Алгоритмы и структуры данных

Структуры данных для геометрического поиска

CS Center, Новосибирск

Location/stabbing problem

Location problem:

- Заданы точки

Location/stabbing problem

Location problem:

- Заданы точки
- Запрос – это область

Location/stabbing problem

Location problem:

- Заданы точки
- Запрос – это область
  - найти точки, попадающие в данную область

Location/stabbing problem

Location problem:

- Заданы точки
- Запрос – это область
  - найти точки, попадающие в данную область

Stabbing problem:

- Заданы области

Location/stabbing problem

Location problem:

- Заданы точки
- Запрос – это область
  - найти точки, попадающие в данную область

Stabbing problem:

- Заданы области
- Запрос – это точка

Location/stabbing problem

Location problem:

- Заданы точки
- Запрос – это область
  - найти точки, попадающие в данную область

Stabbing problem:

- Заданы области
- Запрос – это точка
  - найти области, в которые попадает эта точка

Одномерный случай location problem

- Заданы точки на прямой
- Запрос – это отрезок
  - найти точки, попадающие в отрезок

Одномерный случай location problem

- Заданы точки на прямой
- Запрос – это отрезок
  - найти точки, попадающие в отрезок

Варианты постановки задачи:

- Проверить, существуют ли такие точки (existence)

Одномерный случай location problem

- Заданы точки на прямой
- Запрос – это отрезок
  - найти точки, попадающие в отрезок

Варианты постановки задачи:

- Проверить, существуют ли такие точки (existence)
- Вычислить их количество (counting)

Одномерный случай location problem

- Заданы точки на прямой
- Запрос – это отрезок
  - найти точки, попадающие в отрезок

Варианты постановки задачи:

- Проверить, существуют ли такие точки (existence)
- Вычислить их количество (counting)
- Перечислить эти точки (reporting)

Одномерный случай location problem

- Заданы точки на прямой
- Запрос – это отрезок
  - найти точки, попадающие в отрезок

Варианты постановки задачи:

- Проверить, существуют ли такие точки (existence) – O(logN)
- Вычислить их количество (counting) – O(logN)
- Перечислить эти точки (reporting) – O(logN + A)

Одномерный случай stabbing problem

- Заданы отрезки на прямой
- Запрос – это точка
  - найти отрезки, в которые эта точка попадает

Одномерный случай stabbing problem

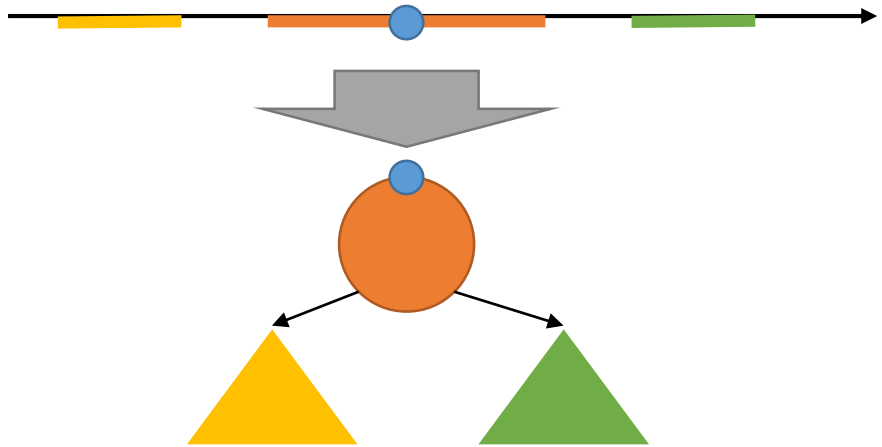
- Заданы отрезки на прямой
- Запрос – это точка
  - найти отрезки, в которые эта точка попадает

Структура для решения задачи – дерево интервалов

1D stabbing problem – дерево интервалов

В каждой вершине – список интервалов,

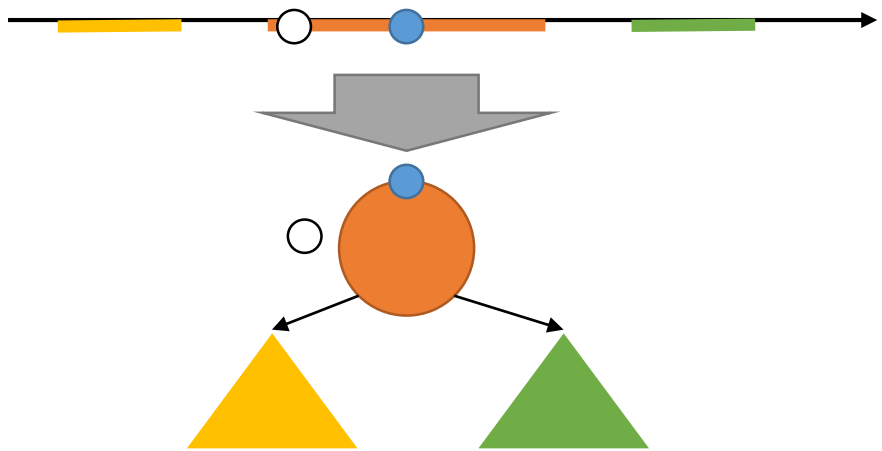
куда попадает точка для этой вершины



1D stabbing problem – дерево интервалов

В каждой вершине – список интервалов,

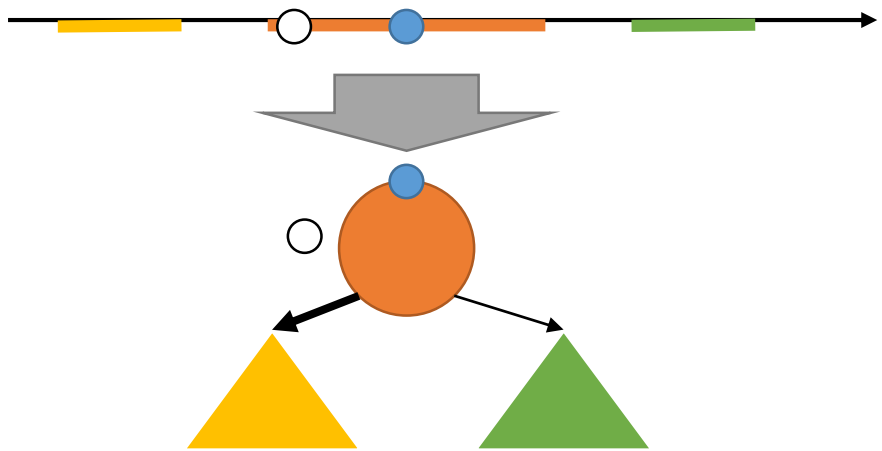
куда попадает точка для этой вершины



1D stabbing problem – дерево интервалов

В каждой вершине – список интервалов,

куда попадает точка для этой вершины

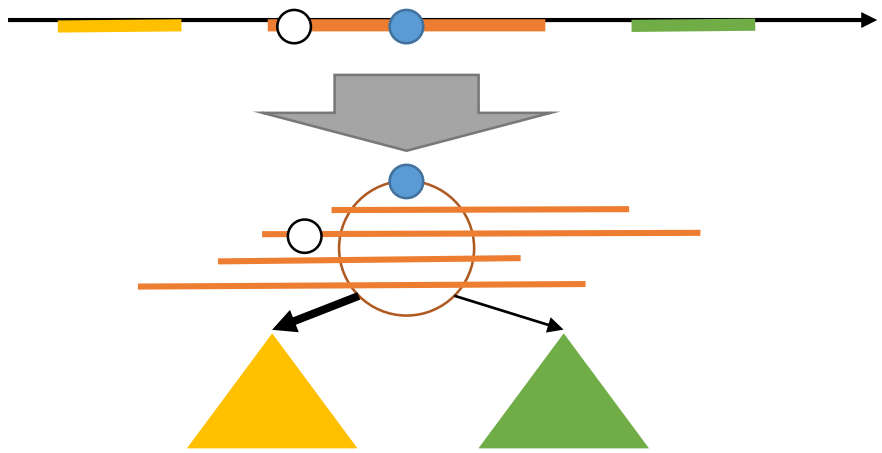


1D stabbing problem – дерево интервалов

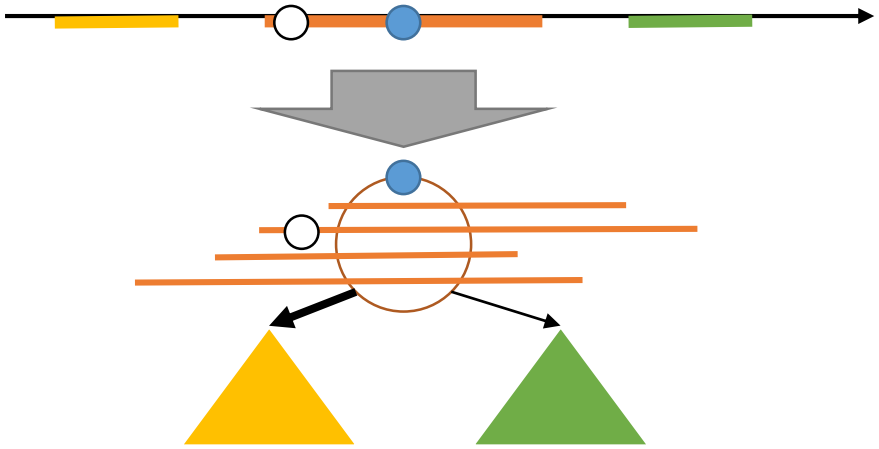
В каждой вершине – список интервалов,

куда попадает точка для этой вершины

- Список упорядочен



1D stabbing problem – дерево интервалов

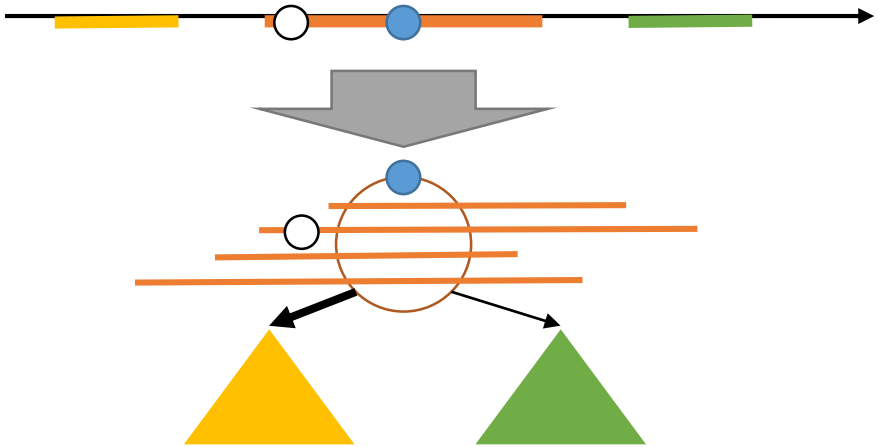


В каждой вершине – список интервалов,

куда попадает точка для этой вершины

- Список упорядочен
  - Каким образом его нужно упорядочить?

1D stabbing problem – дерево интервалов

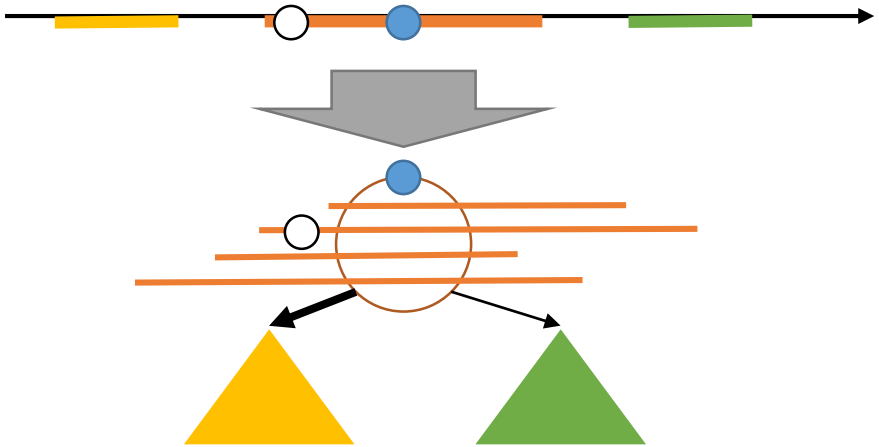


В каждой вершине – список интервалов,

куда попадает точка для этой вершины

- Список упорядочен
  - Каким образом его нужно упорядочить?
  - Нужно два упорядоченных списка

1D stabbing problem – дерево интервалов



В каждой вершине – список интервалов,

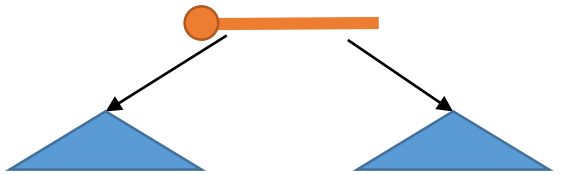
куда попадает точка для этой вершины

- Список упорядочен
  - Каким образом его нужно упорядочить?
  - Нужно два упорядоченных списка
- Сложность:
  - counting – O(logN)
  - reporting – O(logN + A)

1D stabbing problem – дерево интервалов на основе BST

В каждой вершине – один интервал (L_i, R_i)

Дерево – BST с L_i в качестве ключей

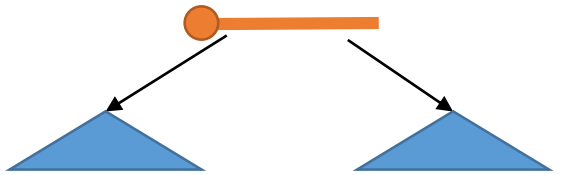


1D stabbing problem – дерево интервалов на основе BST

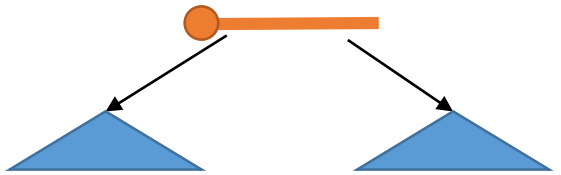
В каждой вершине – один интервал (L_i, R_i)

Дерево – BST с L_i в качестве ключей

В каждой вершине храним max{R_i в поддереве}



1D stabbing problem – дерево интервалов на основе BST



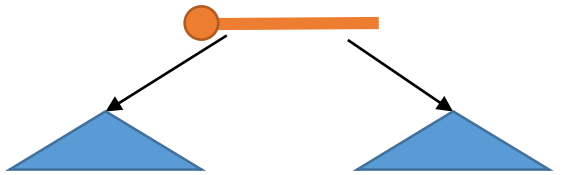
В каждой вершине – один интервал (L_i, R_i)

Дерево – BST с L_i в качестве ключей

В каждой вершине храним max{R_i в поддереве}

Как искать в таком дереве?

1D stabbing problem – дерево интервалов на основе BST

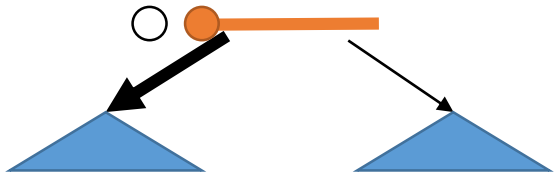


В каждой вершине – один интервал (L_i, R_i)

Дерево – BST с L_i в качестве ключей

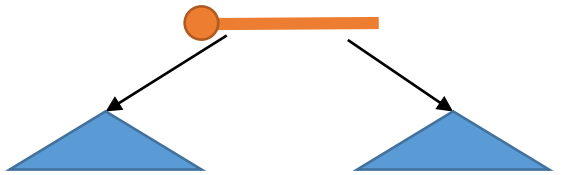
В каждой вершине храним max{R_i в поддереве}

Как искать в таком дереве?



1D stabbing problem – дерево интервалов на основе BST

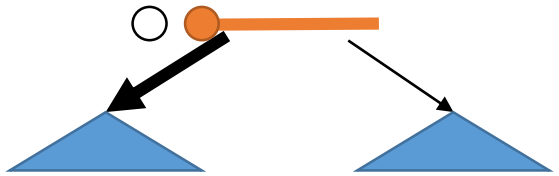
В каждой вершине – один интервал (L_i, R_i)

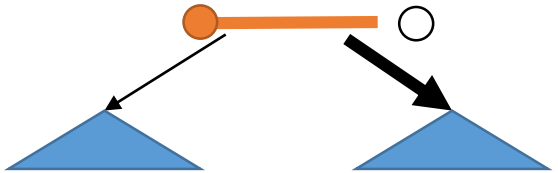


Дерево – BST с L_i в качестве ключей

В каждой вершине храним max{R_i в поддереве}

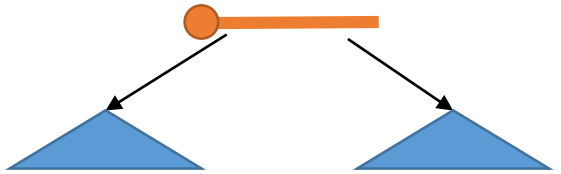
Как искать в таком дереве?





1D stabbing problem – дерево интервалов на основе BST

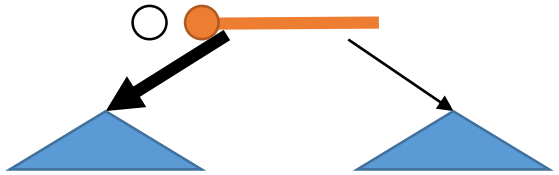
В каждой вершине – один интервал (L_i, R_i)

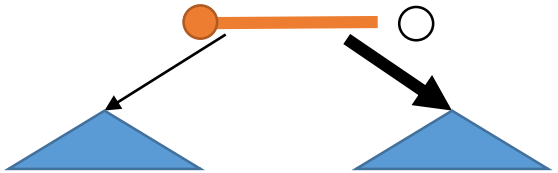


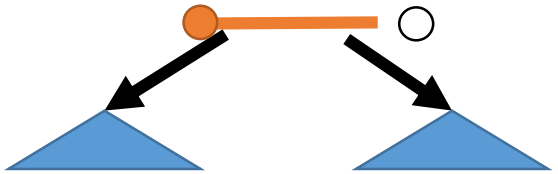
Дерево – BST с L_i в качестве ключей

В каждой вершине храним max{R_i в поддереве}

Как искать в таком дереве?





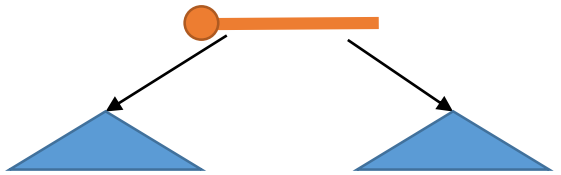


1D stabbing problem – дерево интервалов на основе BST

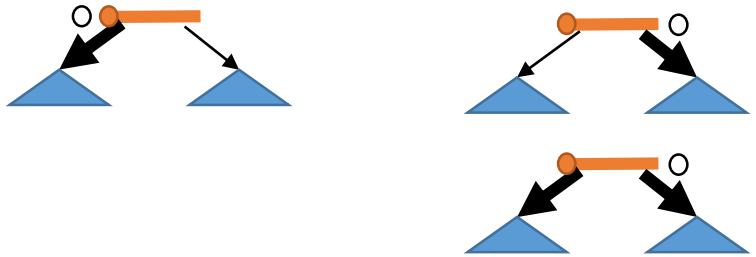
В каждой вершине – один интервал (L_i, R_i)

Дерево – BST с L_i в качестве ключей

В каждой вершине храним max{R_i в поддереве}

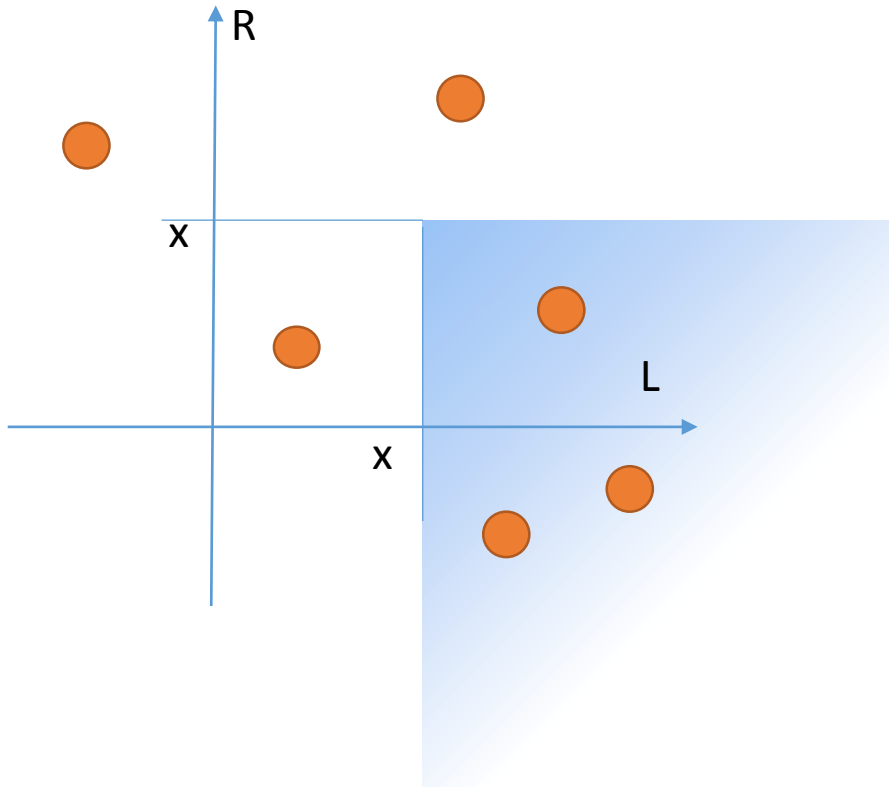


Сложность поиска: O(A logN)



1D stabbing <-> 2D location

$x \in [l, r] \Leftrightarrow (l \leq x, x \leq r)$



2D location: поиск в коридоре

- Запрос вида (left, right, bottom)
  - Найти точки, для которых left ≤ x ≤ right, bottom ≤ y

2D location: поиск в коридоре

- Запрос вида (left, right, bottom)
  - Найти точки, для которых left ≤ x ≤ right, bottom ≤ y

Структура для решения задачи – Priority Search Tree

2D location: Priority Search Tree

- Строим дерево:
  - В корень ставим точку с максимальным y

2D location: Priority Search Tree

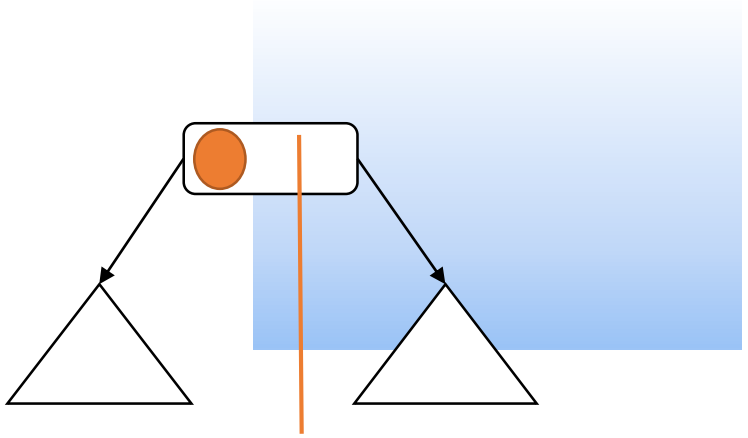
- Строим дерево:
  - В корень ставим точку с максимальным y
  - Остальные точки делим на две равные части

2D location: Priority Search Tree

- Строим дерево:
  - В корень ставим точку с максимальным y
  - Остальные точки делим на две равные части
  - Продолжаем рекурсивно

2D location: Priority Search Tree

- Строим дерево:
  - В корень ставим точку с максимальным y
  - Остальные точки делим на две равные части
  - Продолжаем рекурсивно
- В вершине дерева содержится:
  - точка
  - x – разделитель поддеревьев



2D location: поиск в прямоугольнике

- Запрос вида (left, right, bottom, top)
  - Найти точки, для которых left ≤ x ≤ right, bottom ≤ y ≤ top

2D location: поиск в прямоугольнике

- Запрос вида (left, right, bottom, top)
  - Найти точки, для которых left ≤ x ≤ right, bottom ≤ y ≤ top

Структура для решения задачи – kD Tree

kD Tree

- Строим дерево:
  - В корень ставим точку с x’ = медиана по x

kD Tree

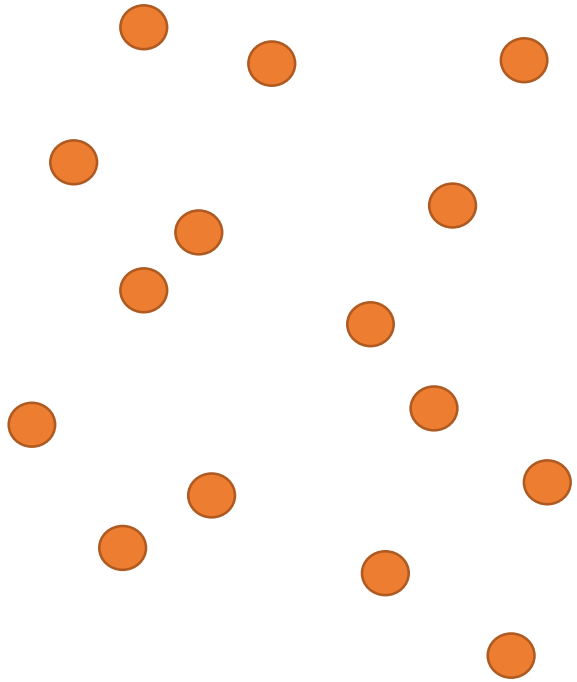
- Строим дерево:
  - В корень ставим точку с x’ = медиана по x
  - В левое поддерево идут точки с x ≤ x’, в правое - с x > x’

kD Tree

- Строим дерево:
  - В корень ставим точку с x’ = медиана по x
  - В левое поддерево идут точки с x ≤ x’, в правое - с x > x’
  - Продолжаем рекурсино
    - Но на следующем уровне меняем x на y

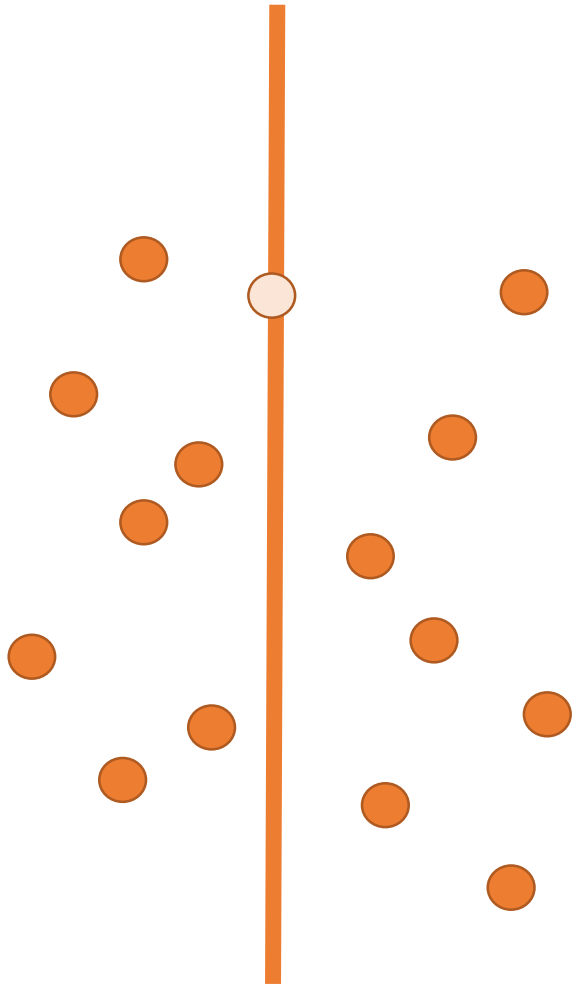
kD Tree

- Строим дерево:
  - В корень ставим точку с x’ = медиана по x
  - В левое поддерево идут точки с x ≤ x’, в правое - с x > x’
  - Продолжаем рекурсино
    - Но на следующем уровне меняем x на y



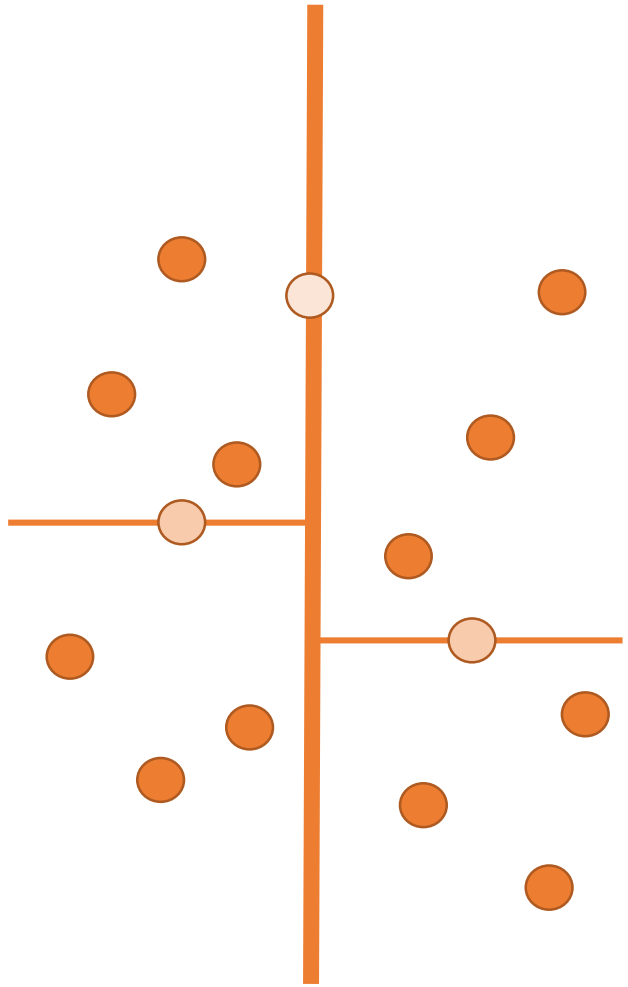
kD Tree

- Строим дерево:
  - В корень ставим точку с x’ = медиана по x
  - В левое поддерево идут точки с x ≤ x’, в правое - с x > x’
  - Продолжаем рекурсино
    - Но на следующем уровне меняем x на y



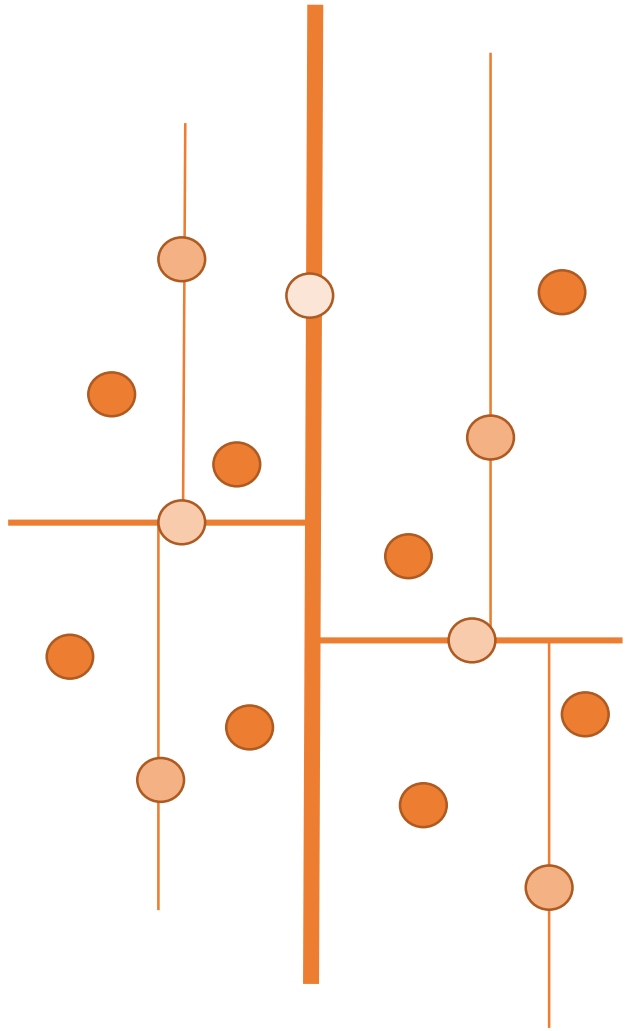
kD Tree

- Строим дерево:
  - В корень ставим точку с x’ = медиана по x
  - В левое поддерево идут точки с x ≤ x’, в правое - с x > x’
  - Продолжаем рекурсино
    - Но на следующем уровне меняем x на y



kD Tree

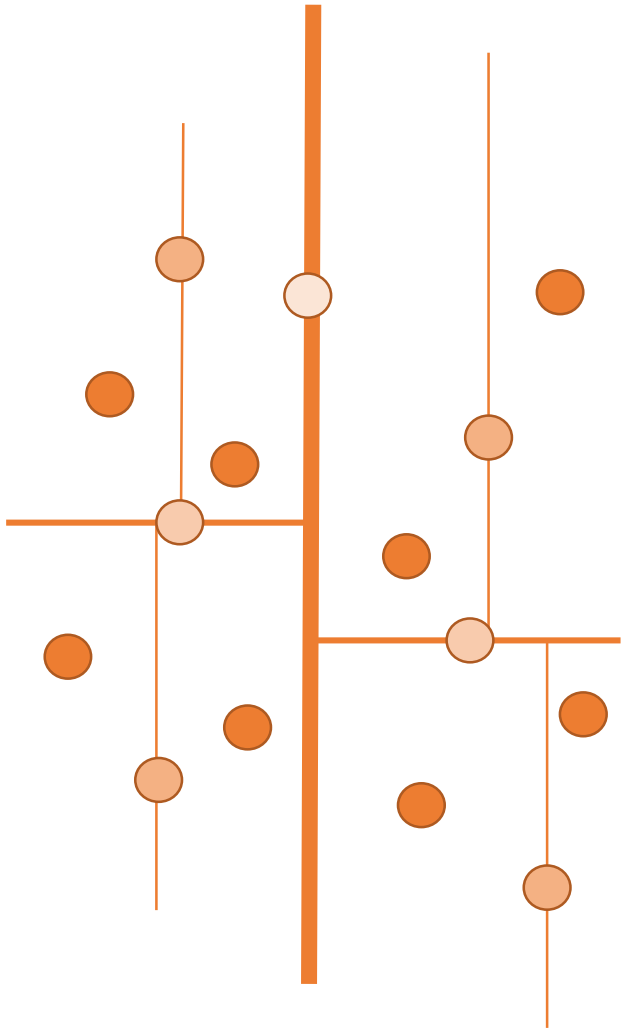
- Строим дерево:
  - В корень ставим точку с x’ = медиана по x
  - В левое поддерево идут точки с x ≤ x’, в правое - с x > x’
  - Продолжаем рекурсино
    - Но на следующем уровне меняем x на y



kD Tree

- Строим дерево:
  - В корень ставим точку с x’ = медиана по x
  - В левое поддерево идут точки с x ≤ x’, в правое - с x > x’
  - Продолжаем рекурсино
    - Но на следующем уровне меняем x на y
- Также годится для больших размерностей
- Для D = 2

  $T_{existence/counting} = O(n^{(1/2)})$



2D Tree

- Построим дерево отрезков по координате x

2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев

2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев

2D Tree

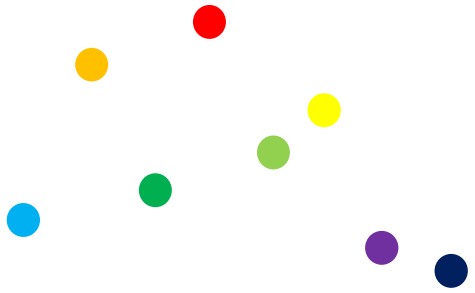
- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек

2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$

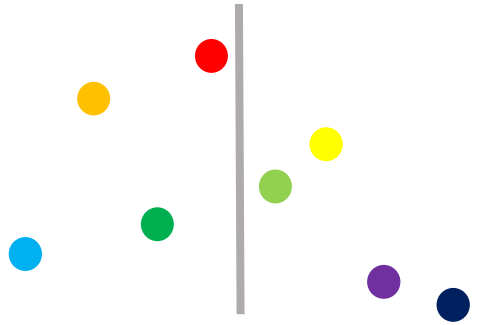
2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$



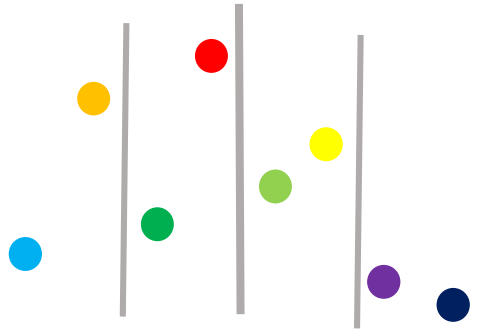
2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$



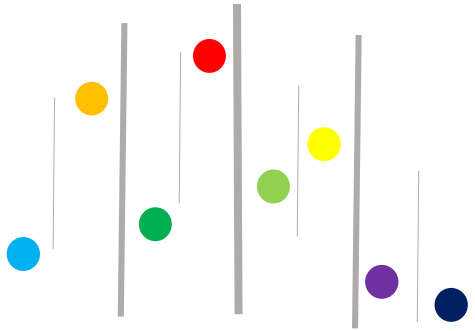
2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$



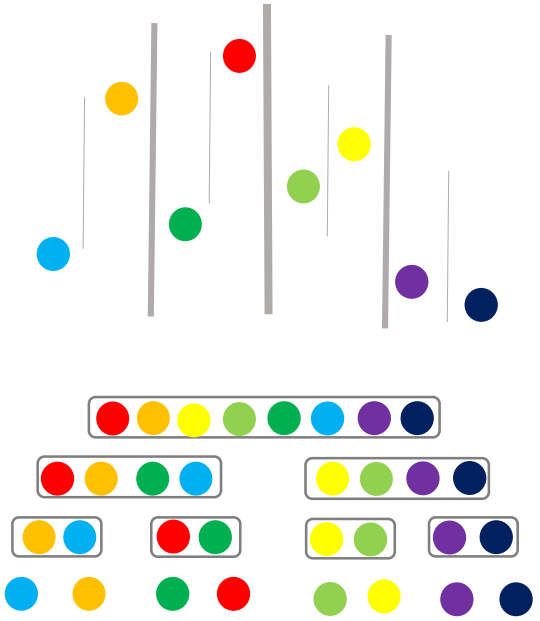
2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$



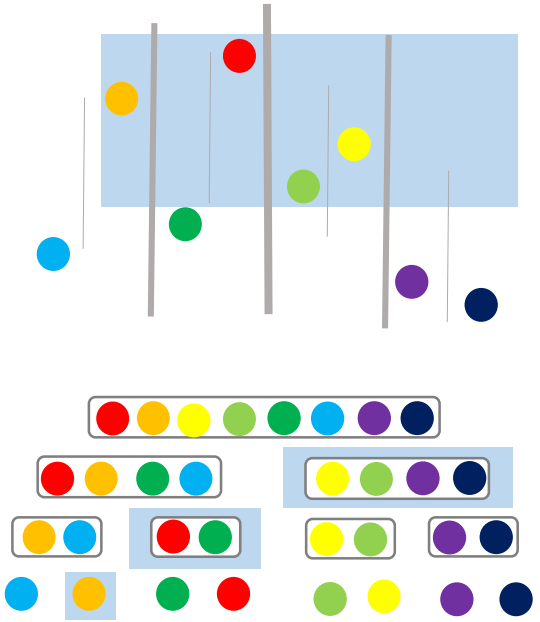
2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$



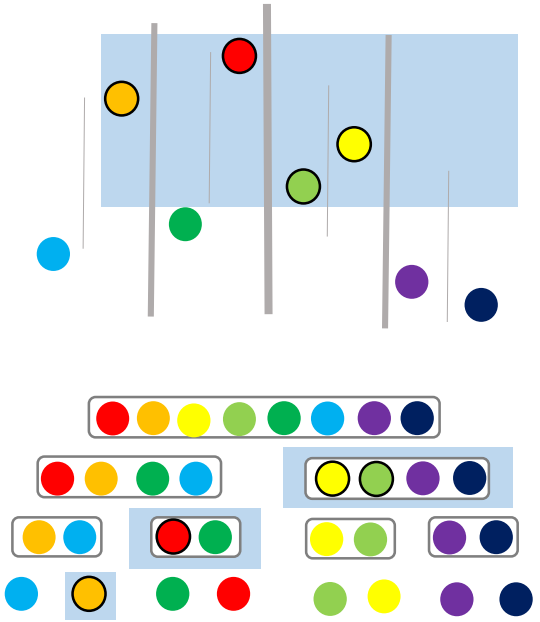
2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$



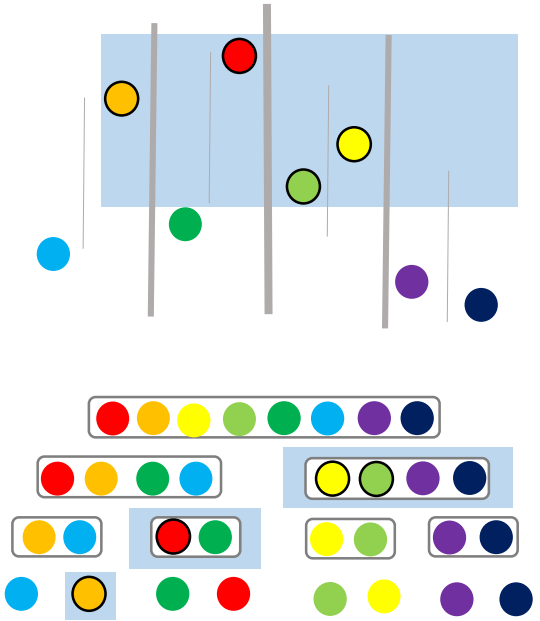
2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$



2D Tree

- Построим дерево отрезков по координате x
- Отрезок [left, right] включает O(logN) его поддеревьев
- Неэффективно идти вглубь этих поддеревьев
- В каждой вершине будем хранить упорядоченный по y список точек
- O(NlogN) памяти
- $T_{existence/counting} = O(\log^2 N)$
  - можно улучшить до O(logN)



Fractional cascading

- Выполнили бинпоиск в объединении двух массивов
- Как теперь избежать поиска в подмассиве?

Fractional cascading

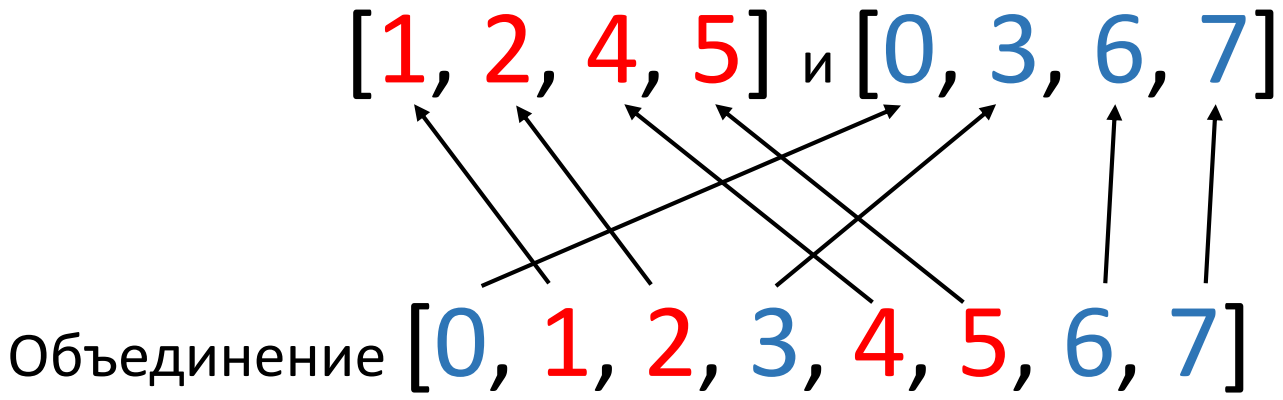
- Выполнили бинпоиск в объединении двух массивов
- Как теперь избежать поиска в подмассиве?

[1, 2, 4, 5] и [0, 3, 6, 7]

Объединение [0, 1, 2, 3, 4, 5, 6, 7]

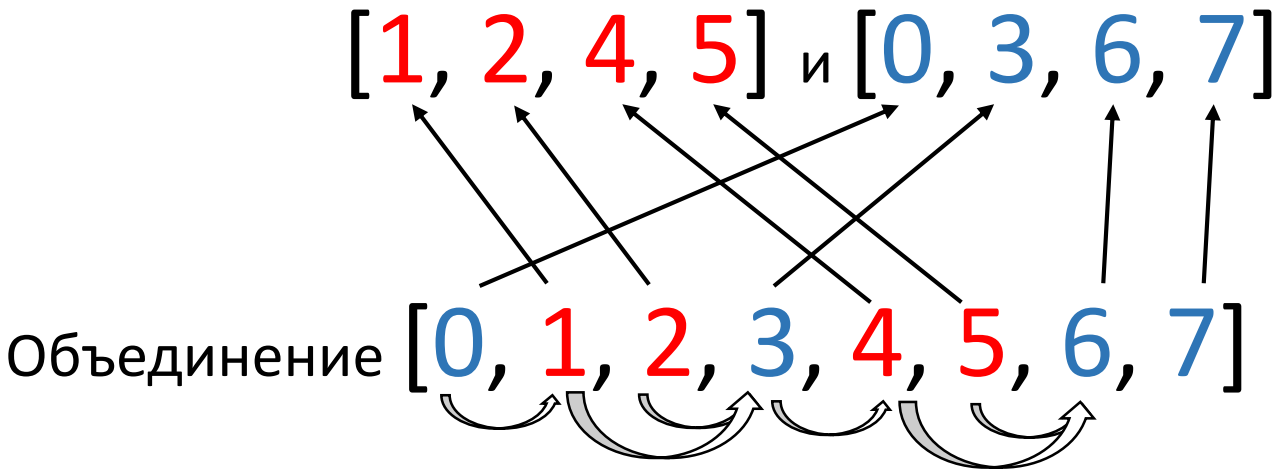
Fractional cascading

- Выполнили бинпоиск в объединении двух массивов
- Как теперь избежать поиска в подмассиве?



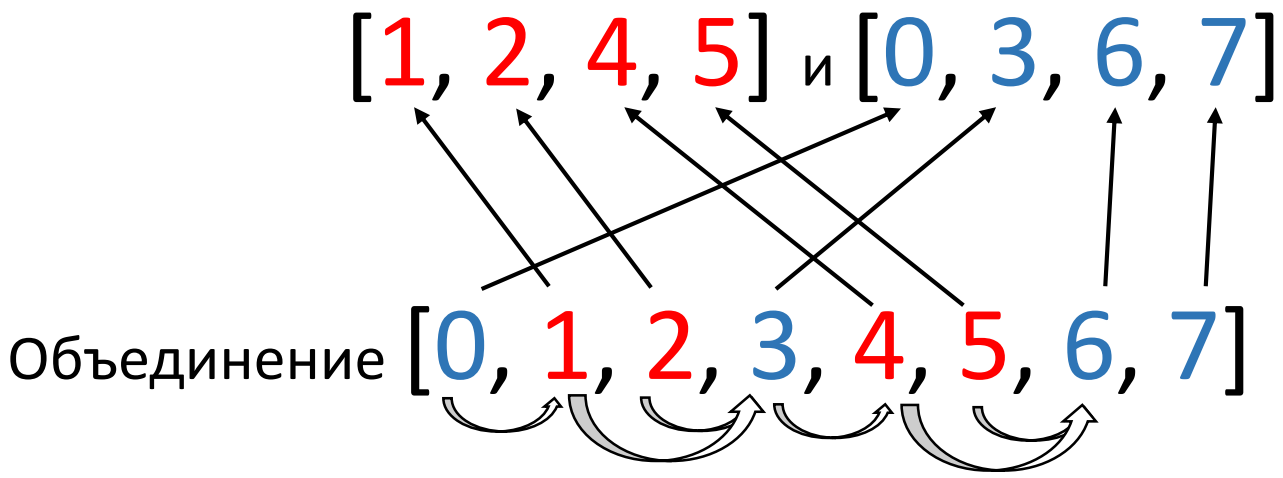
Fractional cascading

- Выполнили бинпоиск в объединении двух массивов
- Как теперь избежать поиска в подмассиве?



Fractional cascading

- Выполнили бинпоиск в объединении двух массивов
- Как теперь избежать поиска в подмассиве?



Переход на следующий уровень за O(1)