# Алгоритмы и структуры данных

# Асимптотический анализ. Разделяй-и-властвуй DaC




Практическое занятие **04**

**2026–2027** учебный год

# План

1. Разделяй-и-властвуй над пространством решений: бинарный поиск по ответу
2. Задача CLOSEST PAIR: поиск ближайшей пары точек на прямой (1D) и плоскости (2D)
3. Задача K-WAY MERGE: слияние набора упорядоченных связных списков [массивов] [для самостоятельного разбора]

# Warm-Up

# FAQ Разделяй-и-властвуй, асимптотика

1. Всегда ли функция $T(n)$ временной сложности является монотонно возрастающей?
2. Какая функция растет быстрее — $2^n$ или $n!$
3. На сколько подзадач нужно разделять задачи при использовании парадигмы «разделяй-и-властвуй»?
4. Верно ли, что применении парадигмы «разделяй-и-властвуй» всегда дает улучшение временной сложности [по сравнению с другими подходами]?

# Разделяй-и-властвуй над пространством решений

## бинарный поиск по ответу

Задача 1

# Бинарный поиск по ответу

время покраски $N$ досок

| $a$ | 2 | 4 | 1 | 3 | 2 |
|---|---|---|---|---|---|

задача оптимальной по времени покраски забора

Вход: $N$ – количество досок, покраска каждой из которых занимает $a[i]$ времени, и $K$ – число маляров, работающих параллельно

Выход: $T^*$ – **минимальное** время, за которое $K$ маляров справятся с покраской забора

маляр красит **непрерывный фрагмент** забора, т. е., например, доски 1-3, а не 1, 2 и 4

Задача 1

# Бинарный поиск по ответу

время покраски $N$ досок

| $a$ | 2 | 4 | 1 | 3 | 2 |
|---|---|---|---|---|---|

[Q] В каких пределах находится общее время покраски забора $T$?

- минимум — **4** [самая длинная доска]
- максимум — **12** [сумма по всем доскам]

задача оптимальной по времени покраски забора

Вход: $N$ – количество досок, покраска каждой из которых занимает $a[i]$ времени, и $K$ – число маляров, работающих параллельно

Выход: $T^*$ – **минимальное** время, за которое $K$ маляров справятся с покраской забора

маляр красит **непрерывный фрагмент** забора, т. е., например, доски 1-3, а не 1, 2 и 4

Задача 1

# Бинарный поиск по ответу

время покраски $N$ досок

| $a$ | 2 | 4 | 1 | 3 | 2 |
|---|---|---|---|---|---|

пространство возможных решений

| $T$ | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
|---|---|---|---|---|---|---|---|---|---|

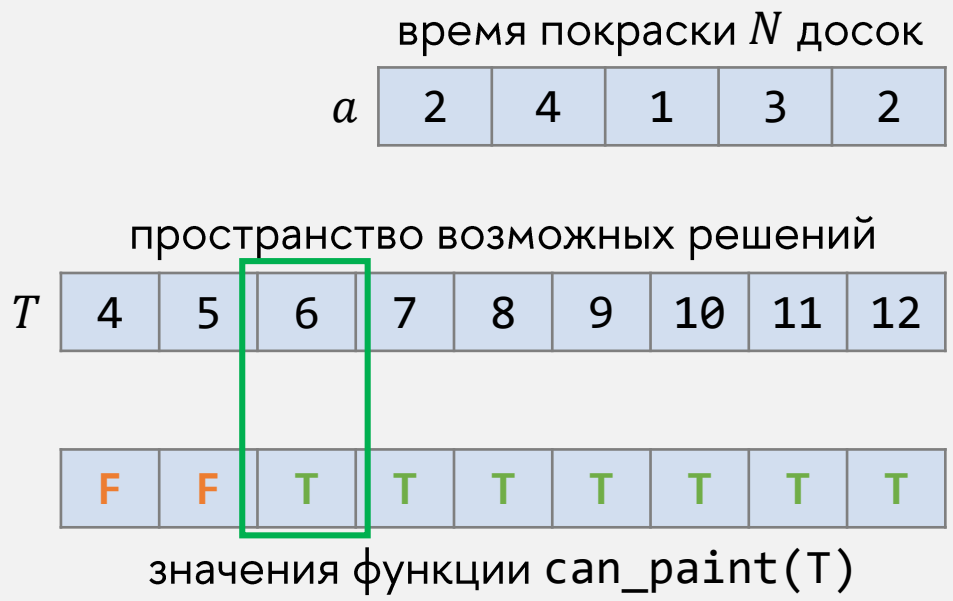
$N = 5$ досок и $K = 2$ маляра

необходимо определить, можем ли мы покрасть забор за время $T$

- функция `can_paint(T)` возвращает `true`, если хватает маляров и `false`, если не хватает

Задача 1

# Бинарный поиск по ответу




$N = 5$ досок и $K = 2$ маляра

необходимо определить, можем ли мы покрасть забор за время $T$

- функция `can_paint(T)` возвращает **`true`**, если хватает маляров и **`false`**, если не хватает
- как только среди значений появляется **`true`**, все последующие значения также **`true`**

$\Rightarrow$ найти первое значение $T$, при котором `can_paint(T) = true`

Задача 1

# Бинарный поиск по ответу




$N = 5$ досок и $K = 2$ маляра

**ИТЕРАЦИЯ 1**

$\text{mid} = 4 + (12 - 4) / 2 = 8$

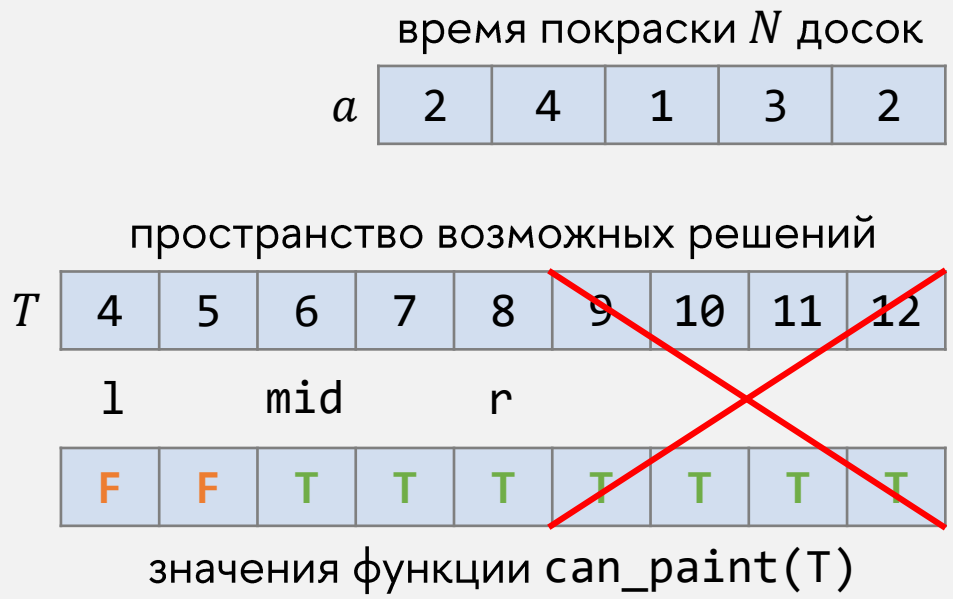
проверяем `can_paint(8)`:
жадно даем малярам доски, пока сумма времени не превысит 8

- маляр 1: $a[0] + a[1] + a[2] = 7$
- маляр 2: $a[3] + a[4] = 5$

`can_paint(8) = true` ⇒ `r = mid`

Задача 1

# Бинарный поиск по ответу




$N = 5$ досок и $K = 2$ маляра

**ИТЕРАЦИЯ 2**

$\text{mid} = 4 + (8 - 4) / 2 = 6$

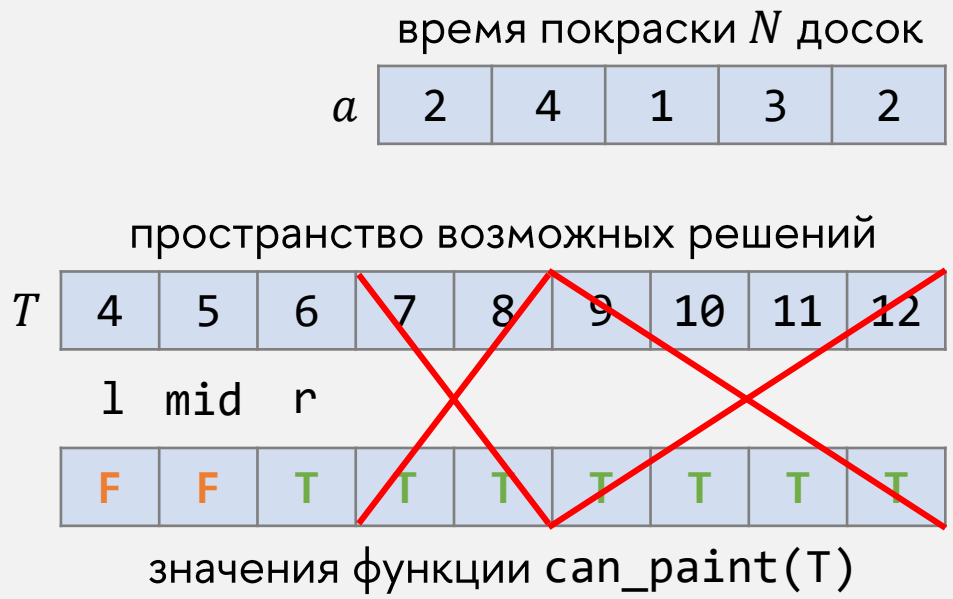
проверяем `can_paint(6)`:
жадно даем малярам доски, пока сумма времени не превысит 6

- маляр 1: $a[0] + a[1] = 6$
- маляр 2: $a[2] + a[3] + a[4] = 6$

`can_paint(6) = true` ⇒ `r = mid`

Задача 1

# Бинарный поиск по ответу




$N = 5$ досок и $K = 2$ маляра

**ИТЕРАЦИЯ 2**

$\text{mid} = 4 + (6 - 4) / 2 = 5$

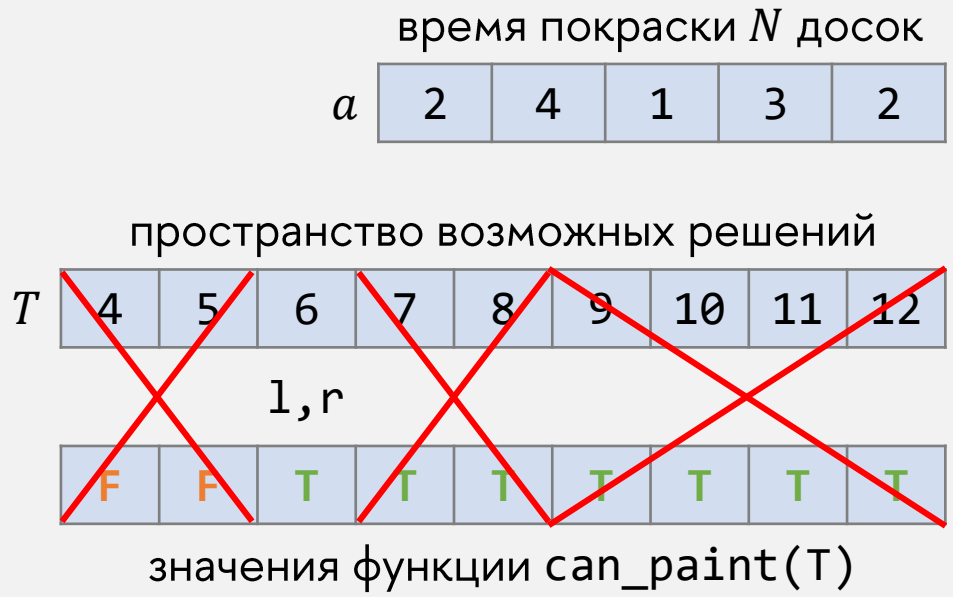
проверяем `can_paint(5)`:
жадно даем малярам доски, пока сумма времени не превысит 5

- маляр 1: $a[0] = 2$
- маляр 2: $a[1] + a[2] = 5$

`can_paint(6)` = `false` ⇒ `l=mid+1`

Задача 1

# Бинарный поиск по ответу




$N = 5$ досок и $K = 2$ маляра

## ИТЕРАЦИЯ 3

`l == r` ⇒ цикл завершен

- мы нашли первое значение $T$ в диапазоне $[4, 12]$, для которого `can_paint(T)` возвращает `true`
- минимальное время покраски забора $T^* = 6$

Задача 1





# Бинарный поиск по ответу

применение парадигмы «разделяй-и-властвуй», но не к входным данным, а к **пространству возможных ответов**

- сложность проверки `[can_paint(T)]` — $O(N)$
- сложность бинпоиска — $O(\log size)$, где $size$ – это размер пространства возможных ответов
- суммарная сложность — $O(N \cdot \log size)$

Задача 1





# Бинарный поиск по ответу

общие шаблоны применимости такого подхода

- можно ли сформулировать задачу как «найти минимальное / максимальное значение $X$, при котором выполняется условие $P$»?
- является ли условие P *монотонным*?
  если $P(X)$ верно, то является ли $P(X+1)$ верным?
- можно ли эффективно [за полином] проверить,
  подходит ли конкретное значение $X$?

# Поиск ближайшей пары точек CLOSEST PAIR

## наивный подход

Задача 2




# CLOSEST PAIR dD

Вход: $P$ – конечное множество $n$ точек, заданных координатами в $d$-мерном пространстве.

Выход: пара точек $p_1$ и $p_2$ из множества $P$, расстояние между которыми минимально.

Задача 2



# CLOSEST PAIR dD NAÏVE

Как найти ближайшую пару точек в $d$-мерном пространстве?

Задача 2





# CLOSEST PAIR dD NAÏVE

1. Рассчитать попарные расстояния между заданными точками по формуле Евклида.
2. Найти минимум среди всех вычисленных расстояний.

[Q] Какова сложность такого алгоритма?

Задача 2




# CLOSEST PAIR dD NAÏVE

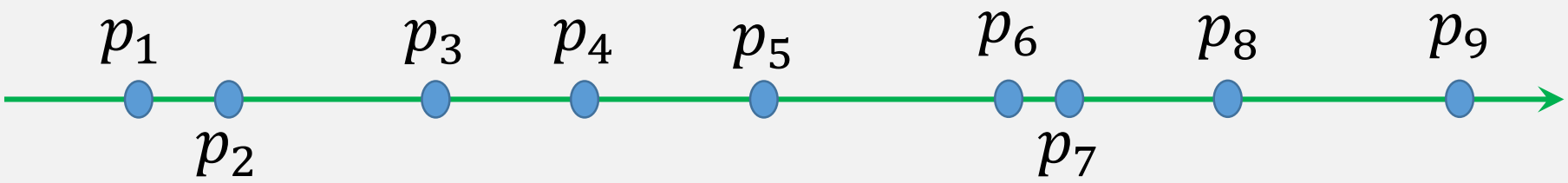
1. Рассчитать попарные расстояния между заданными точками по формуле Евклида.
2. Найти минимум среди всех вычисленных расстояний.

[Q] Какова сложность такого алгоритма? $O(dn^2)$

в некоторых случаях будем явно указывать значимые константы внутри асимптотических границ

Задача 2

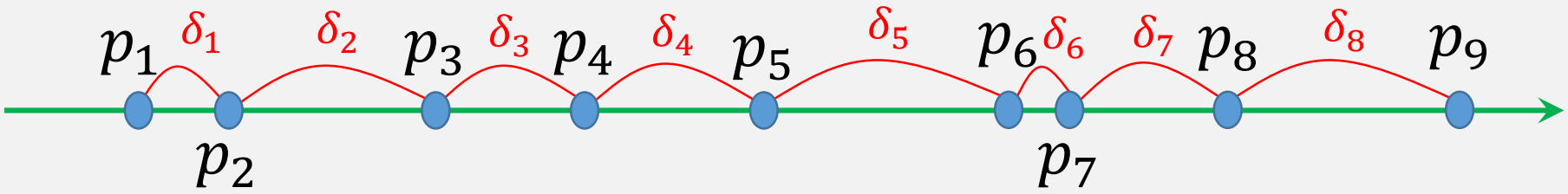
# CLOSEST PAIR 1D NAÏVE




Как найти ближайшую пару точек в одномерном пространстве?

Задача 2

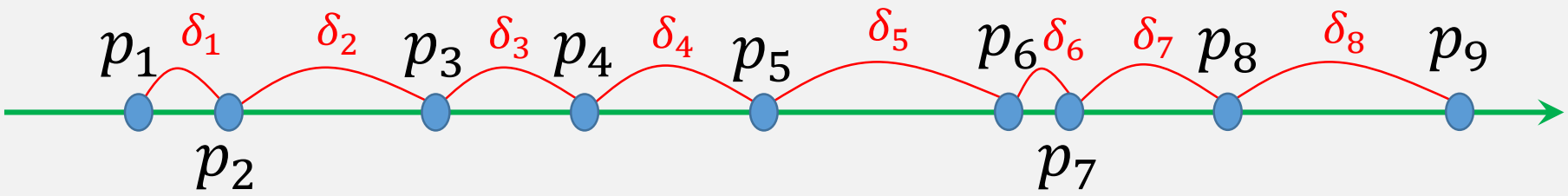
# CLOSEST PAIR 1D NAÏVE




1. Отсортировать точки по координате
2. Вычислить расстояния между соседними точками
3. Найти минимальное расстояние

Задача 2

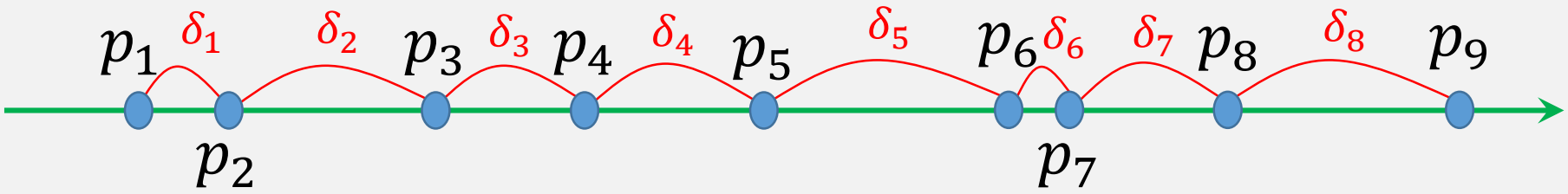
# CLOSEST PAIR 1D NAÏVE




1. $O(?)$ Отсортировать точки по координате
2. $O(?)$ Вычислить расстояния между соседними точками
3. $O(?)$ Найти минимальное расстояние

Задача 2

# CLOSEST PAIR 1D NAÏVE




1. $\mathbf{O(n \log n)}$ Отсортировать точки по координате
   например, с помощью MERGE SORT
2. $\mathbf{O(n)}$ Вычислить расстояния между соседними точками
   линейное вычисление для $n - 1$ пары точек
3. – Найти минимальное расстояние
   минимум можно найти уже на предыдущем шаге

# Поиск ближайшей пары точек CLOSEST PAIR 1D

DaC–подход

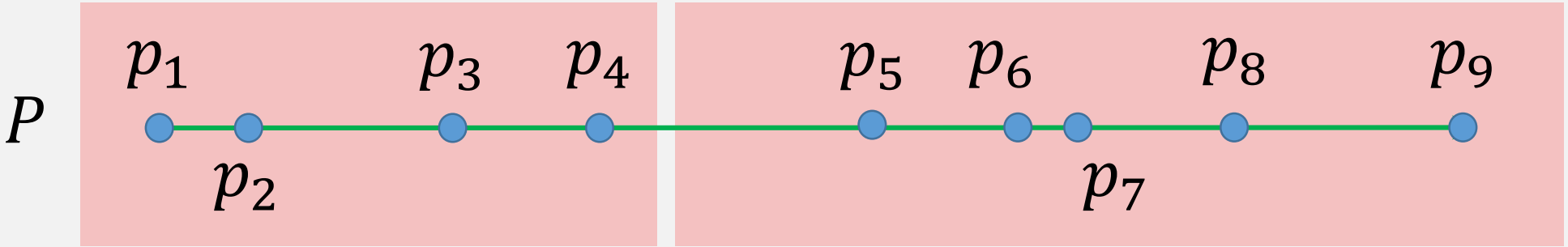
Задача 2

# CLOSEST PAIR 1D DaC



Задача 2

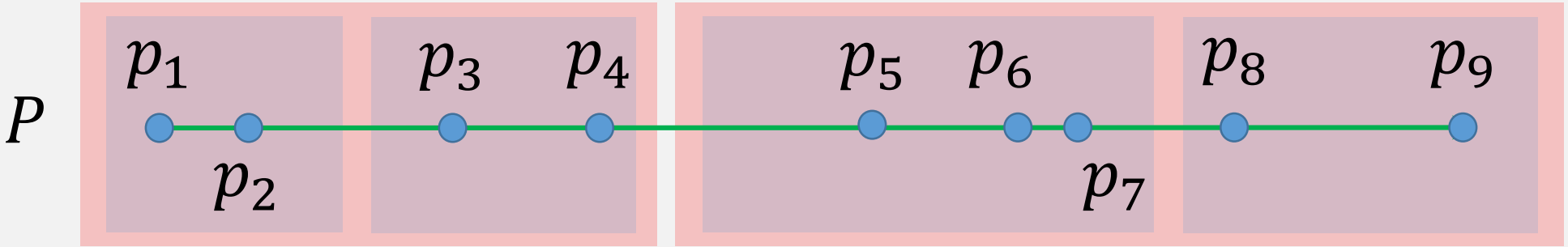
# CLOSEST PAIR 1D DaC




1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

Задача 2

# CLOSEST PAIR 1D DaC

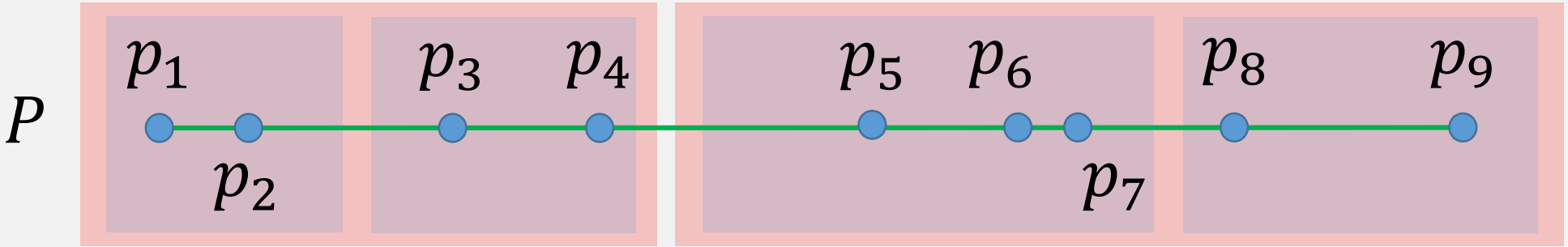



1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

   [Q] Каким должно быть условие остановки рекурсии?

Задача 2

# CLOSEST PAIR 1D DaC

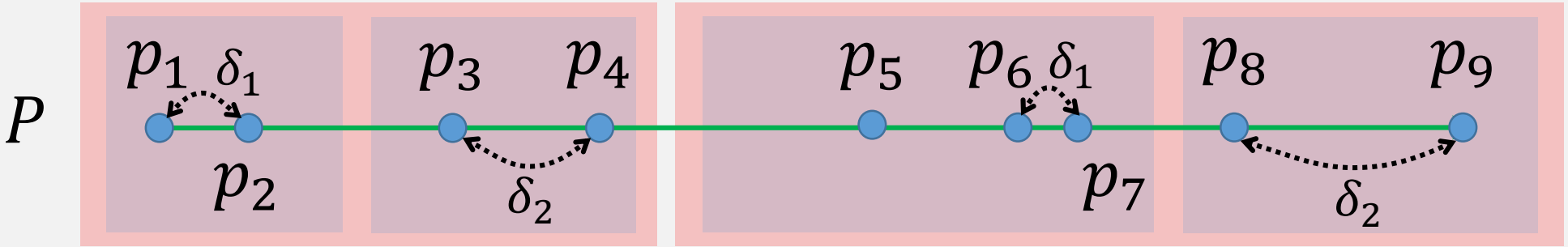



1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

   [Q] Каким должно быть условие остановки рекурсии? $|P_i| \leq 3$

Задача 2

# CLOSEST PAIR 1D DaC



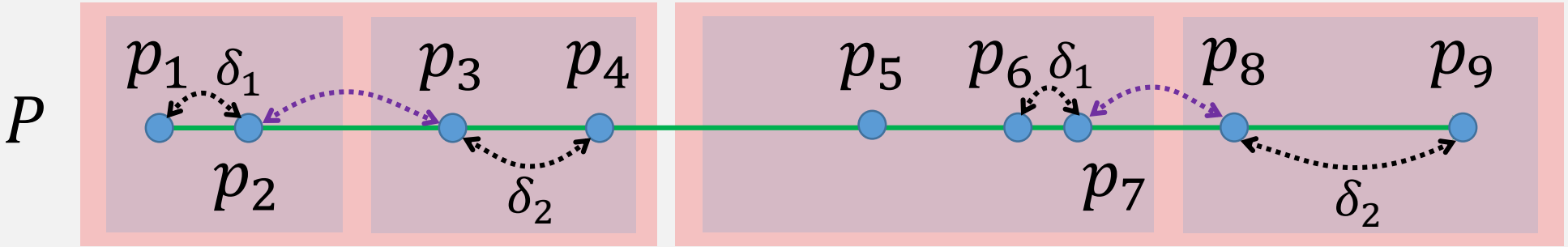

1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

   [Q] Каким должно быть условие остановки рекурсии? $|P_i| \leq 3$

2. **CONQUER** Найти ближайшую пару в каждой из половин $\delta_1$ — мин. расстояние в $P_1$ и $\delta_2$ — мин. расстояние в $P_2$

Задача 2

# CLOSEST PAIR 1D DaC




1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

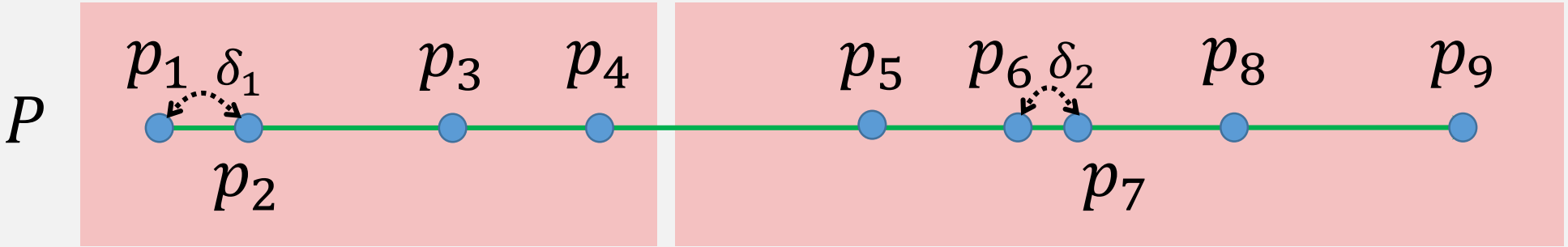
   [Q] Каким должно быть условие остановки рекурсии? $|P_i| \leq 3$

2. **CONQUER** Найти ближайшую пару в каждой из половин $\delta_1$ — мин. расстояние в $P_1$ и $\delta_2$ — мин. расстояние в $P_2$

3. **COMBINE** Найти и вернуть $\delta = \min(\delta_1, \delta_2, |p_m - p_{m+1}|)$

Задача 2

# CLOSEST PAIR 1D DaC




1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

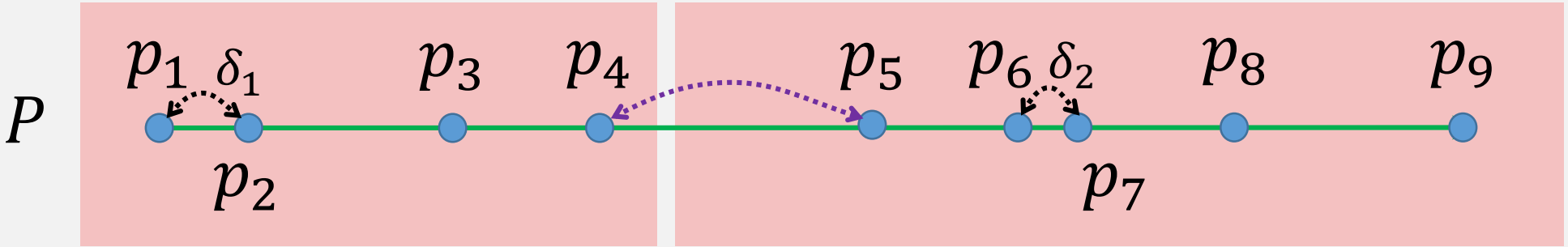
   [Q] Каким должно быть условие остановки рекурсии? $|P_i| \leq 3$

2. **CONQUER** Найти ближайшую пару в каждой из половин $\delta_1$ — мин. расстояние в $P_1$ и $\delta_2$ — мин. расстояние в $P_2$

3. **COMBINE** Найти и вернуть $\delta = \min(\delta_1, \delta_2, |p_m - p_{m+1}|)$

Задача 2

# CLOSEST PAIR 1D DaC




1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

   [Q] Каким должно быть условие остановки рекурсии? $|P_i| \leq 3$

2. **CONQUER** Найти ближайшую пару в каждой из половин $\delta_1$ — мин. расстояние в $P_1$ и $\delta_2$ — мин. расстояние в $P_2$

3. **COMBINE** Найти и вернуть $\delta = \min(\delta_1, \delta_2, |p_m - p_{m+1}|)$

Задача 2

# CLOSEST PAIR 1D DaC




1. **DIVIDE** Выполнить рекурсивное разбиение упорядоченного множество заданных точек на две части по $m = n/2$ $P_1 = \{p_1, \dots, p_m\}$ и $P_2 = \{p_{m+1}, \dots, p_n\}$

   [Q] Каким должно быть условие остановки рекурсии? $|P_i| \leq 3$

2. **CONQUER** Найти ближайшую пару в каждой из половин $\delta_1$ — мин. расстояние в $P_1$ и $\delta_2$ — мин. расстояние в $P_2$

3. **COMBINE** Найти и вернуть $\delta = \min(\delta_1, \delta_2, |p_m - p_{m+1}|)$

Задача 2

# CLOSEST PAIR 1D DaC

| | **closest_pair_1D**($P_{\text{sorted}} = \{p_1, p_2, \dots, p_n\}$) | $T(n)$ |
|---|---|---|
| 1 | **if** $n = 1$ **return** $\infty$ | $O(1)$ |
| 2 | **if** $n = 2$ **return** $\lvert p_1 - p_2 \rvert$ | |
| 3 | **if** $n = 3$ **return** . . . | |
| 4 | $m = n/2$ – индекс медианы массива | |
| 5 | $\delta_1$ = **closest_pair_1D**$(p_1, p_2, \dots, p_m)$ | $2T(n/2)$ |
| 6 | $\delta_2$ = **closest_pair_1D**$(p_{m+1}, p_{m+2}, \dots, p_n)$ | |
| 8 | **return** **min**$(\delta_1, \delta_2, \lvert p_m - p_{m+1} \rvert)$ | $O(1)$ |

Задача 2

# CLOSEST PAIR 1D DaC

| | **closest_pair_1D**($P_{\text{sorted}} = \{p_1, p_2, \dots, p_n\}$) | $T(n)$ |
|---|---|---|
| 1 | **if** $n = 1$ **return** $\infty$ | $O(1)$ |
| 2 | **if** $n = 2$ **return** $\lvert p_1 - p_2 \rvert$ | |
| 3 | **if** $n = 3$ **return** . . . | |
| 4 | $m = n/2$ – индекс медианы массива | |
| 5 | $\delta_1$ = **closest_pair_1D**$(p_1, p_2, \dots, p_m)$ | $2T(n/2)$ |
| 6 | $\delta_2$ = **closest_pair_1D**$(p_{m+1}, p_{m+2}, \dots, p_n)$ | |
| 8 | **return** **min**$(\delta_1, \delta_2, \lvert p_m - p_{m+1} \rvert)$ | $O(1)$ |

1. Рекуррентное соотношение $T(n) = 2T(n/2) + O(1) \Rightarrow T(n) = O(n)$

Задача 2

# CLOSEST PAIR 1D DaC

| | **closest_pair_1D**($P_{\text{sorted}} = \{p_1, p_2, \dots, p_n\}$) | $T(n)$ |
|---|---|---|
| 1 | **if** $n = 1$ **return** $\infty$ | $O(1)$ |
| 2 | **if** $n = 2$ **return** $\lvert p_1 - p_2 \rvert$ | |
| 3 | **if** $n = 3$ **return** . . . | |
| 4 | $m = n/2$ – индекс медианы массива | |
| 5 | $\delta_1$ = **closest_pair_1D**$(p_1, p_2, \dots, p_m)$ | $2T(n/2)$ |
| 6 | $\delta_2$ = **closest_pair_1D**$(p_{m+1}, p_{m+2}, \dots, p_n)$ | |
| 8 | **return** **min**$(\delta_1, \delta_2, \lvert p_m - p_{m+1} \rvert)$ | $O(1)$ |

1. Рекуррентное соотношение $T(n) = 2T(n/2) + O(1) \Rightarrow T(n) = O(n)$
2. Необходимо учесть сложность сортировки массива точек $O(n \log n)$
3. Общая сложность $T_{\text{total}} = O(n) + O(n \log n) = \mathbf{O(n \log n)}$

Задача 2

# CLOSEST PAIR 1D DaC

| | **closest_pair_1D**($P_{\text{sorted}} = \{p_1, p_2, \dots, p_n\}$) | $T(n)$ |
|---|---|---|
| 1 | **if** $n = 1$ **return** $\infty$ | $O(1)$ |
| 2 | **if** $n = 2$ **return** $\lvert p_1 - p_2 \rvert$ | |
| 3 | **if** $n = 3$ **return** . . . | |
| 4 | $m = n/2$ – индекс медианы массива | |
| 5 | $\delta_1$ = **closest_pair_1D**$(p_1, p_2, \dots, p_m)$ | $2T(n/2)$ |
| 6 | $\delta_2$ = **closest_pair_1D**$(p_{m+1}, p_{m+2}, \dots, p_n)$ | |
| 8 | **return** **min**$(\delta_1, \delta_2, \lvert p_m - p_{m+1} \rvert)$ | $O(1)$ |

Теоретического и практического выигрыша во временной сложности по сравнению с CLOSEST PAIR 1D NAÏVE мы не получаем… Более того, мы только добавим накладные расходы, связанные с рекурсией ☹

# Поиск ближайшей пары точек CLOSEST PAIR 2D

DaC–подход

Задача 2

# CLOSEST PAIR 2D DaC

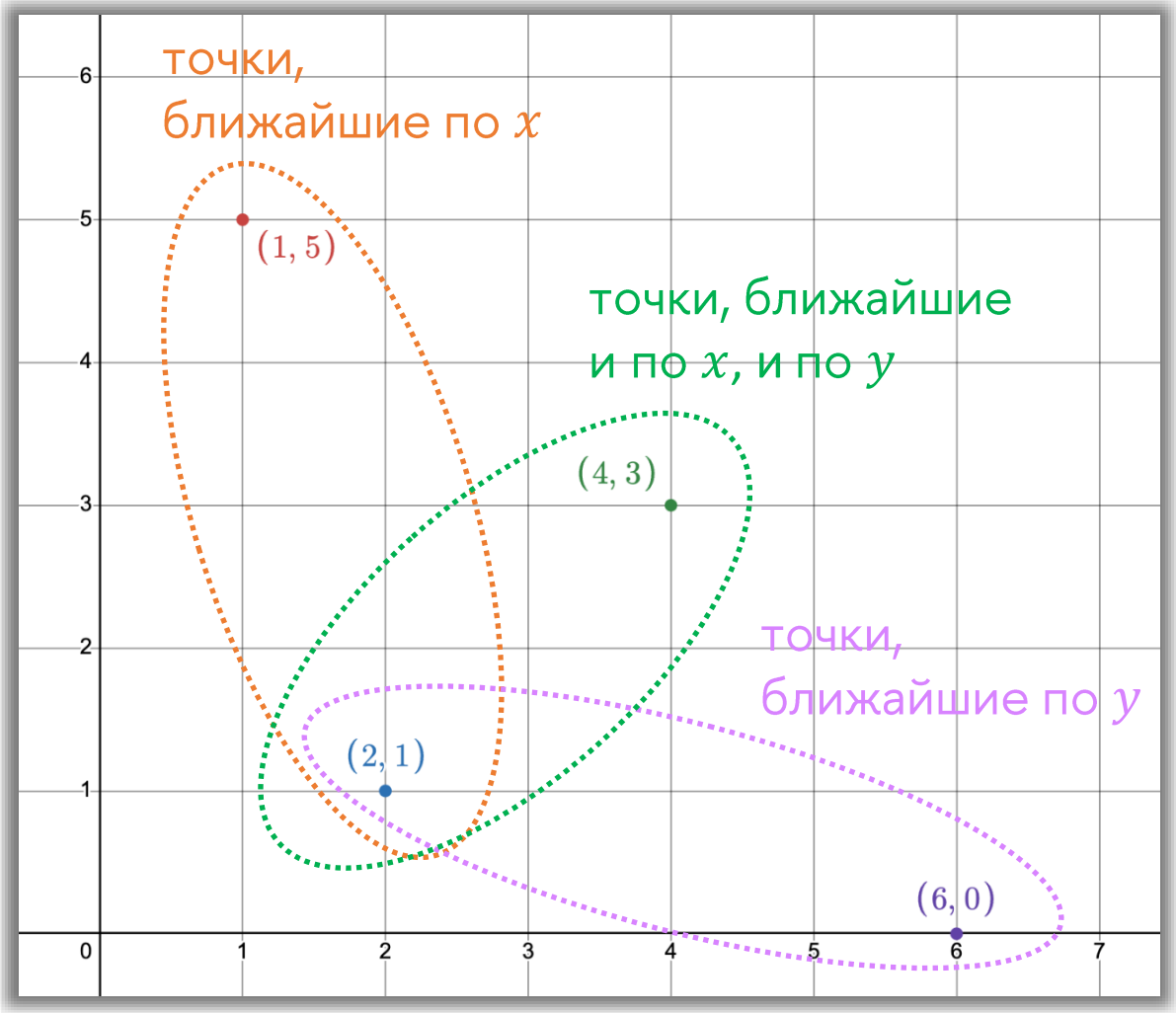
1. Расстояния вычисляем по формуле Евклида

$$\mathrm{dist}(a,b) = \sqrt{(b_x - a_x)^2 + \left(b_y - a_y\right)^2}$$

2. Сложность алгоритма должна совпадать с одномерным случаем $T(n) = O(n \log n)$

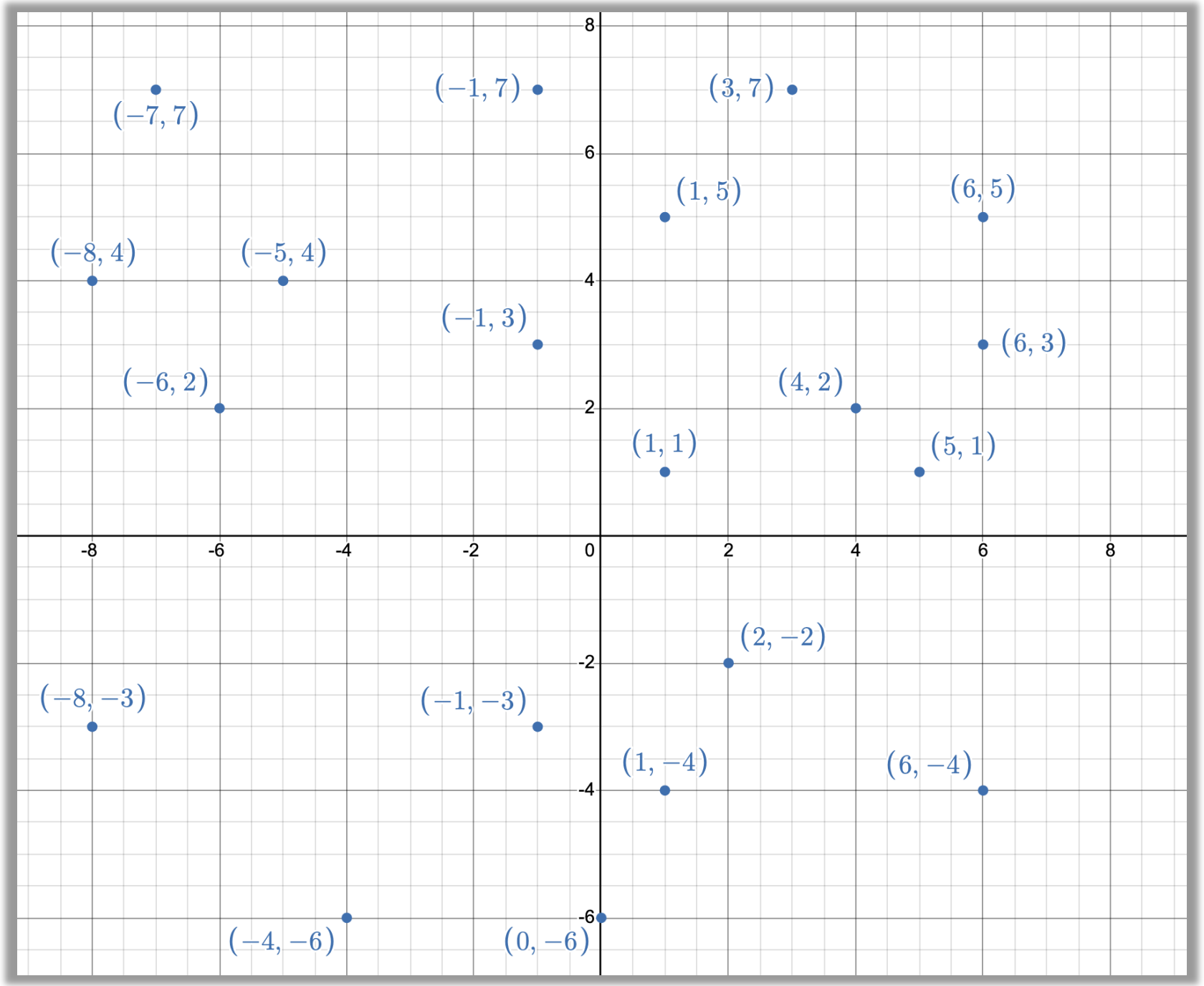
Задача 2

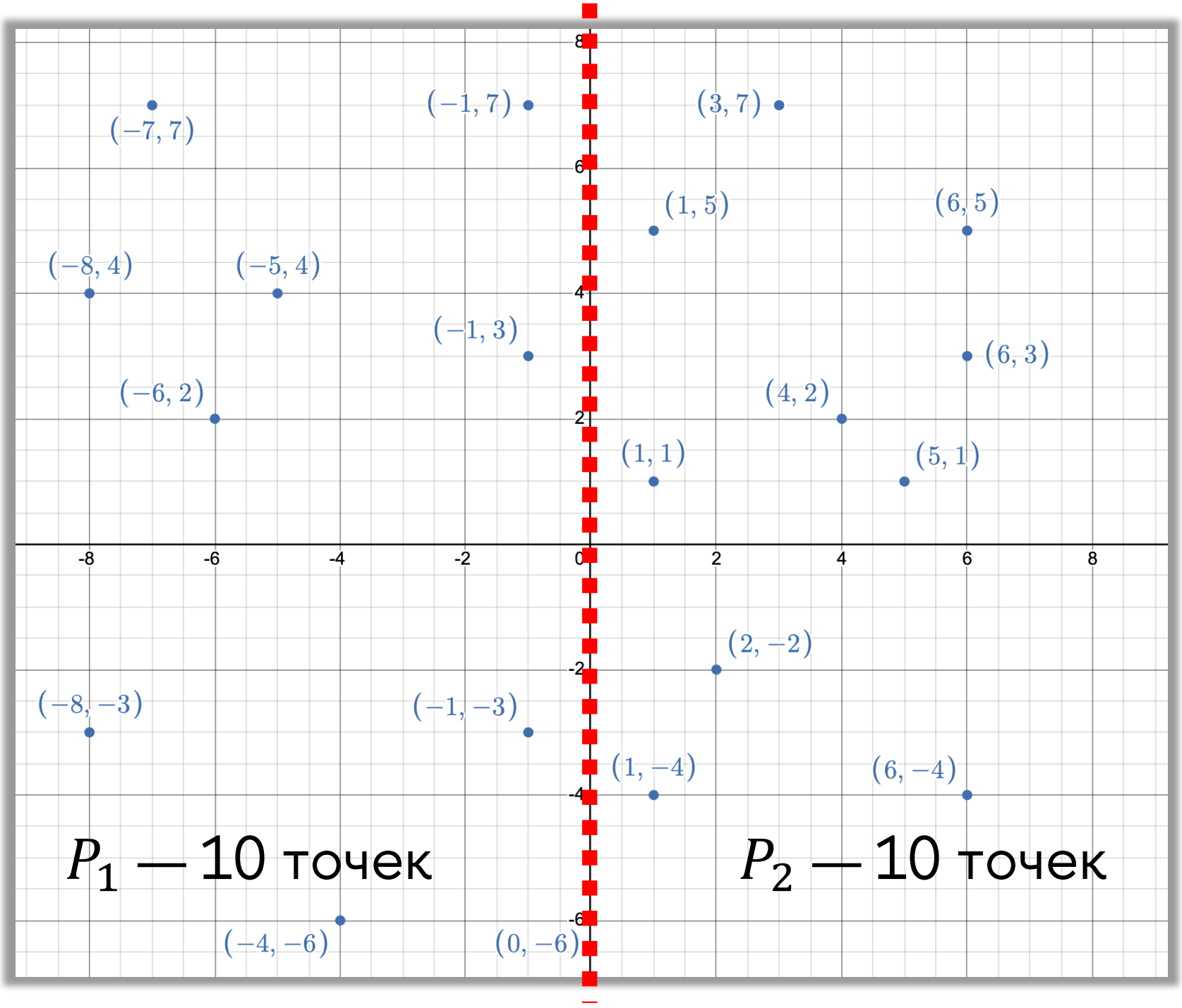
# CLOSEST PAIR 2D DaC




Ближайшие точки на плоскости не обязательно являются точками, ближайшими только по одной координате!

Сортировка точек и линейный поиск [как в случае с 1D] нам не подходят... ПОЧЕМУ?


(−7, 7)
(−1, 7)
(3, 7)
(1, 5)
(6, 5)
(−8, 4)
(−5, 4)
(−1, 3)
(6, 3)
(−6, 2)
(4, 2)
(1, 1)
(5, 1)
(2, −2)
(−8, −3)
(−1, −3)
(1, −4)
(6, −4)
(−4, −6)
(0, −6)
8
6
4
2
0
-2
-4
-6
-8
-6
-4
-2
2
4
6
8

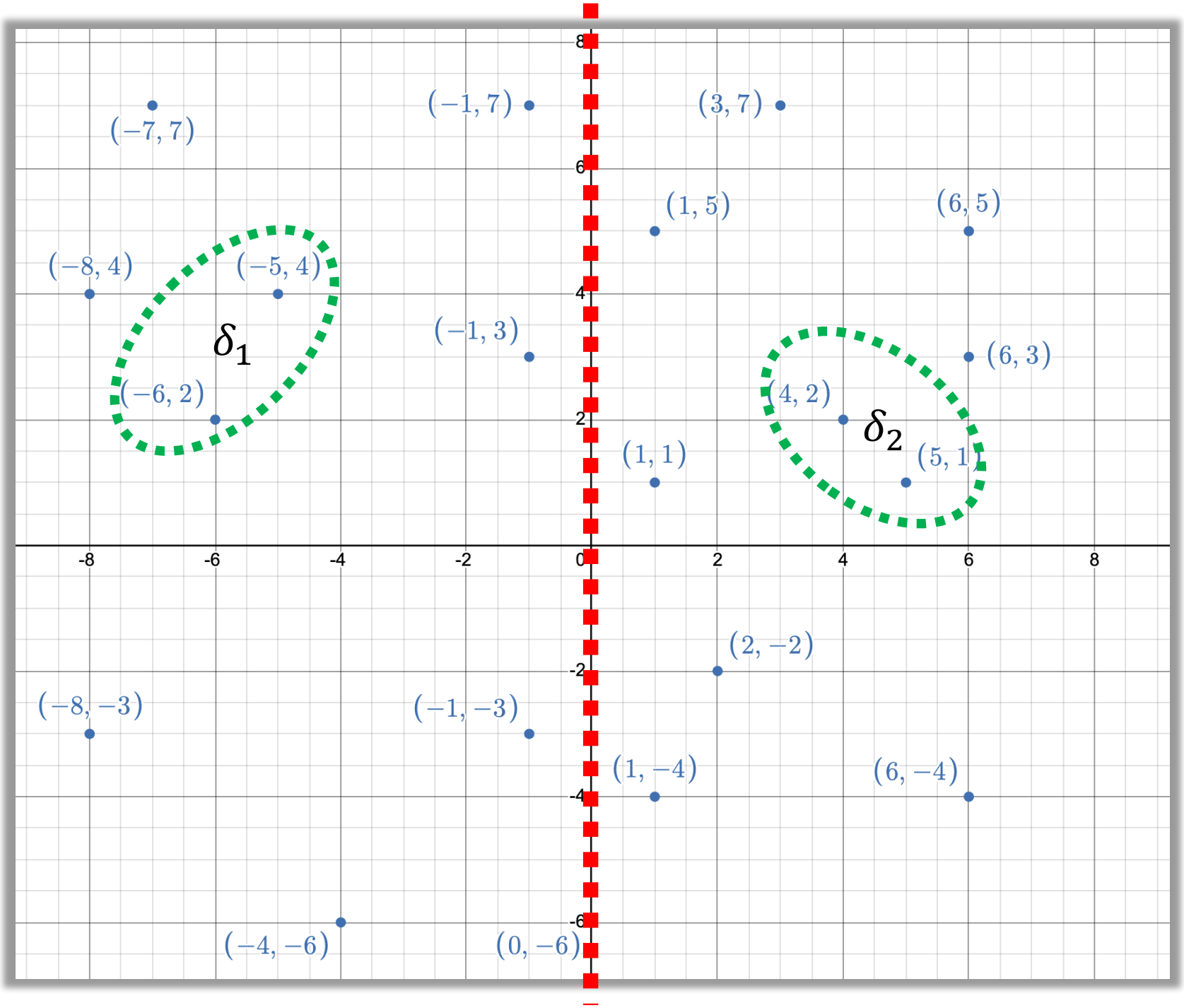



## DIVIDE

Рекурсивно разбиваем точки, упорядоченные по $x$, на две половины $P_1$ и $P_2$

в нашем случае $x_{\text{mid}} = 0$

рассматриваем один рекурсивный шаг

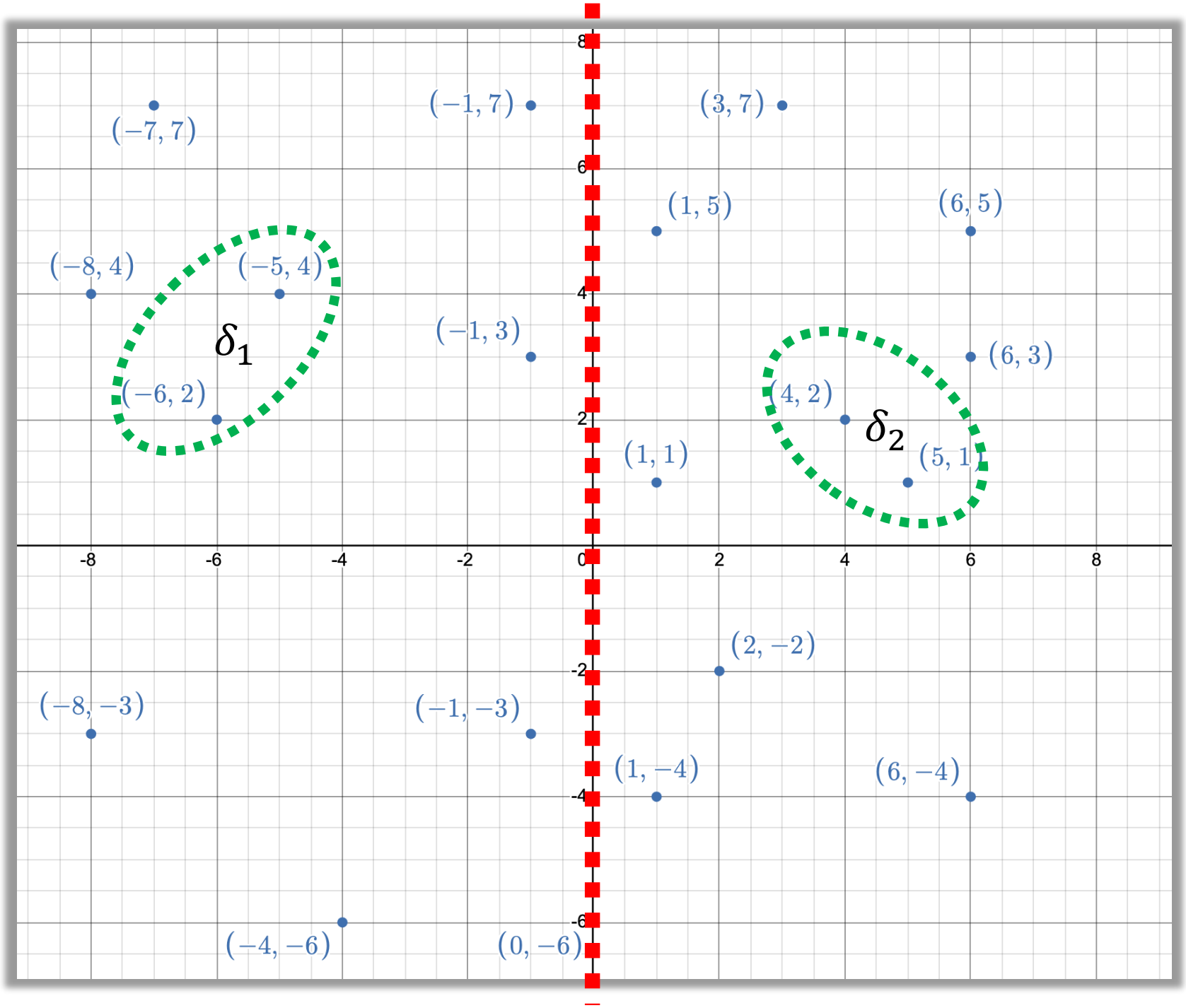



## CONQUER

Найти ближайшую пару точек в каждой из половин

$\delta_1$ — мин. расстояние в $P_1$

$\delta_2$ — мин. расстояние в $P_2$

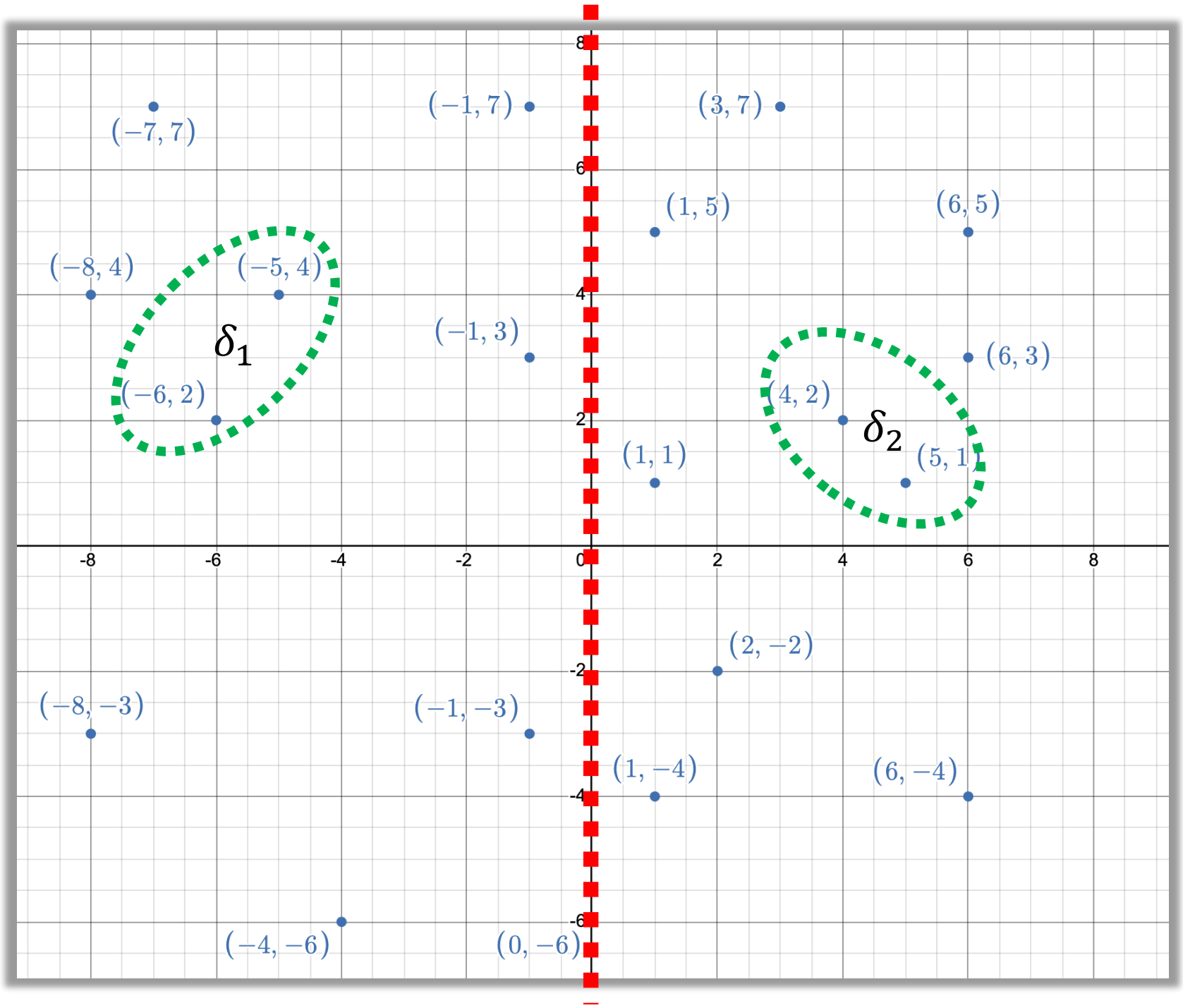



мы внутри шага рекурсии

## COMBINE

Найти ближайшую пару по разные стороны от $x_{\text{mid}} = 0$

[Q] Может сопоставить все точки слева со всеми справа от $x_{\text{mid}} = 0$?



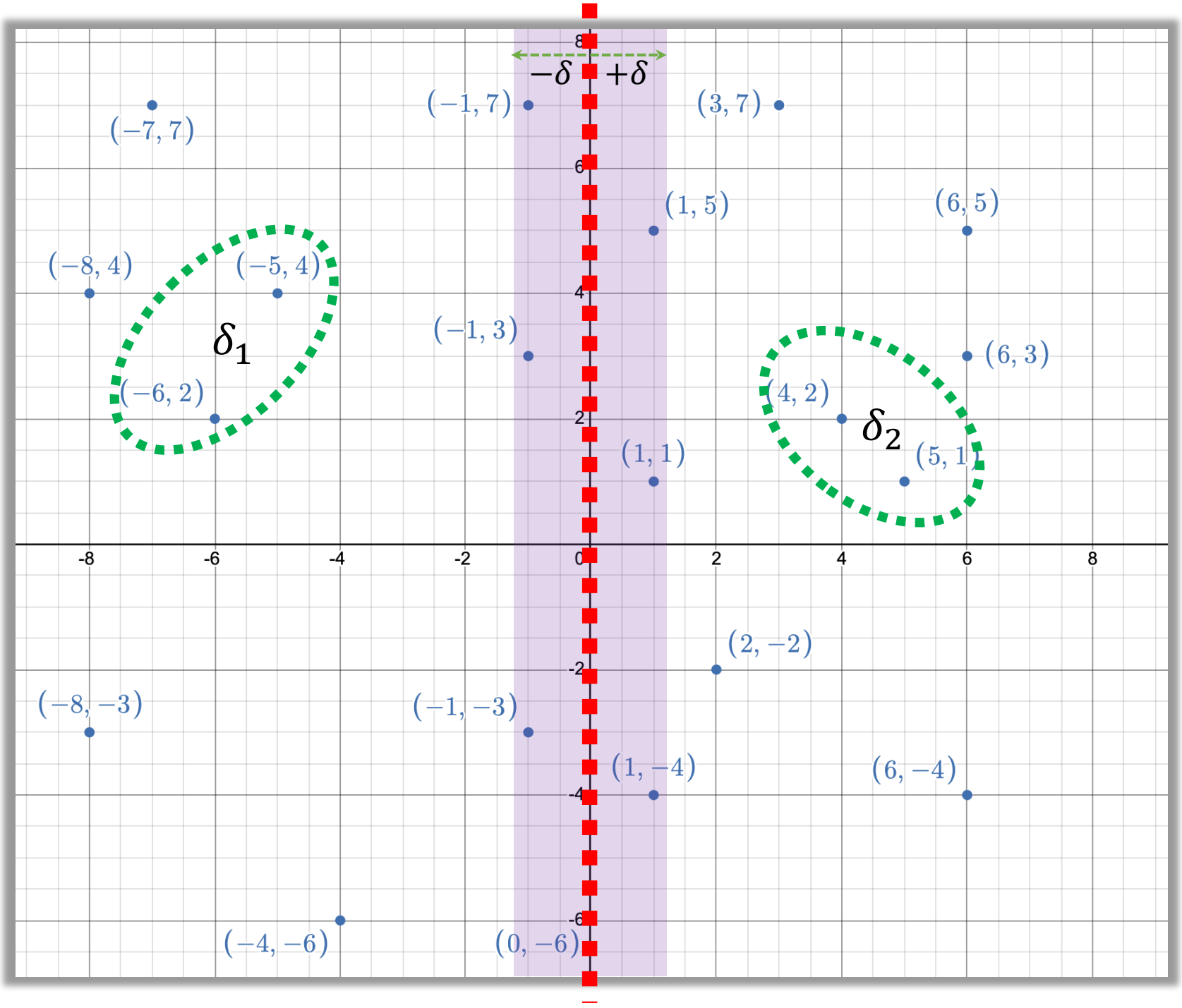

мы внутри шага рекурсии

## COMBINE

Найти ближайшую пару по разные стороны от $x_{\text{mid}} = 0$

[Q] Может сопоставить все точки слева со всеми справа от $x_{\text{mid}} = 0$?

Выполним $n^2/4$ операций, что не позволить достичь общей сложности $O(n \log n)$




мы внутри шага рекурсии

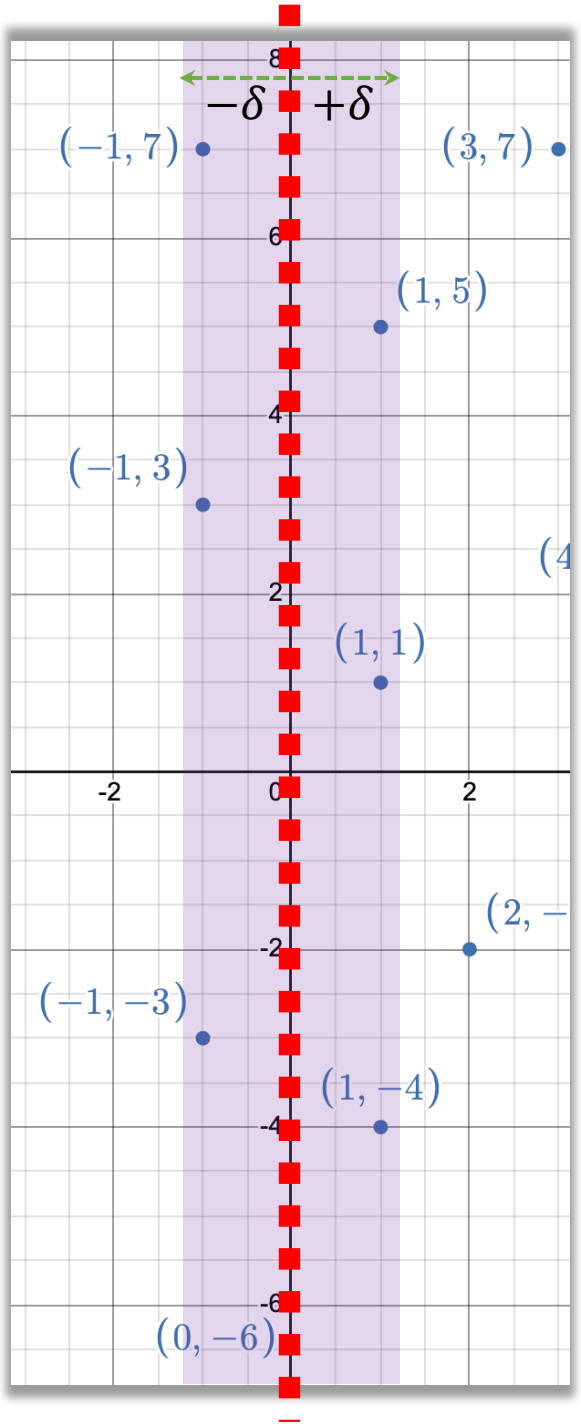
## COMBINE

Найти ближайшую пару по разные стороны от $x_{\text{mid}} = 0$

[Q] Может сопоставить все точки слева со всеми справа от $x_{\text{mid}} = 0$?

Нас интересуют только те точки, которые попадают в $[x_{mid} - \delta, x_{mid} + \delta]$

$\delta = \min(\delta_1, \delta_2) \approx 1.41$



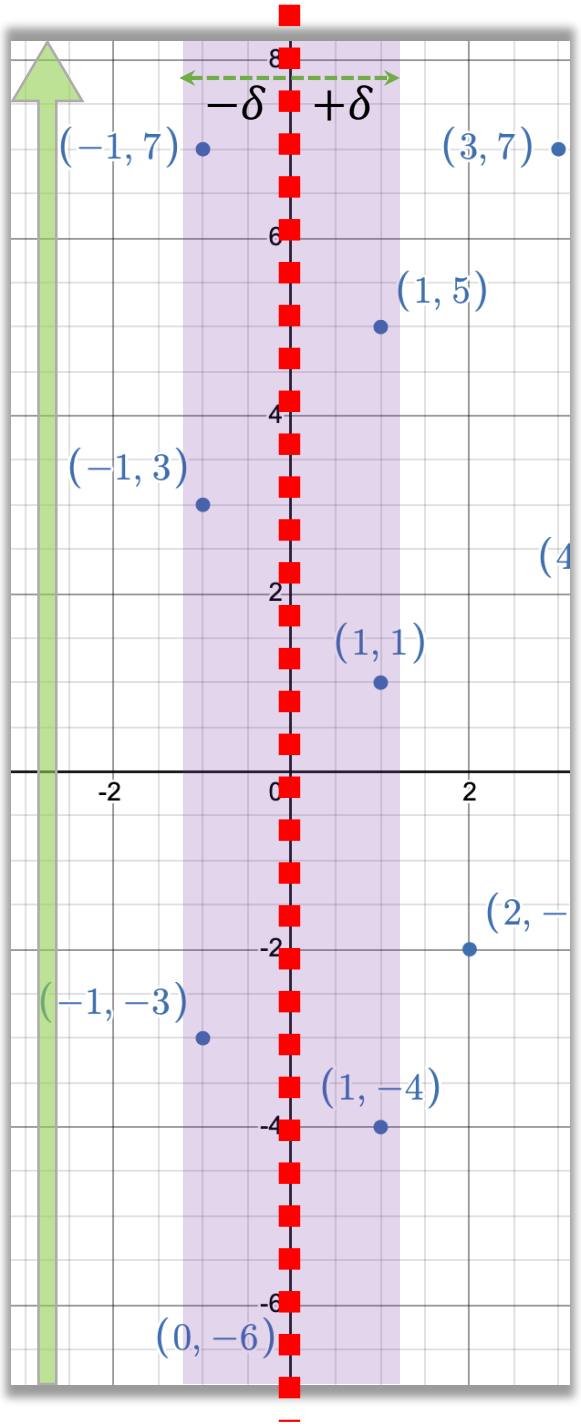

мы внутри шага рекурсии

## COMBINE

Найти ближайшую пару по разные стороны от $x_{\text{mid}} = 0$

Рассматриваем точки в полосе $[x_{mid} - 1.41, x_{mid} + 1.41]$

- Практика показывает, что выбор полосы существенно снижает количество рассматриваемых точек
  однако все $n$ точек также могут оказаться в этой полосе...




мы внутри рекурсии

## COMBINE

Найти ближайшую пару по разные стороны от $x_{\text{mid}} = 0$

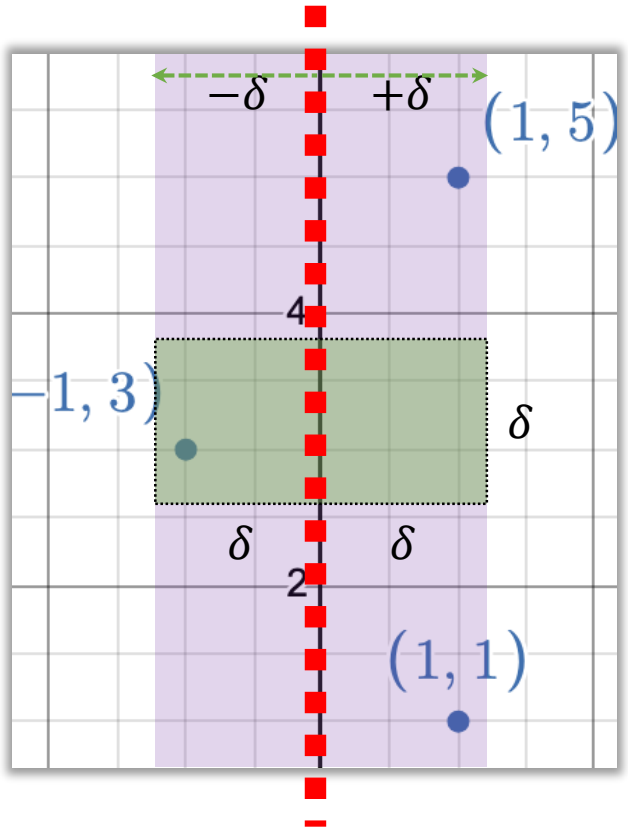
Рассматриваем точки в полосе $[x_{mid} - 1.41, x_{mid} + 1.41]$

- Практика показывает, что выбор полосы существенно снижает количество рассматриваемых точек
  однако все $n$ точек также могут оказаться в этой полосе...

- Внутри полосы рассматриваем упорядоченные по $y$ точки, $y$-координаты которых отличаются не более чем на $1.41$
  - точки $(0, -6)$ и $(1, -4)$ точно не подходят
  - точки $(-1, -3)$ и $(1, -4)$ возможно подходят
  - ...

  не нужно попарно сравнивать все точки в полосе — для каждой нужно проверить ограниченное число соседей

мы внутри рекурсии




## COMBINE

Найти ближайшую пару по разные стороны от $x_{\text{mid}} = 0$

Рассматриваем точки в полосе $[x_{mid} - 1.41, x_{mid} + 1.41]$

- Внутри полосы рассматриваем упорядоченные по $y$ точки, $y$-координаты которых отличаются не более чем на $1.41$
  ненужно попарно сравнивать все точки в полосе — для каждой нужно проверить ограниченное число соседей
- Интересующие нас точки-соседи всегда будут лежать в прямоугольнике $\delta \times 2\delta$ — не более 8 точек!

мы внутри рекурсии

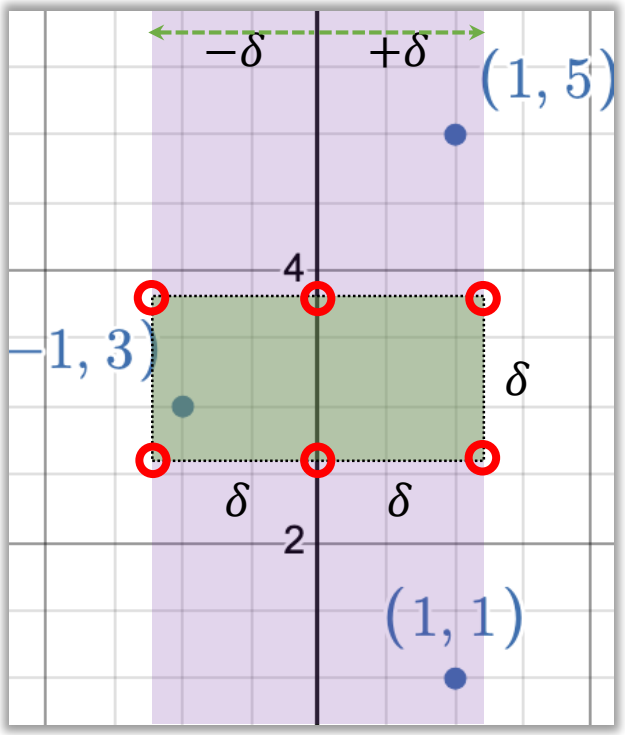
## COMBINE

Найти ближайшую пару по разные стороны от $x_{\text{mid}} = 0$

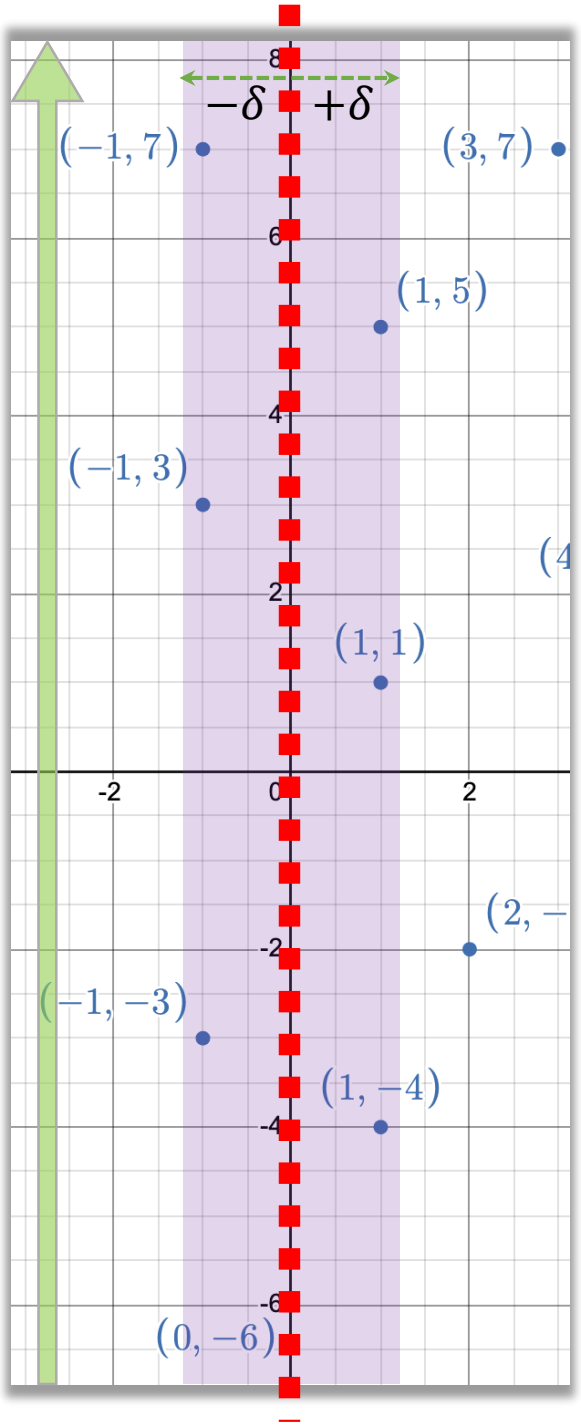
Рассматриваем точки в полосе $[x_{mid} - 1.41, x_{mid} + 1.41]$

- Внутри полосы рассматриваем упорядоченные по $y$ точки, $y$-координаты которых отличаются не более чем на $1.41$
  не нужно попарно сравнивать все точки в полосе — для каждой нужно проверить ограниченное число соседей
- Интересующие нас точки-соседи всегда будут лежать в прямоугольнике $\delta \times 2\delta$ — не более 8 точек!

Тогда, для каждой точки в полосе потребуется проверить расстояние не более чем до 7 следующих за ней [в массиве, отсортированном по $y$] ⇒ линейные временные затраты



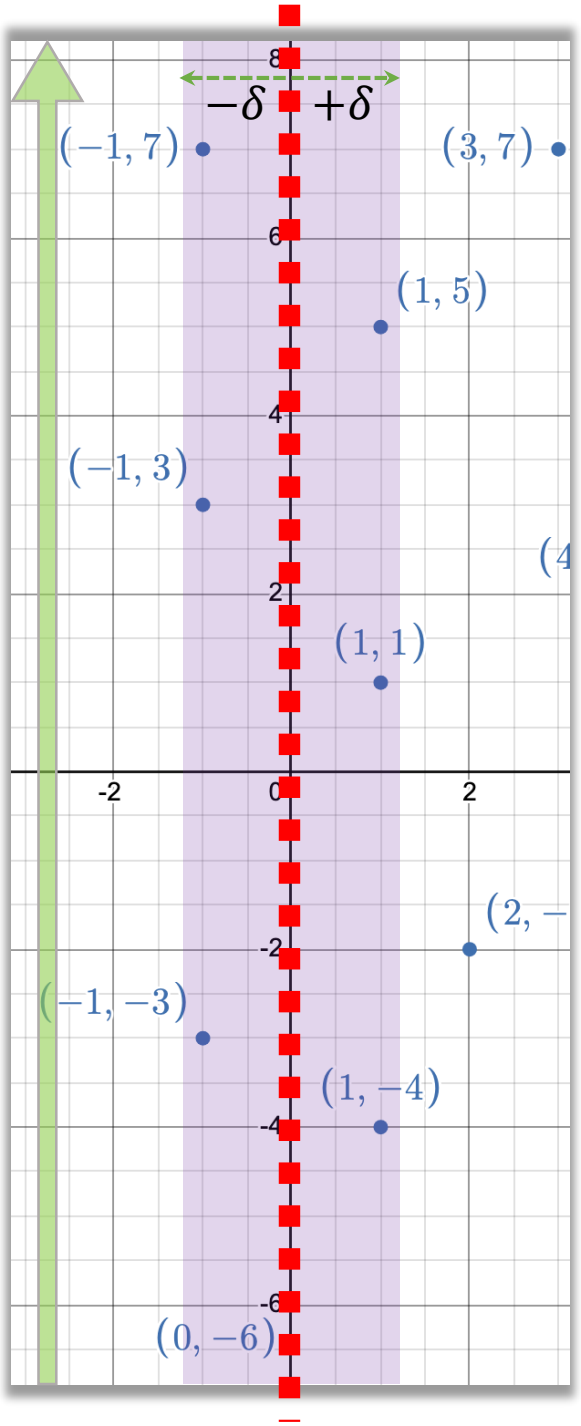

точки, лежащие на разделительной линии могут быть в 2 экземплярах




## DIVIDE уточнения

- Нам требуются точки, упорядоченные как по $x$, так и по $y$
- Сортировка за $O(n \log n)$ по $y$ внутри шага **COMBINE** не позволит достичь целевой сложности всего алгоритма
- Передаем в алгоритм две копии — $P^x_{\text{sorted}}$ и $P^y_{\text{sorted}}$

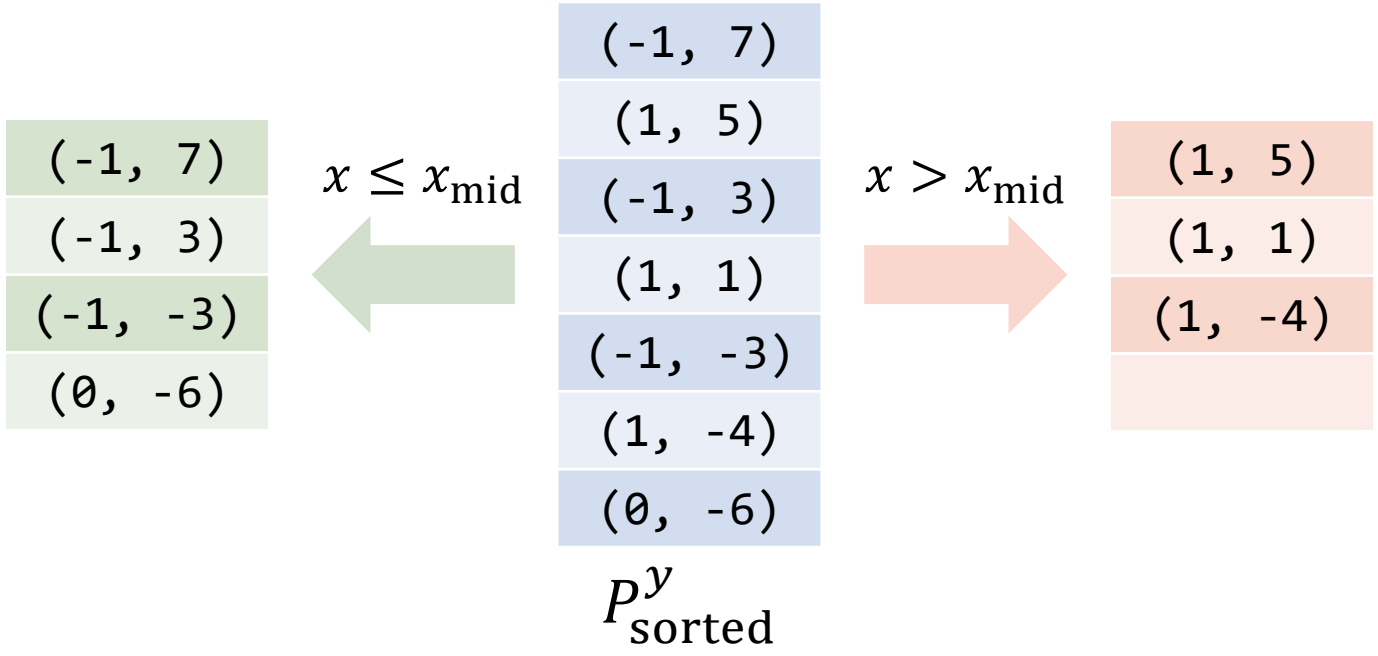
[Q] Как для половин массива $P^x_{\text{sorted}}$ [по $x_{\text{mid}}$] получить те же половины, но отсортированные по $y$?




## DIVIDE уточнение

- Нам требуются точки, упорядоченные как по $x$, так и по $y$
- Сортировка за $O(n \log n)$ по $y$ внутри шага **COMBINE** не позволит достичь целевой сложности всего алгоритма
- Передаем в алгоритм две копии — $P^x_{\text{sorted}}$ и $P^y_{\text{sorted}}$

[Q] Как для половин массива $P^x_{\text{sorted}}$ [по $x_{\text{mid}}$] получить те же половины, но отсортированные по $y$? «REVERSE MERGE»



Задача 2

# CLOSEST PAIR 2D DaC

| | **closest_pair_2D**($P^x_{\text{sorted}}$, $P^y_{\text{sorted}}$) | $T(n) = \ldots$ |
|---|---|---|
| 1 | **if** $n \leq 3$ **return** . . . | $O(1)$ |
| 2 | $m = n/2$ – индекс медианы массива $P^x_{\text{sorted}}$ | $O(n)$ |
| 3 | $P^x_1 = \{p^x_1, \ldots, p^x_m\}, P^x_2 = \{p^x_{m+1}, \ldots, p^x_n\}$ | |
| 4 | $\{P^y_1, P^y_2\} \leftarrow$ **reverse_merge**($P^y_{\text{sorted}}$, $P^x_{\text{sorted}}$[m]) | |
| 5 | $\delta_1 =$ **closest_pair_2D**($P^x_1$, $P^y_1$) | $2T(n/2)$ |
| 6 | $\delta_2 =$ **closest_pair_2D**($P^x_2$, $P^y_1$) | |
| 7 | $\delta = \min(\delta_1, \delta_2)$ | |
| 8 | strip = **create_strip**($P^y_{\text{sorted}}$, $\delta$) | $O(n)$ |
| 9 | **compare_within_strip**(strip, $\delta$) | |
| 10 | **return** $\delta$ | |

Задача 2

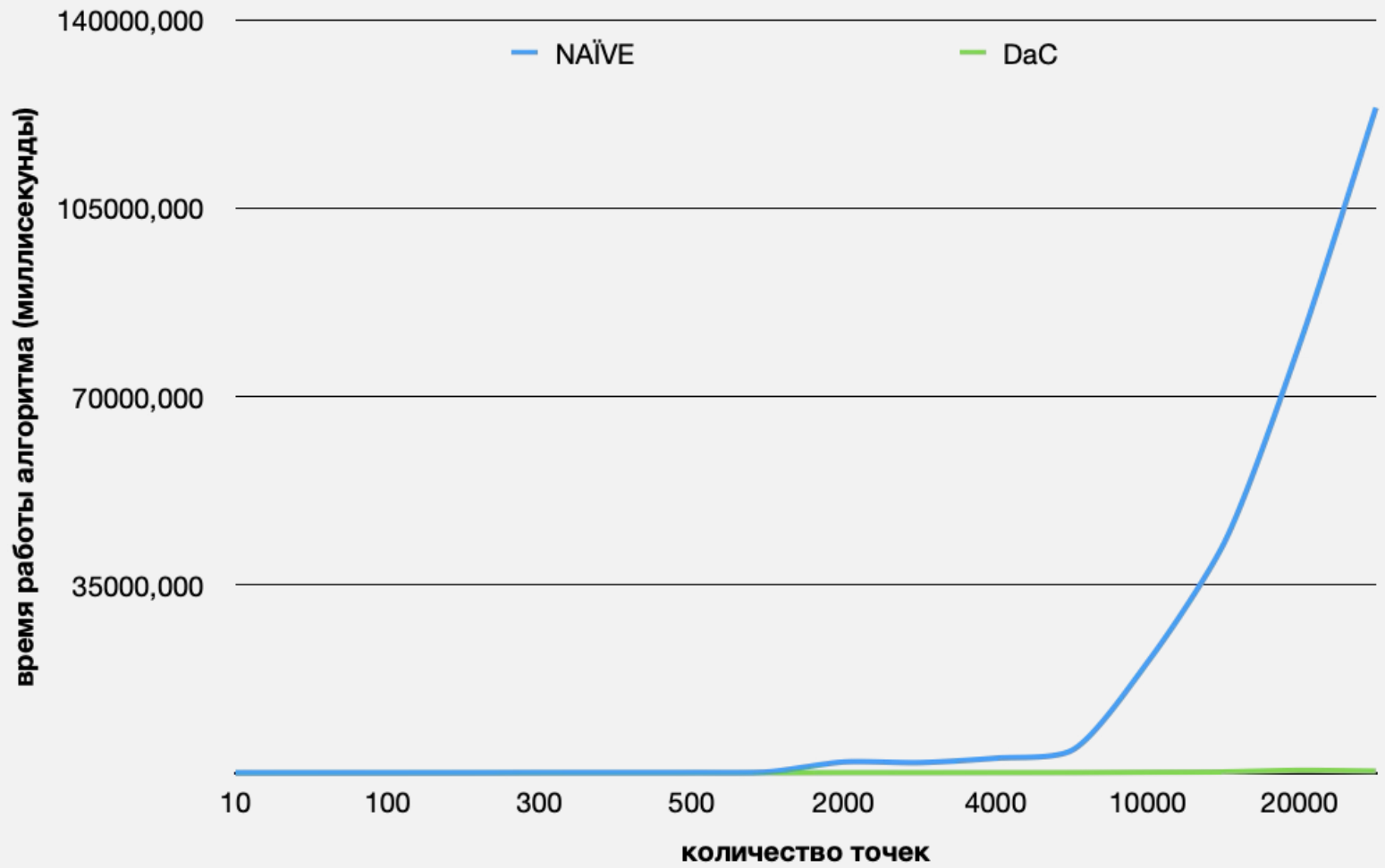
# CLOSEST PAIR 2D DaC

| | compare_within_strip(strip, $\delta$) | $T(n)$ |
|---|---|---|
| 1 | **for** i = 0 **to** \|strip\| | $O(n)$ |
| 2 | j = i + n | |
| 3 | **while** (j < \|strip\| **AND** | |
| 4 | strip[j].y – strip[i].y < $\delta$) | |
| 5 | d = **dist**(strip[i], strip[j]) | $O(1)$ |
| 6 | if d < $\delta$ then $\delta$ = d | |
| 7 | j = j + 1 | |
| 8 | **end while** | |
| 9 | **end for** | |

внутренний цикл **while** проверит не более 7 соседей точки strip[i]

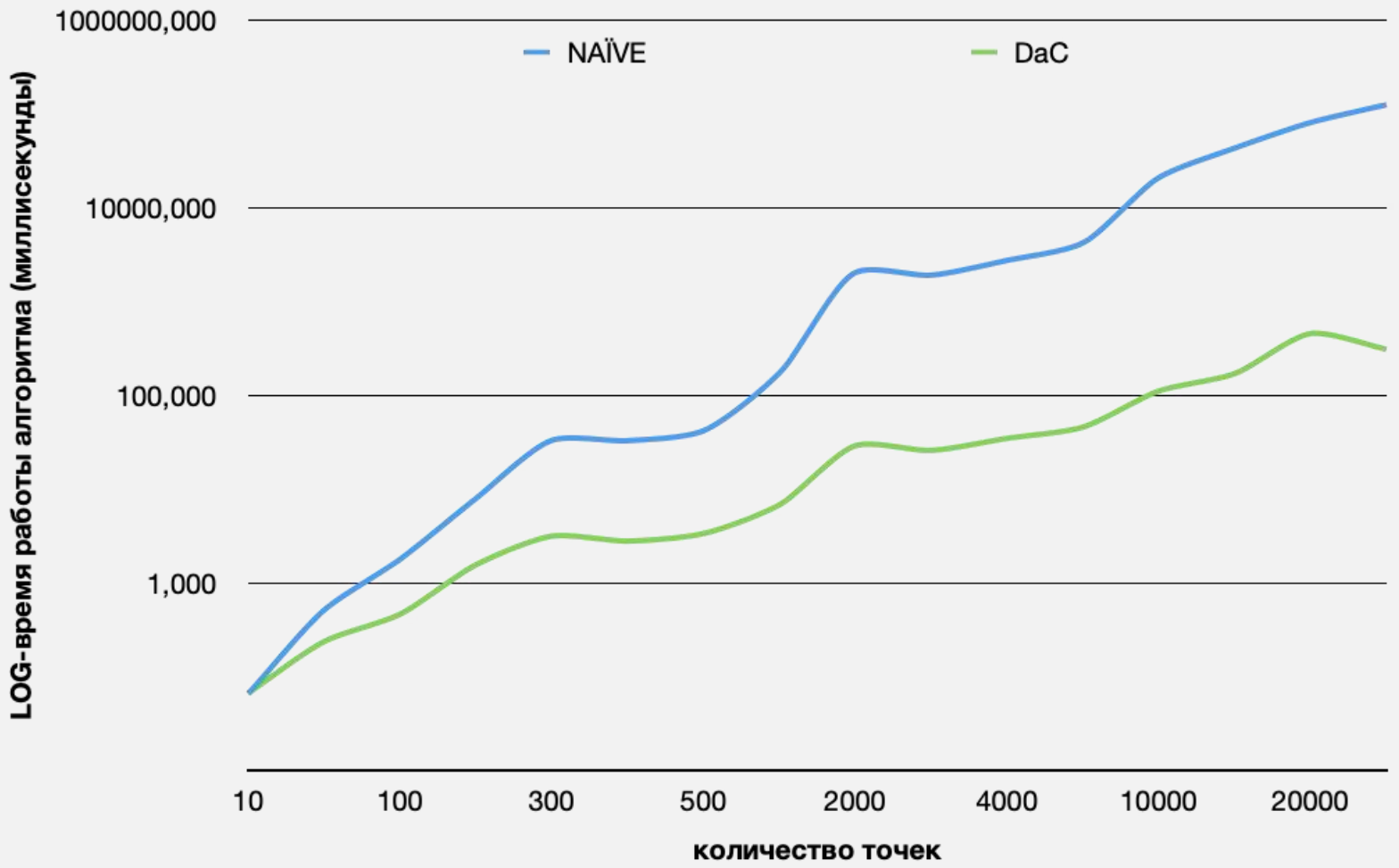
Задача 2

# CLOSEST PAIR 2D



Задача 2

# CLOSEST PAIR 2D



Задача 2





# CLOSEST PAIR dD

Идею формирования полосы **strip** можно распространить на большее число измерений

- компьютерная графика, игры, VR/AR
- геоинформационные системы
- машинное обучение и анализ данных
- сети и транспорт
- биоинформатика

...

# Слияние отсортированных списков K-WAY MERGE

грубая сила и друг за другом

Задача 2





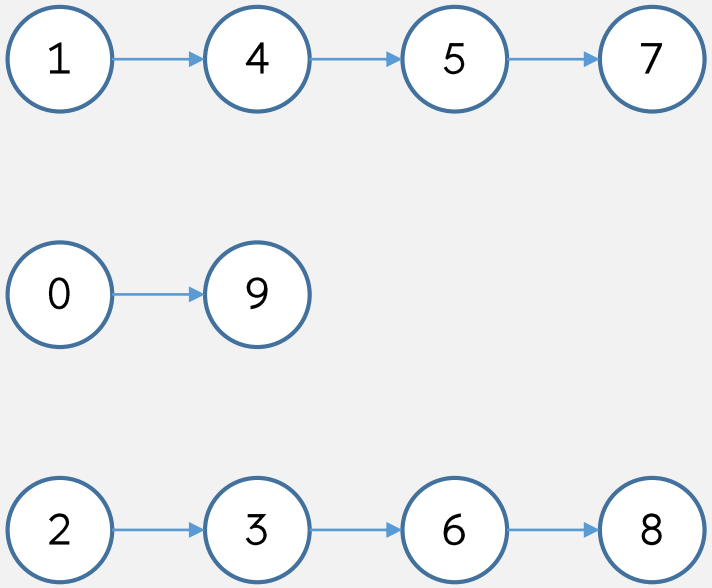
# K-WAY MERGE

Вход: $k$ связных списков, значения в узлах которых отсортированы по некоторому критерию.

Выход: связный список $L$, полученный в результате слияния входных списков, значения в узлах которого отсортированы по тому же критерию.

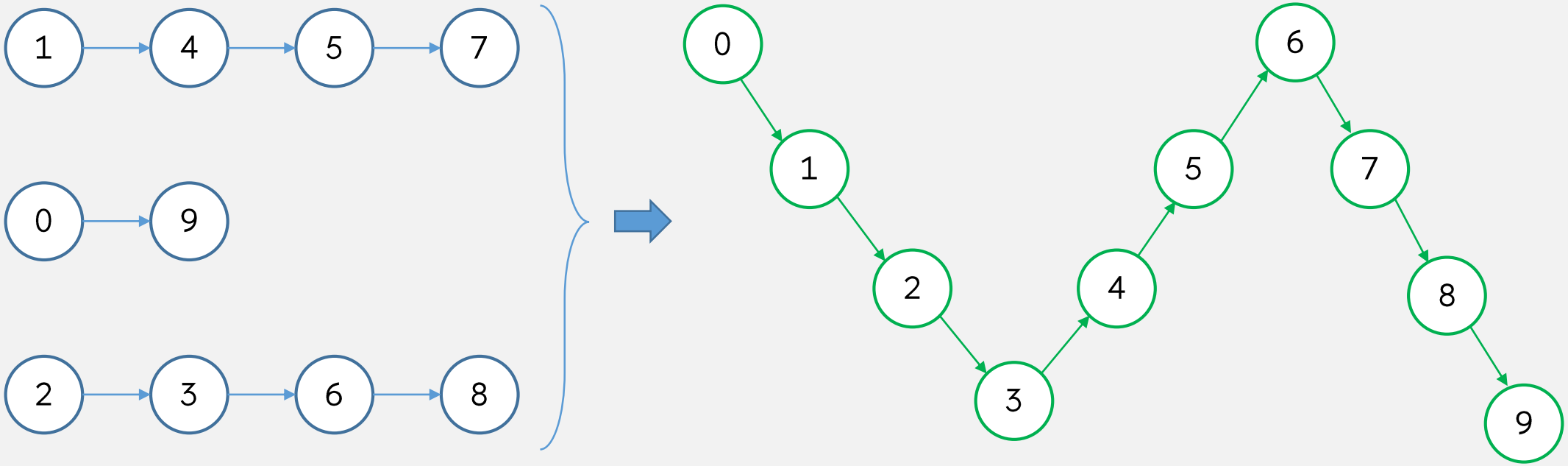
Задача 2

# K-WAY MERGE



Задача 2

# K-WAY MERGE



Задача 2





# K-WAY MERGE BRUTE FORCE

1. Обойти исходные списки и записать значения, содержащиеся в узлах списка, в массив $A$
2. Отсортировать массив $A$
3. Создать выходной список $L$ и переписать в него значения из массива $A$

Задача 2



# K-WAY MERGE BRUTE FORCE

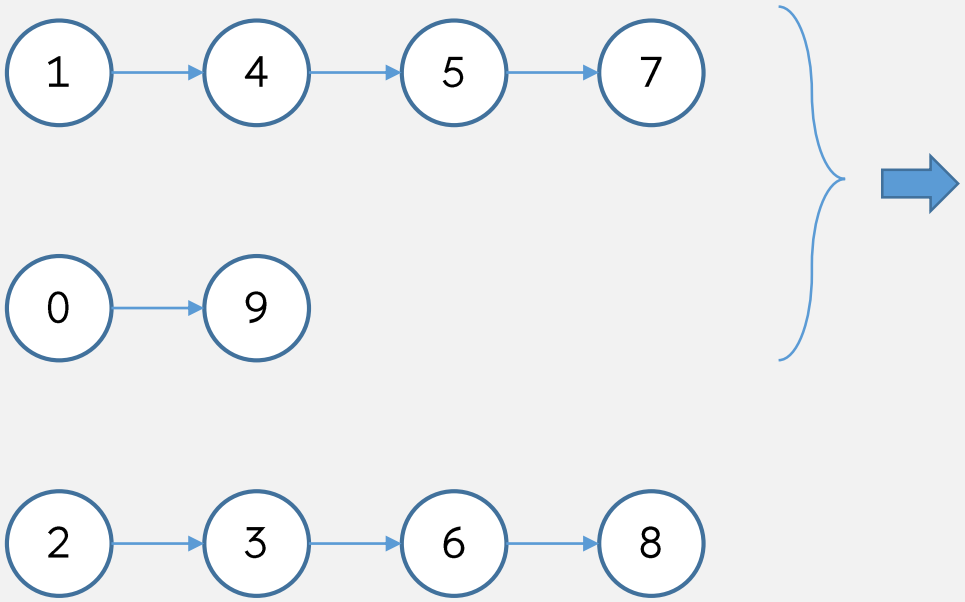
1. Обойти исходные списки и записать значения, содержащиеся в узлах списка, в массив $A$
2. **Отсортировать массив $A$**
3. Создать выходной список $L$ и переписать в него значения из массива $A$

**Сложность сортировки доминирует** над любыми другими шагами этого варианта алгоритма

$T(n) = O(g(n))$, где $O(g(n))$ — сложность сортировки

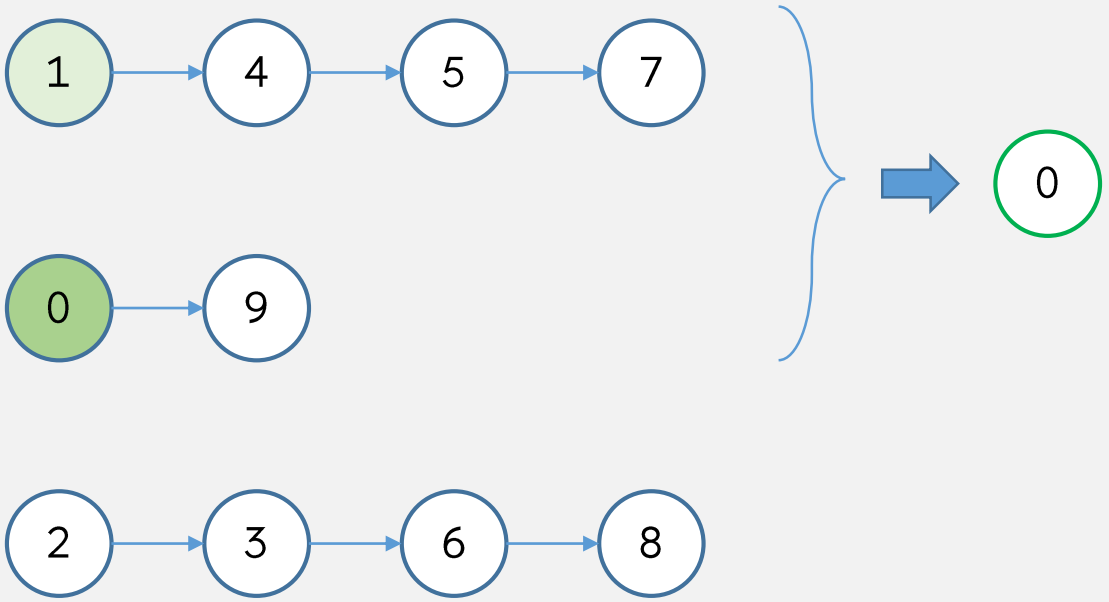
Задача 2

# K-WAY MERGE ONE-BY-ONE



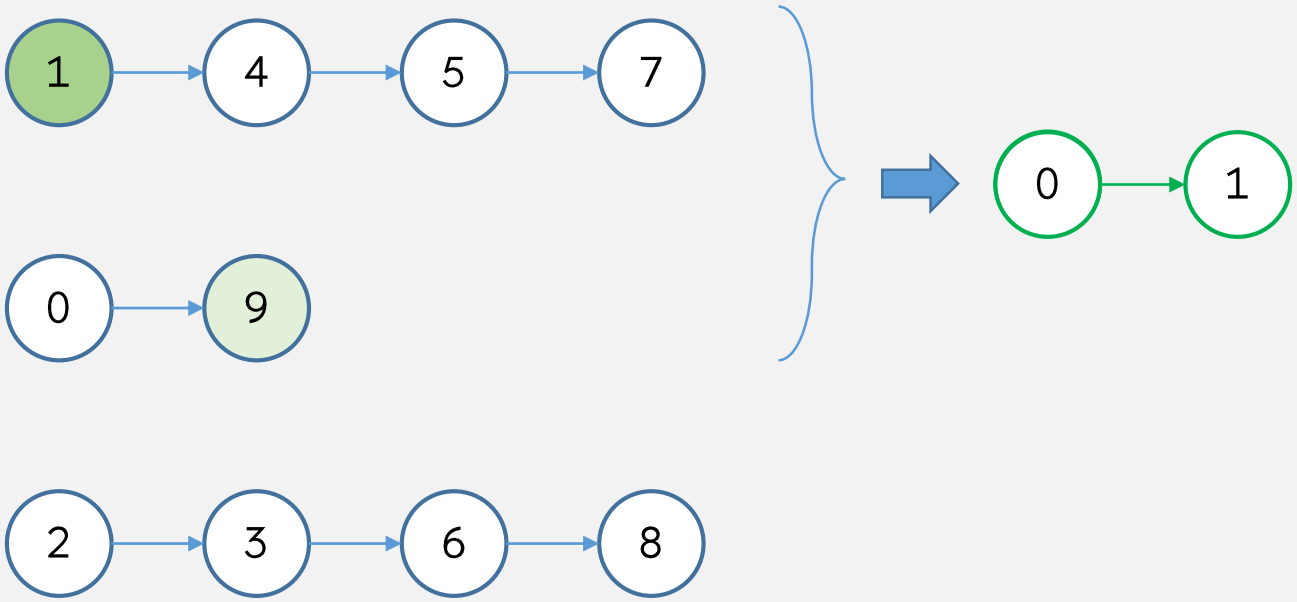
Задача 2

# K-WAY MERGE ONE-BY-ONE



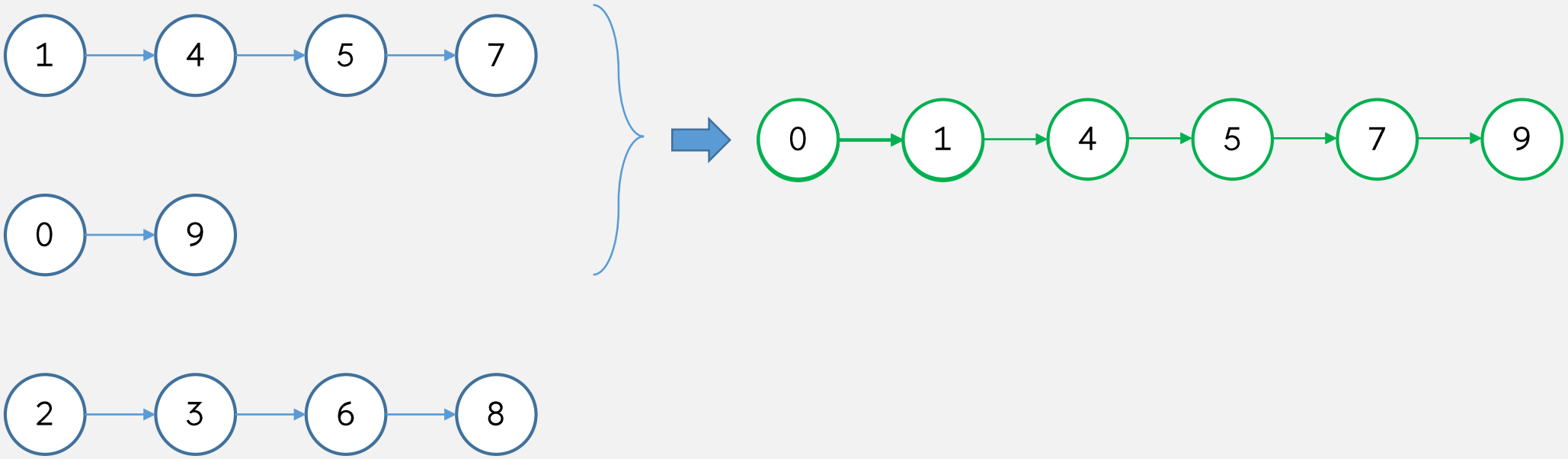
Задача 2

# K-WAY MERGE ONE-BY-ONE



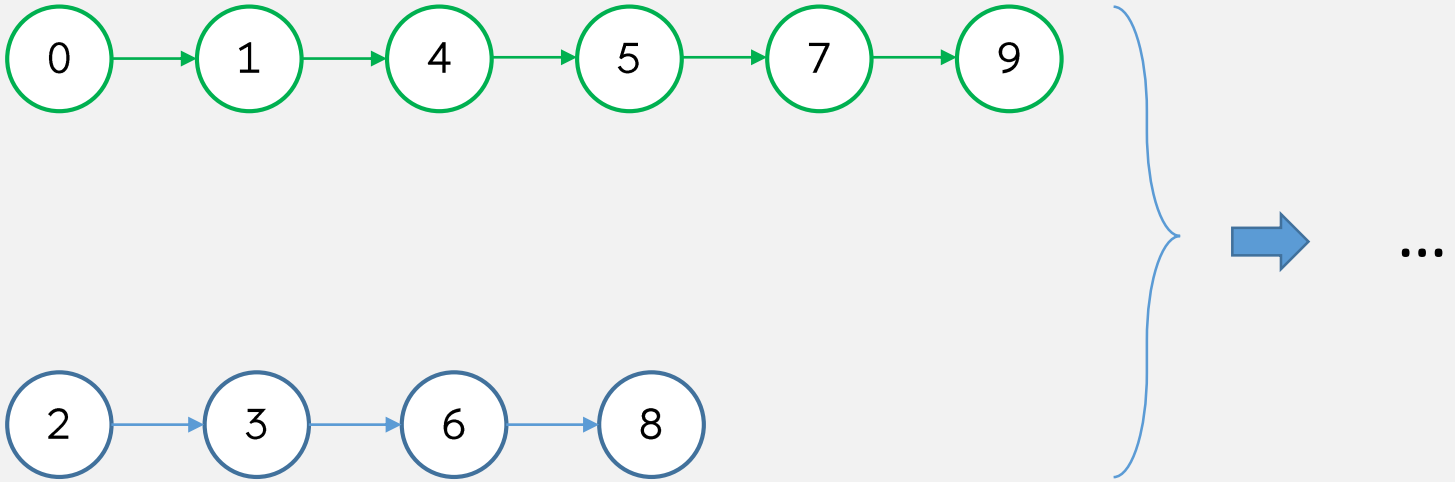
Задача 2

# K-WAY MERGE ONE-BY-ONE



Задача 2

# K-WAY MERGE ONE-BY-ONE



Задача 2



# K-WAY MERGE ONE-BY-ONE

**k_way_merge_one_by_one(lists, k)**

```
1  if k = 0 return null
2  if k = 1 return lists[0]
3  head = merge_sorted_lists(lists[0], lists[1])
4  for i = 2 to k
5      head = merge_sorted_lists(head, lists[i])
6  return head
```

Функция для базового случая — **merge_sorted_lists**

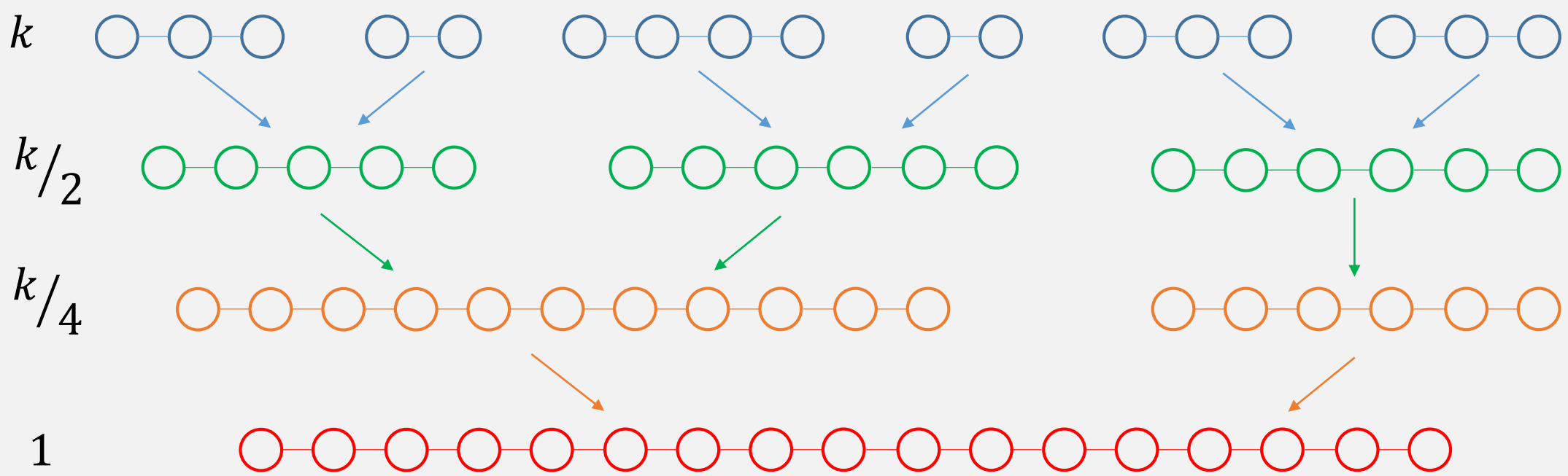
[Q] Какова сложность этого варианта алгоритма?

# Слияние отсортированных списков K-WAY MERGE

## DaC-подход

Задача 2

# K-WAY MERGE DaC




Выполняем попарное слияние списков на каждом шаге

Задача 2




# K-WAY MERGE DaC

**k_way_merge_dac(lists)**

```
1  if lists.size() <= 1 return list[0]
2  mid = lists.size() / 2
3  left = lists[1], ..., lists[m]
4  right = lists[m + 1], ..., lists[k]
5  mergedLeft = k_way_merge_dac(left)
6  mergedRight = k_way_merge_dac(right)
7  return merge_sorted_lists(mergedLeft, mergedRight)
```

Функция **merge_sorted_lists** выполняется $\log_2 k$ раз

Общая временная сложность — $O(n \log k)$

Задача 2




# K-WAY MERGE DaC

**k_way_merge_dac(lists, k)**

```
1  if k = 0 return null
2  d = 1
3  while d < k
4      i = 0
5      while i + d < k
6          lists[i] = merge_sorted_lists(lists[i], lists[i + d])
7          i = i + d * 2
8      intrv *= 2
9  return lists[0]
```

**Нерекурсивный** вариант реализации DaC-подхода

Задача 2




# K-WAY MERGE

Оптимизация слияния отсортированных списков также достигается за счет использования бинарной кучи [heap]

- базы данных и поисковые системы
- обработка событий в реальном времени
- внешняя сортировка данных на диске
- системное программирование
- проектирование пользовательского интерфейса

...

# Резюме

1. Бинарный поиск по ответу

2. Поиск ближайших точек `CLOSEST PAIR dD`
   - $O(n \log n)$ — одномерный случай
   - $O(n \log n)$ — двумерный случай

3. Слияние отсортированных списков `K-WAY MERGE`
   - $O(n \log n)$ — грубая сила
   - $O(n\,k)$ — друг за другом
   - $O(n \log k)$ — попарное DaC-слияние