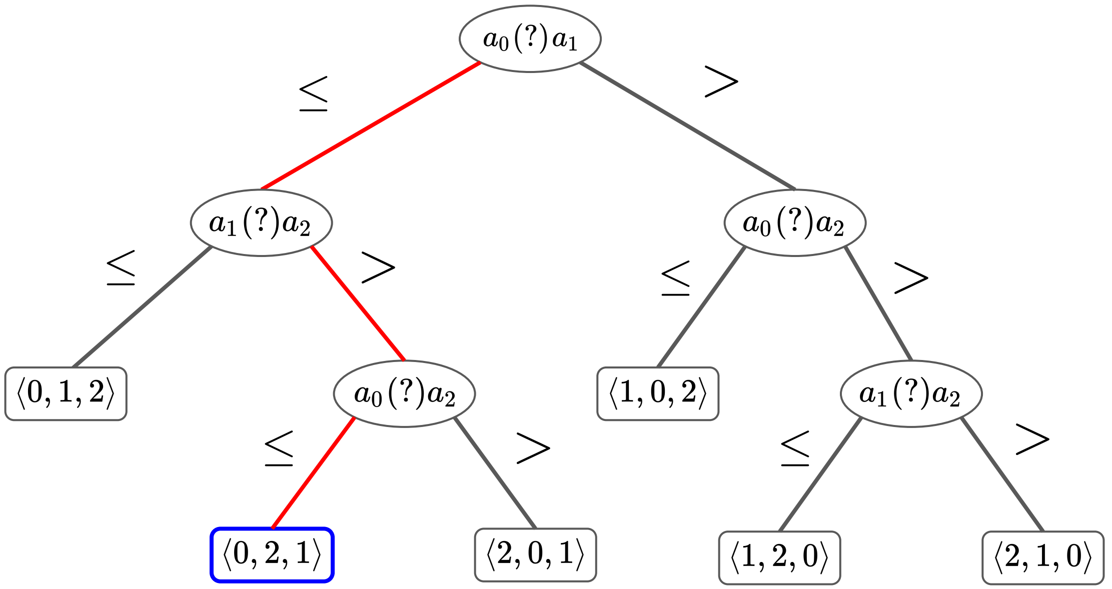
$$\Longrightarrow$$

1	2	3	4	5	6	7	8
a_2	a_3	a_5	a_4	a_7	a_0	a_1	a_6

$$< 2, 3, 5, 4, 7, 0, 1, 6 >$$

Сколько таких перестановок может быть? **N!**

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

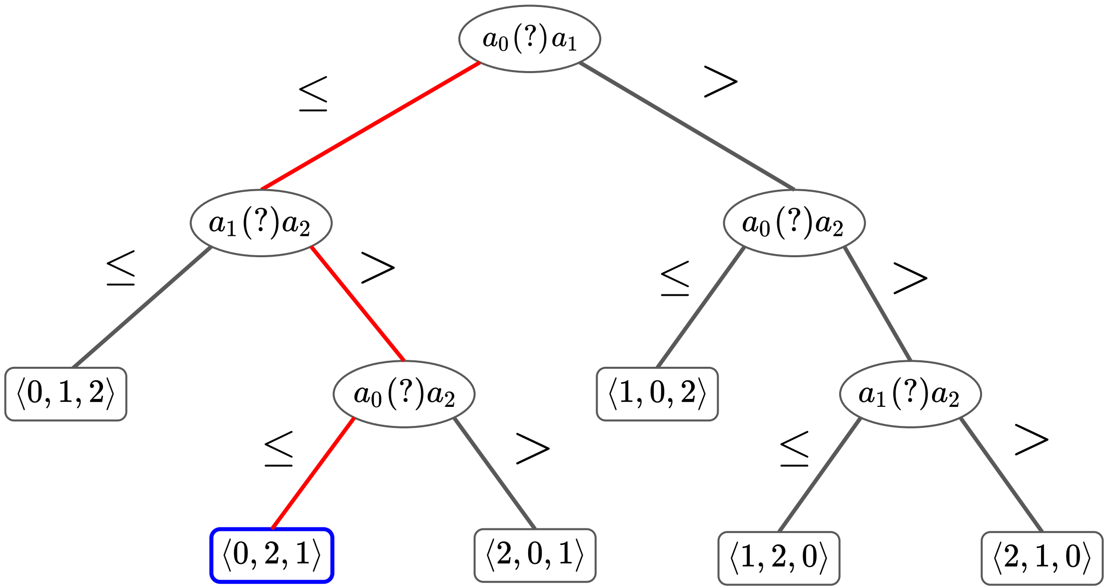
Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

#(листьев) $\leq 2^k$

Но при этом каждый лист - перестановка.

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

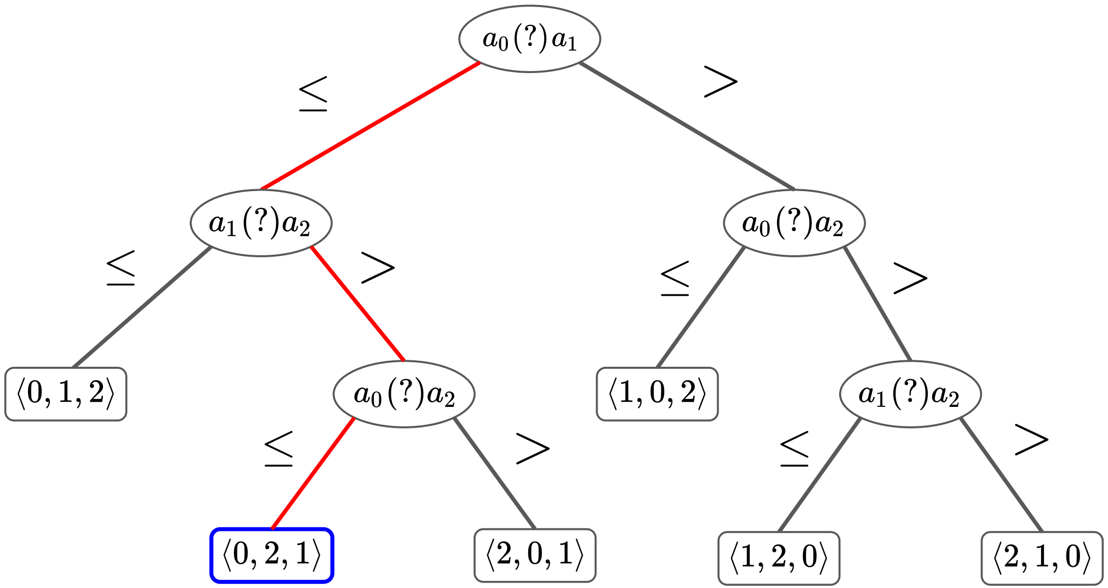
Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

#(листьев) $\leq 2^k$

Но при этом каждый лист - перестановка.

И для корректной сортировки их не может быть меньше N!

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

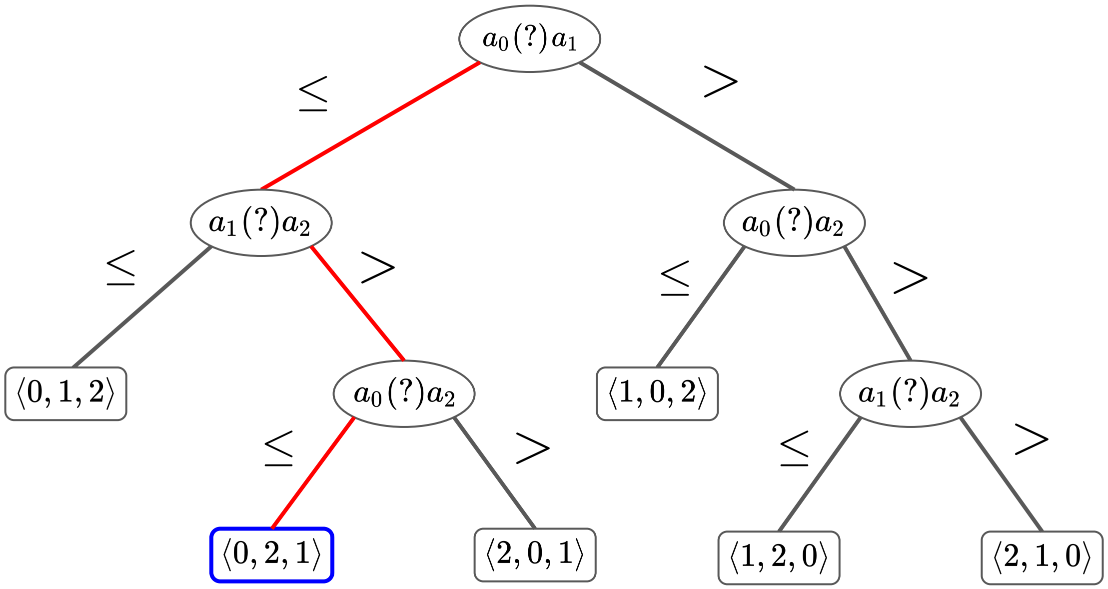
#(листьев) $\leq 2^k$

Но при этом каждый лист - перестановка.

И для корректной сортировки их не может быть меньше N!

Т.к. любую перестановку можно получить на некоторых входных данных.

Дерево решений



Максимальное количество сравнений - k
(совпадает с количеством уровней дерева, высотой)

Все возможные варианты работы сортировки сравнений образуют дерево решений.

Это бинарное дерево, т.е. у каждой вершины не больше двух детей.

#(листьев) $\leq 2^k$

Но при этом каждый лист - перестановка.

И для корректной сортировки их не может быть меньше N!

#(листьев) $\geq N!$

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

#(листьев) $\leq 2^k$

#(листьев) $\geq N!$

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

$$\#(\text{листьев}) \leq 2^k$$
$$\#(\text{листьев}) \geq N! \quad \Rightarrow \quad 2^k \geq N!$$

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

$$\#(\text{листьев}) \le 2^k \qquad \Rightarrow 2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2}\right) \cdot \ldots \cdot 1 \Rightarrow$$

$$\#(\text{листьев}) \ge N!$$

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

$$\#(\text{листьев}) \le 2^k$$
$$\#(\text{листьев}) \ge N!$$
$$\Rightarrow 2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2}\right) \cdot \ldots \cdot 1 \Rightarrow$$

$$2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2} + 1\right) \Rightarrow$$

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

$$\#(\text{листьев}) \le 2^k$$
$$\#(\text{листьев}) \ge N! \quad \Rightarrow \quad 2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2}\right) \cdot \ldots \cdot 1 \Rightarrow$$

$$2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2} + 1\right) \Rightarrow$$

$$2^k \ge \left(\frac{N}{2}\right)^{\frac{N}{2}}$$

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

$$\#(\text{листьев}) \le 2^k$$
$$\#(\text{листьев}) \ge N!$$
$$\Rightarrow 2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2}\right) \cdot \ldots \cdot 1 \Rightarrow$$

$$2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2} + 1\right) \Rightarrow$$

$$2^k \ge \left(\frac{N}{2}\right)^{\frac{N}{2}} \Rightarrow k \ge \left(\frac{N}{2}\right) \cdot \log\left(\frac{N}{2}\right)$$

Пусть N - размер входных данных,

k - максимальное количество сравнений для входных данных такого размера

$$\#(\text{листьев}) \le 2^k$$

$$\#(\text{листьев}) \ge N!$$

$$\Rightarrow 2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2}\right) \cdot \ldots \cdot 1 \Rightarrow$$

$$2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2} + 1\right) \Rightarrow$$

$$2^k \ge \left(\frac{N}{2}\right)^{\frac{N}{2}} \Rightarrow k \ge \left(\frac{N}{2}\right) \cdot \log\left(\frac{N}{2}\right) \Rightarrow k = \Omega(N \cdot \log(N))$$

Пусть N - размер входных данных,
k - максимальное количество сравнений для входных данных такого размера

$$\begin{array}{l}\#(\text{листьев}) \le 2^k \\ \#(\text{листьев}) \ge N!\end{array} \Rightarrow 2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2}\right) \cdot \ldots \cdot 1 \Rightarrow$$

$$2^k \ge N \cdot (N-1) \cdot (N-2) \cdot \ldots \cdot \left(\frac{N}{2} + 1\right) \Rightarrow$$

$$2^k \ge \left(\frac{N}{2}\right)^{\frac{N}{2}} \Rightarrow k \ge \left(\frac{N}{2}\right) \cdot \log\left(\frac{N}{2}\right) \Rightarrow k = \Omega(N \cdot \log(N)) \quad \square$$

$T(n) \Leftrightarrow \Omega(f(n))$, *если* $\exists C > 0,\ k\colon \forall n > k \Rightarrow C \cdot f(n) \le T(n)$

Перед тем, как перейдем в высшую лигу

Вопрос: можем ли мы вообще отсортировать массив быстрее, чем за O(N*logN)?

Ответ: НЕТ (для сортировок основанных на сравнении элементов)

Следствие: сложность merge sort — $\Theta(n * logn)$

Перед тем, как перейдем в высшую лигу

Вопрос: можем ли мы вообще отсортировать массив быстрее, чем за O(N*logN)?

Ответ: НЕТ (для сортировок основанных на сравнении элементов)

Следствие: сложность merge sort — $\Theta(n * logn)$

Так и что, все, победа? Быстрее никак?

Сортировка подсчетом (counting sort)

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =	4	5	1	2	2	1	1	0

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

		K-1						
A[N] =	4	5	1	2	2	1	1	0

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

K-1

A[N] =

4	5	1	2	2	1	1	0

Заводим массив C[K]

C[K] =

0	0	0	0	0	0
C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

K-1

A[N] =	4	5	1	2	2	1	1	0

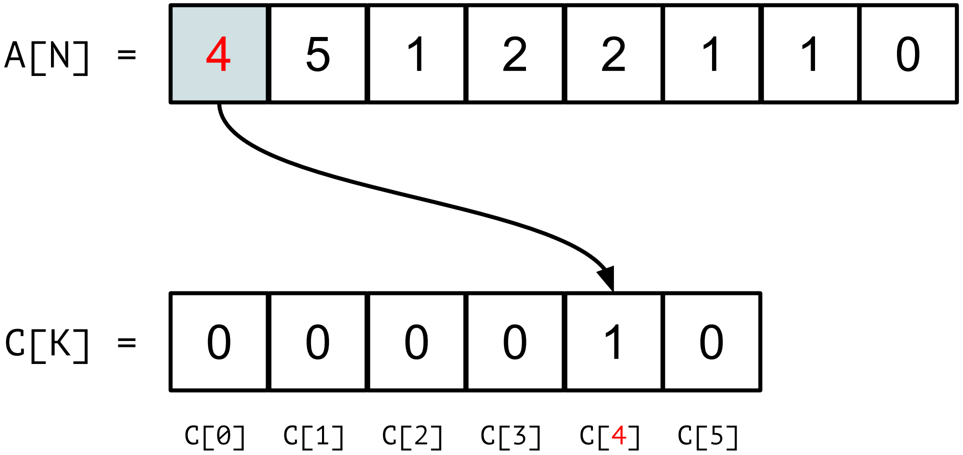
Заводим массив C[K]

C[K] =	0	0	0	0	0	0
	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

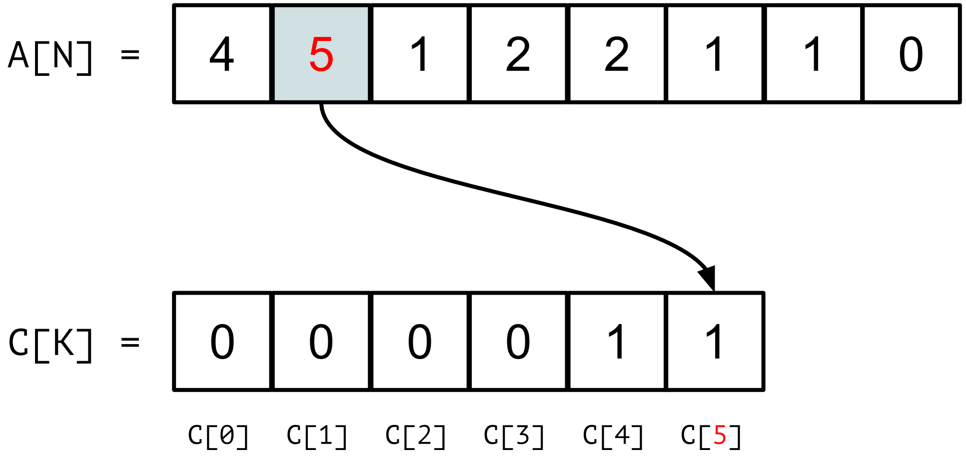
Хотим отсортировать массив состоящий из целых чисел 0, … , K-1



Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1



Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	1	0	0	1	1

Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =	4	5	1	2	2	1	1	0

C[K] =	0	1	1	0	1	1
	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

Проходим по исходному массиву и "считаем" элементы

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	1	2	0	1	1

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	2	2	0	1	1

Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =	4	5	1	2	2	1	1	0

C[K] =	0	3	2	0	1	1
	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =	4	5	1	2	2	1	1	0

C[K] =	1	3	2	0	1	1
	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

Проходим по исходному массиву и "считаем" элементы

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

R[N] =

Проходим по C и строим отсортированный массив

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

Проходим по C и строим отсортированный массив

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

R[N] =

0							

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

R[N] =

0	1	1	1				

Проходим по C и строим отсортированный массив

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

R[N] =

0	1	1	1	2	2		

Проходим по C и строим отсортированный массив

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

1	3	2	0	1	1
C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

R[N] =

0	1	1	1	2	2		

Проходим по C и строим отсортированный массив

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

R[N] =

0	1	1	1	2	2	4	

Проходим по C и строим отсортированный массив

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

Проходим по C и строим отсортированный массив

R[N] =

0	1	1	1	2	2	4	5

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =	4	5	1	2	2	1	1	0

C[K] =	1	3	2	0	1	1
	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

R[N] =	0	1	1	1	2	2	4	5

Проходим по C и строим отсортированный массив

Сложность?

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

1	3	2	0	1	1
C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

R[N] =

0	1	1	1	2	2	4	5

Проходим по C и строим отсортированный массив

Сложность?

O(N + K)

Сортировка подсчетом (counting sort)

Хотим отсортировать массив состоящий из целых чисел 0, … , K-1

A[N] =

4	5	1	2	2	1	1	0

C[K] =

1	3	2	0	1	1
C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

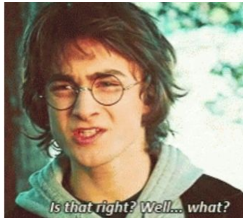
R[N] =

0	1	1	1	2	2	4	5

Проходим по C и строим отсортированный массив

Сложность?

O(N + K)

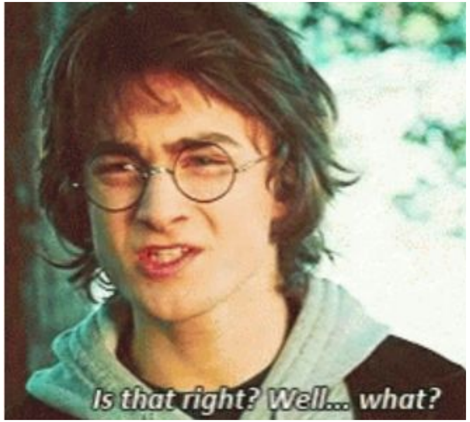


а если K = O(N), то сложность O(N)

Сортировка подсчетом (counting sort)

Стоп! Почему это сработало?

Была ведь теорема, что нельзя быстрее, чем за O(N*logN)!



Перед тем, как перейдем в высшую лигу

Вопрос: можем ли мы вообще отсортировать массив быстрее, чем за O(N*logN)?

Ответ: НЕТ (для сортировок основанных на сравнении элементов)

Т.е. у нас нет никакой дополнительной информации о свойствах элементов, мы просто раз за разом сравниваем их на >, < или ==

Сортировка подсчетом (counting sort)

Стоп! Почему это сработало?

Была ведь теорема, что нельзя быстрее, чем за O(N*logN)!

Потому, что мы ничего не сравнивали здесь, только обращались по индексу в массиве C.

Сортировка подсчетом (counting sort)

Стоп! Почему это сработало?

Была ведь теорема, что нельзя быстрее, чем за O(N*logN)!

Потому, что мы ничего не сравнивали здесь, только обращались по индексу в массиве C.

Имеет смысл только на массивах из достаточно маленьких чисел.

Сортировка подсчетом (counting sort)

Стоп! Почему это сработало?

Была ведь теорема, что нельзя быстрее, чем за O(N*logN)!

Потому, что мы ничего не сравнивали здесь, только обращались по индексу в массиве C.

Имеет смысл только на массивах из достаточно маленьких чисел. Как обобщить?

Сортировка подсчетом (counting sort)

Пусть есть (сюръективное) отображение из множества элементов массива, в некое множество ключей:

$$\xi : Elements \to Keys$$

Сортировка подсчетом (counting sort)

Пусть есть (сюръективное) отображение из множества элементов массива, в некое множество ключей:

$\xi : Elements \to Keys$

При этом $|Elements| = N$, $Keys = 1, \ldots, K$

Сортировка подсчетом (counting sort)

Пусть есть (сюръективное) отображение из множества элементов массива, в некое множество ключей:

$$\xi: Elements \to Keys$$

При этом $|Elements| = N$, $Keys = 1, \ldots, K$

Модернизируем наш алгоритм, чтобы он сортировал элементы по соответствующим ключам

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

Тогда $\xi(element) = |element|$

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

Тогда $\xi(\,element\,) = |element|$

Заводим массив C[K]

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	0	0	0	0	0

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

Тогда $\xi(element) = |element|$

Заводим массив C[K]

C[K] =

0	0	0	0	0	0
C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

Проходим по исходному массиву и "считаем" элементы. Записываем результат в $C[\xi(element)]$

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

Тогда $\xi(element) = |element|$

Заводим массив C[K]

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	0	0	0	1	0

Проходим по исходному массиву и "считаем" элементы. Записываем результат в $C[\xi(element)]$

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

Тогда $\xi(element) = |element|$

Заводим массив C[K]

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	0	0	0	1	1

Проходим по исходному массиву и "считаем" элементы. Записываем результат в $C[\xi(element)]$

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =	-4	5	-1	2	-2	1	-1	0

Тогда $\xi(element) = |element|$

Заводим массив C[K]

C[K] =	1	3	2	0	1	1
	C[0]	C[1]	C[2]	C[3]	C[4]	C[5]

Проходим по исходному массиву и "считаем" элементы. Записываем результат в $C[\xi(element)]$

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	3	2	0	1	1

Преобразуем массив C, в цикле добавляя к каждому элементу значение предыдущего

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	2	0	1	1

Преобразуем массив C, в цикле добавляя к каждому элементу значение предыдущего

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	0	1	1

Преобразуем массив C, в цикле добавляя к каждому элементу значение предыдущего

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	1	1

Преобразуем массив C, в цикле добавляя к каждому элементу значение предыдущего

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	7	1

Преобразуем массив C, в цикле добавляя к каждому элементу значение предыдущего

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	7	8

Преобразуем массив C, в цикле добавляя к каждому элементу значение предыдущего

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	7	8

Преобразуем массив C, в цикле добавляя к каждому элементу значение предыдущего

Получили в i-ой ячейке C - количество ключей, меньших либо равных i-ому

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	7	8

Теперь идем обратным ходом по исходному массиву и строим результат!

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]

Теперь идем обратным ходом по исходному массиву и строим результат!

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
1	4	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0							

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	4	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0							

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	4	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0							

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	3	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0			-1				

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в
$$R\left[C[\xi(A[i])]-1\right]$$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	3	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0			-1				

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	3	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0			-1				

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	2	6	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0		1	-1				

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1
по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	2	5	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0		1	-1		-2		

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в
$$R\big[C[\xi(A[i])]-1\big]$$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	2	4	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0		1	-1	2	-2		

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	1	4	6	7	8

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0	-1	1	-1	2	-2		

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\big[C[\xi(A[i])]-1\big]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	1	4	6	7	7

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0	-1	1	-1	2	-2		5

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\big[C[\xi(A[i])]-1\big]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Хотим отсортировать массив из целых чисел -K, … , K-1 по модулю значений

A[N] =

-4	5	-1	2	-2	1	-1	0

C[K] =

C[0]	C[1]	C[2]	C[3]	C[4]	C[5]
0	1	4	6	6	7

R[N] =

R[0]	R[1]	R[2]	R[3]	R[4]	R[5]	R[6]	R[7]
0	-1	1	-1	2	-2	-4	5

Теперь идем обратным ходом по исходному массиву и строим результат!

1. Берем $C[\xi(A[i])]$
2. Ставим элемент в $R\left[C[\xi(A[i])]-1\right]$
3. Уменьшаем счетчик

Сортировка подсчетом (counting sort)

Получили:

1. Сложность?

Сортировка подсчетом (counting sort)

Получили:

1. Сложность - O(N + |Keys|)
   Если |Keys| ⩽ N, то O(N)

Сортировка подсчетом (counting sort)

Получили:

1. Сложность - O(N + |Keys|)
   Если |Keys| ⩽ N, то O(N)

2. Стабильную сортировку!

Сортировка подсчетом (counting sort)

Получили:

1. Сложность - O(N + |Keys|)
   Если |Keys| ⩽ N, то O(N)

2. Стабильную сортировку! (элементы с одинаковыми ключами остаются в том же порядке, что и были)

Сортировка подсчетом (counting sort)

Получили:

1. Сложность - O(N + |Keys|)
   Если |Keys| ⩽ N, то O(N)

2. Стабильную сортировку! (элементы с одинаковыми ключами остаются в том же порядке, что и были)

3. Работает на всем, что можно отобразить на множество ключей

Сортировка подсчетом (counting sort)

Получили:

1. Сложность - O(N + |Keys|)
   Если |Keys| ⩽ N, то O(N)

2. Стабильную сортировку! (элементы с одинаковыми ключами остаются в том же порядке, что и были)

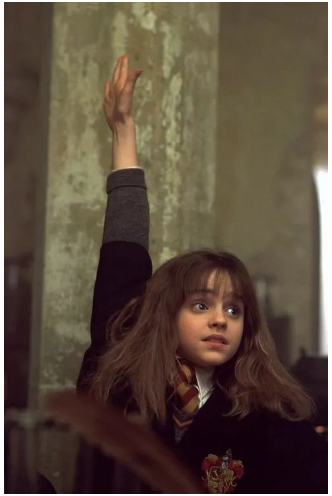
3. Работает на всем, что можно отобразить на множество ключей, но:

   a. Отображение должно работать быстро
   b. Множество ключей должно быть ограничено

А можно отсортировать хотя бы числа, но без ограничений?

А можно отсортировать хотя бы числа, но без ограничений?

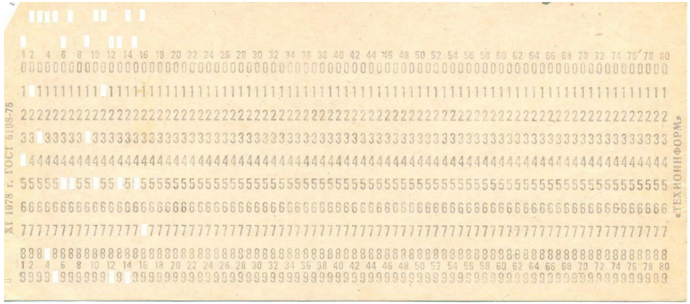
можно



Поразрядная сортировка (LSB radix sort)

Поразрядная сортировка (radix sort)

Когда-то использовалась для сортировки (физической!) перфокарт



Поразрядная сортировка (radix sort)

Идеи:

Поразрядная сортировка (radix sort)

Идеи:

- Любое целое число состоит из разрядов, цифр

Поразрядная сортировка (radix sort)

Идеи:

- Любое целое число состоит из разрядов, цифр
- Цифр мало - всего 10 штук (в десятичной системе)

Поразрядная сортировка (radix sort)

Идеи:

- Любое целое число состоит из разрядов, цифр
- Цифр мало - всего 10 штук (в десятичной системе)

- А что, если сортировать по одному из разрядов?

Поразрядная сортировка (radix sort)

Идеи:

- Любое целое число состоит из разрядов, цифр
- Цифр мало - всего 10 штук (в десятичной системе)

- А что, если сортировать по одному из разрядов?

  Для этого идеально подходит counting sort:
  ✓ ключи - это значения в разряде,
  ✓ множество ключей - ограничено

Поразрядная сортировка (radix sort)

Идеи:

- А что, если сортировать по одному из разрядов?

113	10	277	66	3	5	899	1

Поразрядная сортировка (radix sort)

Идеи:

- А что, если сортировать по одному из разрядов?

113	010	277	066	003	005	899	001

Поразрядная сортировка (radix sort)

Идеи:

- А что, если сортировать по одному из разрядов?

113	010	277	066	003	005	899	001

Поразрядная сортировка (radix sort)

Идеи:

- А что, если сортировать по одному из разрядов?

113	010	277	066	003	005	899	001

Counting sort

010	001	113	003	005	066	277	899

LSD radix sort - least significant digit

Поразрядная сортировка (radix sort)

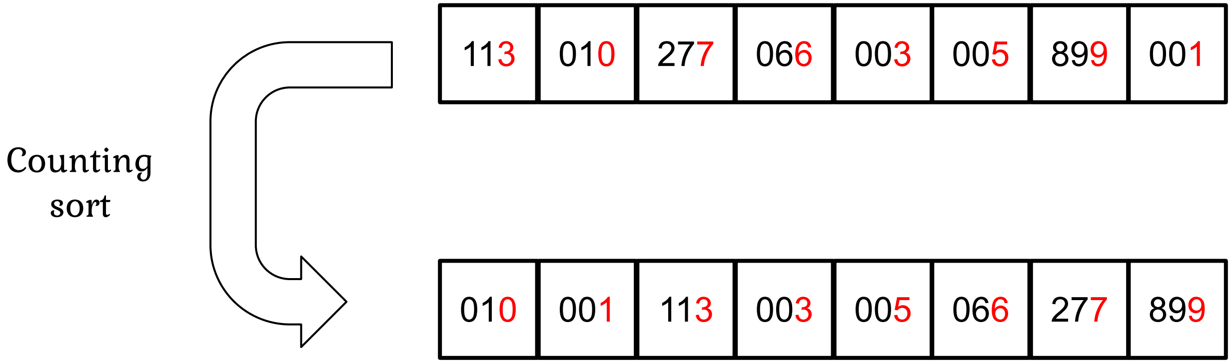
Идеи:

- Любое целое число состоит из разрядов, цифр
- Цифр мало - всего 10 штук (в десятичной системе)
- А что, если сортировать по одному из разрядов? (для этого идеально подходит counting sort)
- Продолжаем сортировать по остальным разрядам

Поразрядная сортировка (radix sort)

Идеи:

- Продолжаем сортировать по остальным разрядам



Здесь очень важно свойство стабильности сортировки по каждому из разрядов!

Поразрядная сортировка (radix sort)

Идеи:

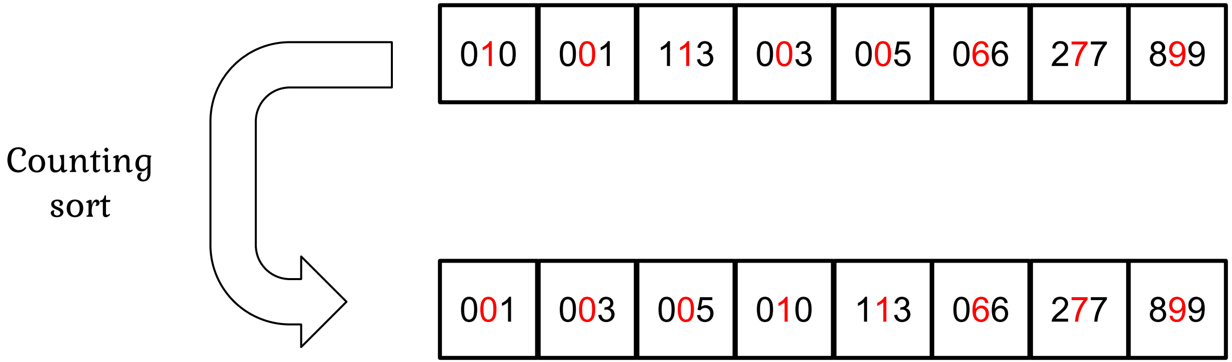
- Продолжаем сортировать по остальным разрядам

010	001	113	003	005	066	277	899

Поразрядная сортировка (radix sort)

Идеи:

- Продолжаем сортировать по остальным разрядам



Поразрядная сортировка (radix sort)

Идеи:

- Продолжаем сортировать по остальным разрядам

001	003	005	010	113	066	277	899

Counting sort

001	003	005	010	066	113	277	899

Поразрядная сортировка (radix sort)

Анализ:

- Сложность?

Поразрядная сортировка (radix sort)

Анализ:

- Сложность: O(N*V), где V - количество разрядов

Поразрядная сортировка (radix sort)

Анализ:

- Сложность: O(N*V), где V - количество разрядов
- На самом деле разряд ("цифра") - понятие растяжимое. Можно взять что угодно в качестве разряда.

Поразрядная сортировка (radix sort)

Анализ:

- Сложность: O(N*V), где V - количество разрядов
- На самом деле разряд ("цифра") - понятие растяжимое. Можно взять что угодно в качестве разряда.

  Например, будем считать, что 1 разряд в 32-битном числе (int из Java, например) - это 1 байт.

Поразрядная сортировка (radix sort)

Анализ:

- Сложность: O(N*V), где V - количество разрядов
- На самом деле разряд ("цифра") - понятие растяжимое. Можно взять что угодно в качестве разряда.

  Например, будем считать, что 1 разряд в 32-битном числе (int из Java, например) - это 1 байт.
- Всего 4 разряда для всех int-ов. Тогда сложность radix sort на всем множестве int-ов - O(N*4) = O(N)

Поразрядная сортировка (radix sort)

Вопрос: почему бы всегда не использовать radix sort, раз эта сортировка так хороша?

Поразрядная сортировка (radix sort)

Вопрос: почему бы всегда не использовать radix sort?

Ответ. Radix sort:

- Работает только с разрядами => не универсальна

Поразрядная сортировка (radix sort)

Вопрос: почему бы всегда не использовать radix sort?

Ответ. Radix sort:

- Работает только с разрядами => не универсальна
- Для получения линейной сложности нужно, чтобы элементы состояли из k разрядов (обычно переходят к битам), при этом $k \leqslant \log N$

Поразрядная сортировка (radix sort)

Вопрос: почему бы всегда не использовать radix sort?

Ответ. Radix sort:

- Работает только с разрядами => не универсальна
- Для получения линейной сложности нужно, чтобы элементы состояли из k разрядов (обычно переходят к битам), при этом $k \leqslant \log N$
- Требует дополнительной памяти

Поразрядная сортировка (radix sort)

Вопрос: почему бы всегда не использовать radix sort?

Ответ. Radix sort:

- Работает только с разрядами => не универсальна
- Для получения линейной сложности нужно, чтобы элементы состояли из k разрядов (обычно переходят к битам), при этом $k \leqslant \log N$
- Требует дополнительной памяти
- Константы не очень хороши, из-за этого есть более практичные сортировки

Поразрядная сортировка (radix sort)

Вопрос: почему бы всегда не использовать radix sort?

Ответ. Radix sort:

- Работает только с разрядами => не универсальна
- Для получения линейной сложности нужно, чтобы элементы состояли из k разрядов (обычно переходят к битам), при этом $k \leqslant \log N$
- Требует дополнительной памяти
- Константы не очень хороши, из-за этого есть более практичные сортировки (если мы говорим про числа)

Поразрядная сортировка (radix sort)

Вопрос: а когда radix sort действительно хороша?

Поразрядная сортировка (radix sort)

Вопрос: а когда radix sort действительно хороша?

1. Когда значения в любом случае долго сравнивать

Поразрядная сортировка (radix sort)

Вопрос: а когда radix sort действительно хороша?

1. Когда значения в любом случае долго сравнивать (например, в случае строк и их сортировки в лексикографическом порядке)

Поразрядная сортировка (radix sort)

Вопрос: а когда radix sort действительно хороша?

1. Когда значения в любом случае долго сравнивать (например, в случае строк и их сортировки в лексикографическом порядке)

2. Когда сортировка идет не с младшего, а со старшего разряда (MSD radix sort), сортировка хорошо распараллеливается

Поразрядная сортировка (radix sort)

Вопрос: а когда radix sort действительно хороша?

1. Когда значения в любом случае долго сравнивать (например, в случае строк и их сортировки в лексикографическом порядке)

2. Когда сортировка идет не с младшего, а со старшего разряда (MSD radix sort), сортировка хорошо распараллеливается

3. Используется на практике в алгоритмах построения суффиксных массивов строк (Kärkkäinen, Sanders, Burkhardt)

Мини-задача #10 (1 балл)

Реализовать алгоритм поразрядной сортировки (LSD radix sort) для набора строк одинаковой длины.

Подготовить набор данных для тестирования, на нем проверять корректность алгоритма, сравнивая с результатом работы любой сортировки основанной на сравнении.