A Survey on Data Structures

Jeffrey Mei

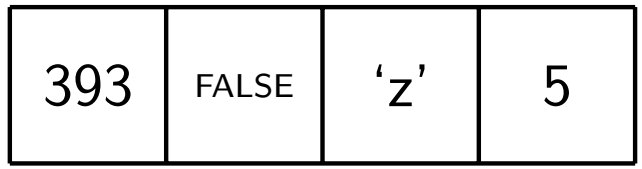
Introduction

- what are data structures?
  - data format for storage/organization
- why do we need data structures?
  - efficient storage
  - efficient retrieval
- needed for more complex tasks

Outline

- Arrays
- Linked Lists
  - Queues
  - Stacks
- Trees
  - Binary Search Trees
  - Heaps
- K-Dimensional Trees

Memory

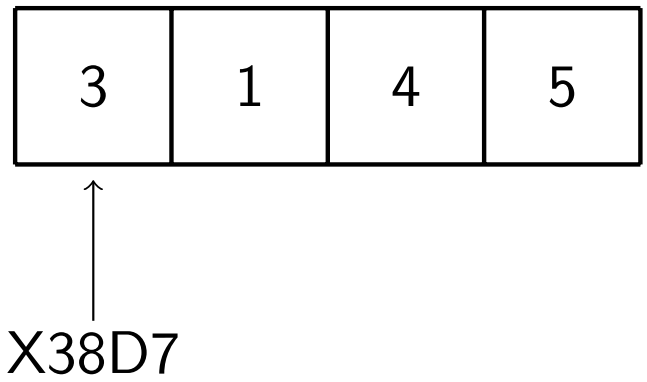


- memory addresses represented by hexadecimal code
- variables refer to addresses

Common Size of Data Types

- bool: 1 bit
- character: 8 bits (1 byte)
- integer: 32 bits (4 bytes)
- double: 64 bits (8 bytes)

Arrays



Array

- defined as an address in memory
- holds fixed number of values of single type
- all content is denoted as subsequent spaces from address
- $\mathrm{arr}\,[\mathrm{k}] = \mathrm{arr} + \mathrm{k} * \mathrm{DATATYPE\_SIZE}$

Restaurant Reservation Problem

- do not know how many values to store (e.g. logging events)
- concatenation: add elements to array
- not specifying space needed can be very inefficient

Restaurant Reservation Problem

- do not know how many values to store (e.g. logging events)
- concatenation: add elements to array
- not specifying space needed can be very inefficient

Restaurant Reservation Problem

- do not know how many values to store (e.g. logging events)
- concatenation: add elements to array
- not specifying space needed can be very inefficient

Restaurant Reservation Problem

- do not know how many values to store (e.g. logging events)
- concatenation: add elements to array
- not specifying space needed can be very inefficient

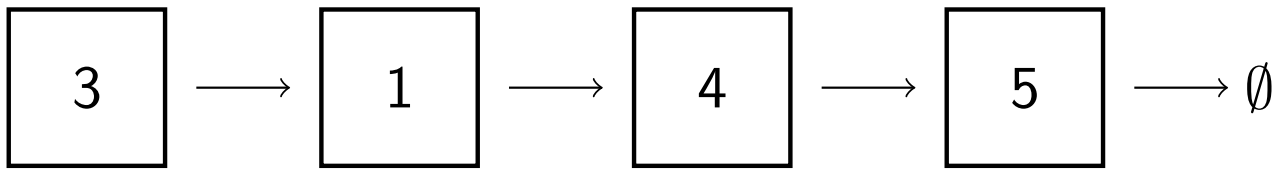
Restaurant Reservation Problem

- do not know how many values to store (e.g. logging events)
- concatenation: add elements to array
- not specifying space needed can be very inefficient

Arrays: why aren't they good enough?

- not dynamic (can use up too much/little space)
- unstructured search
- doesn't always make use of all contextual information (hierarchical, spatial, temporal, priority, etc.)

Linked Lists



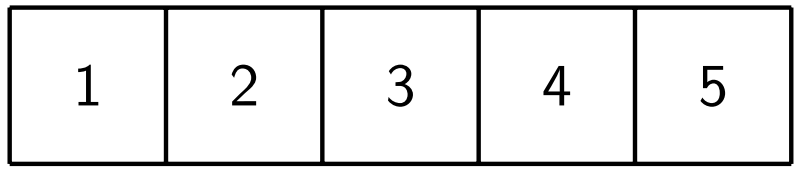
Each node contains:

1. data
2. next address

Overview:

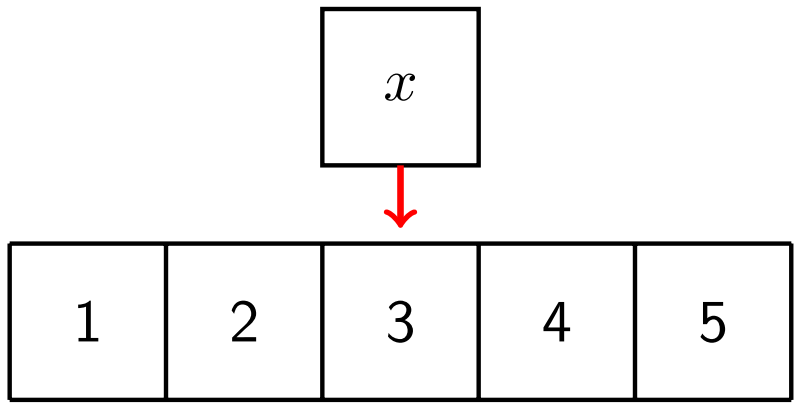
- does not rely on contiguous memory spaces
- building blocks of many data structures

Linked Lists: Insertion and Deletion



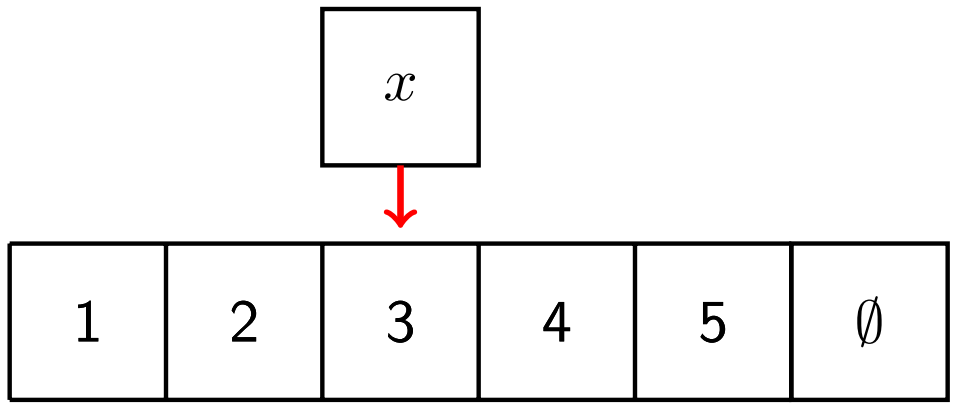
Insert value into middle of array

Linked Lists: Insertion and Deletion



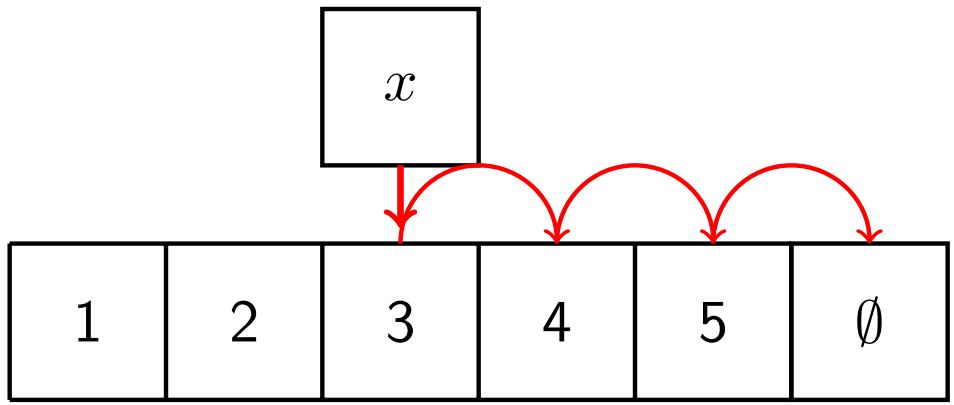
Cannot insert without overwriting

Linked Lists: Insertion and Deletion



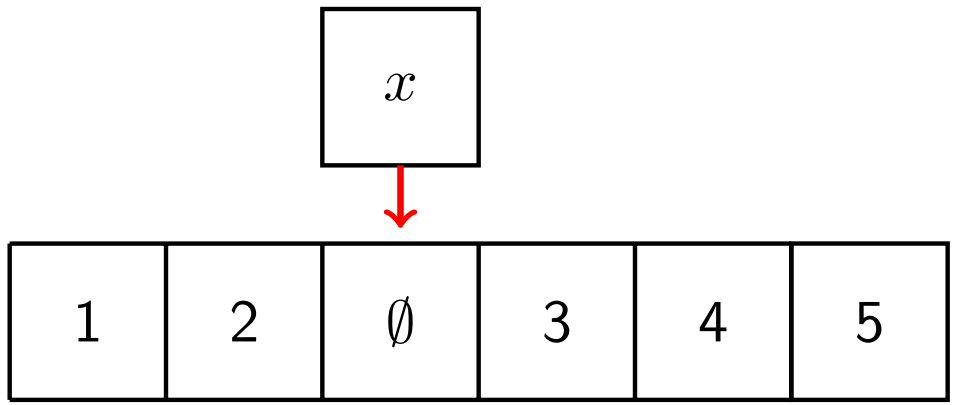
Must resize array

Linked Lists: Insertion and Deletion



Shuffle array

Linked Lists: Insertion and Deletion

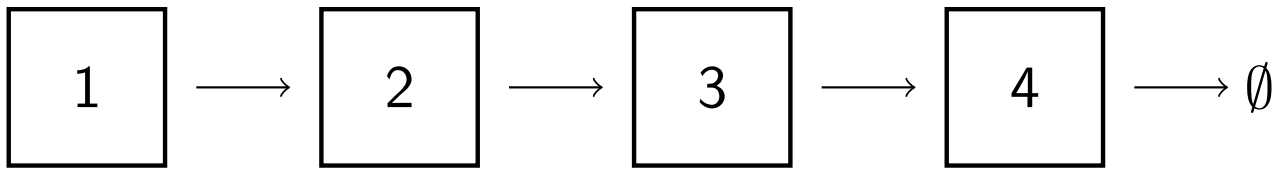


Shuffle array

Linked Lists: Insertion and Deletion

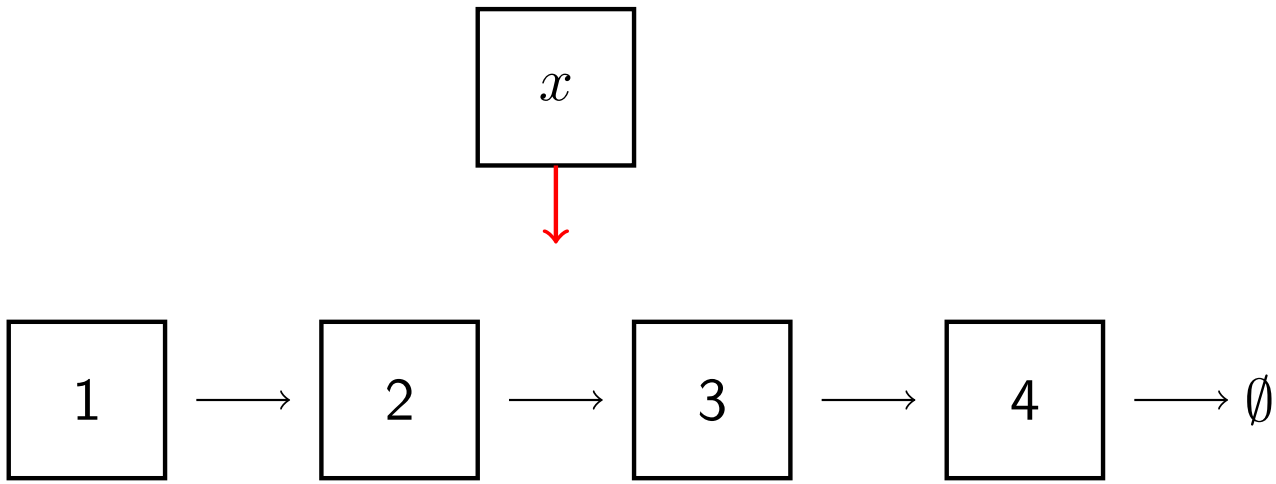
Insert value

Linked Lists: Insertion and Deletion



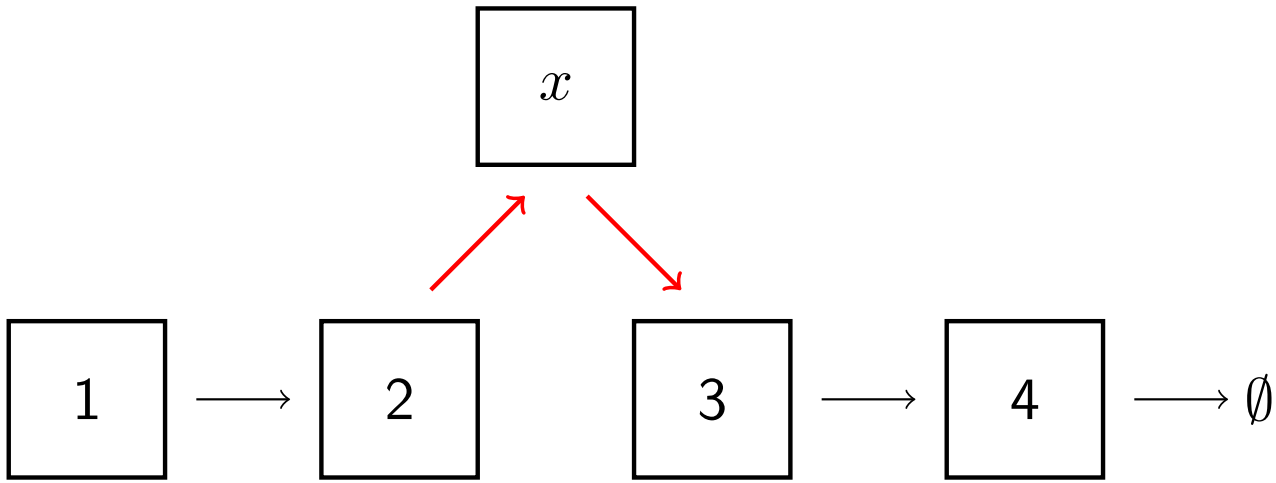
Insert value into middle of linked list (after 2 and before 3)

Linked Lists: Insertion and Deletion



Insert value into middle of linked list (after 2 and before 3)

Linked Lists: Insertion and Deletion



Change next address

Linked Lists: Variations

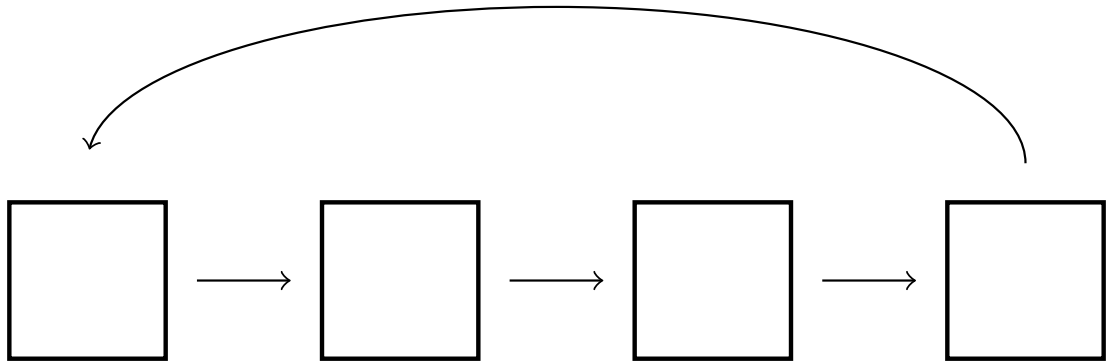


Figure: Circular Linked Lists

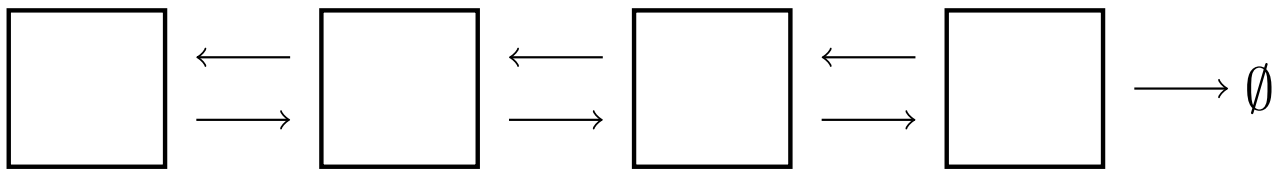


Figure: Doubly Linked Lists

Linked Lists: Variations

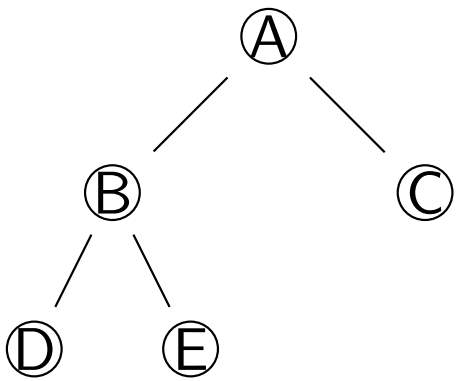


Figure: Trees

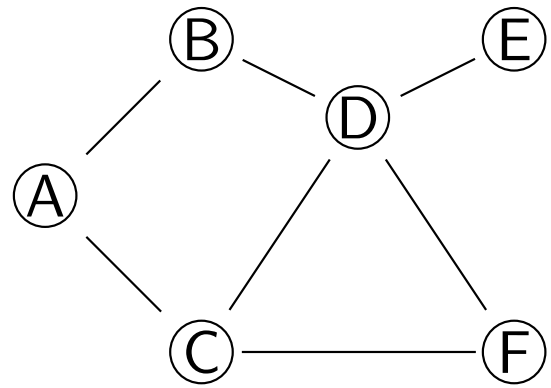


Figure: Graphs

Linked Lists: Overview

Better than arrays because...

- dynamic size
- faster insertion and deletion

Problems:

- no random access: must traverse through each node sequentially
- cannot move backwards

Queue

Overview:

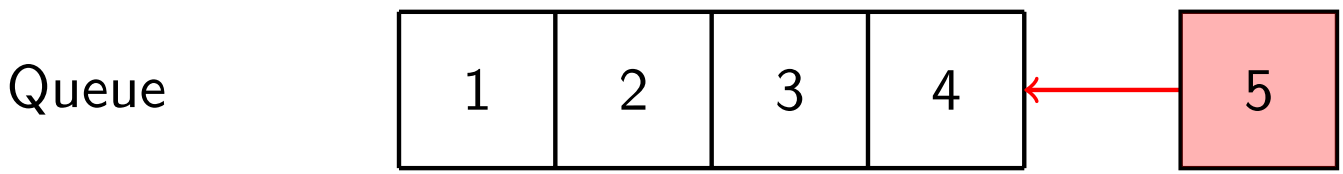
- exploits temporal information
- first in, first out (FIFO)
- dynamic array (size changes)

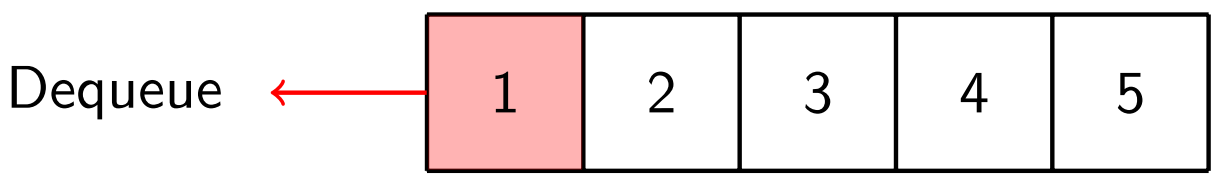
Operations:

- queue: add to rear
- dequeue: remove from front

Applications:

- task scheduling
- resource allocation





Stack

Overview:

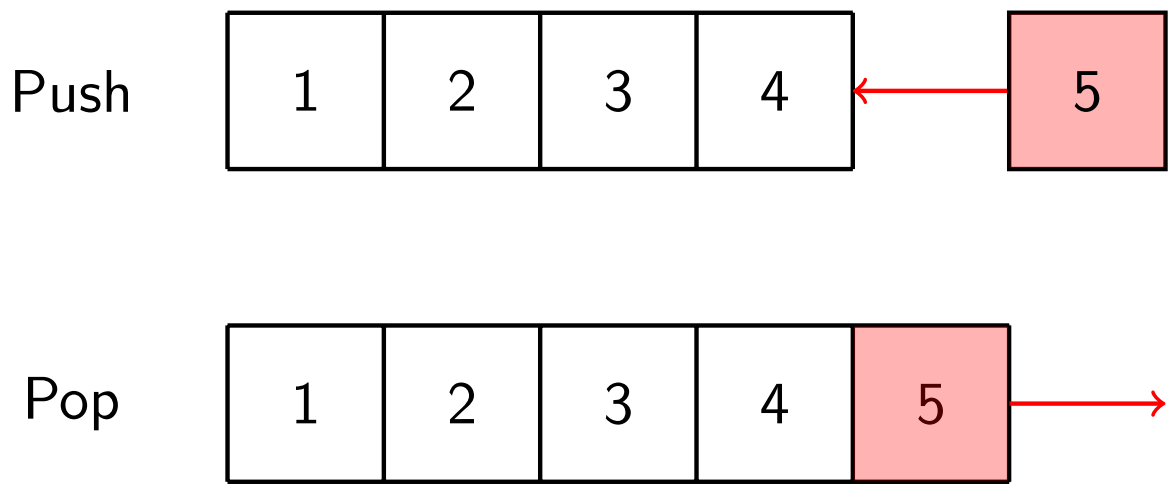
- exploits temporal information
- last in, first out (LIFO)
- dynamic array (size changes)

Operations:

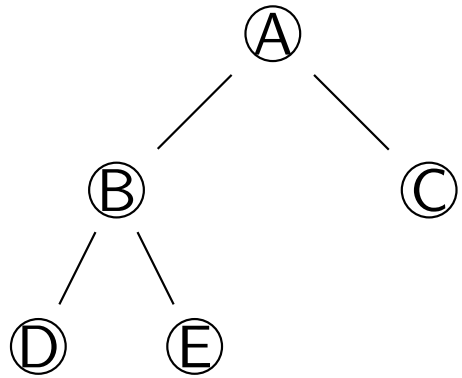
- push: add to stack
- pop: remove from stack

Applications:

- used for "undo" operations



Trees



Overview:

- exploits hierarchical structure

Applications:

- file systems
- expression trees
- game trees

Tree Applications

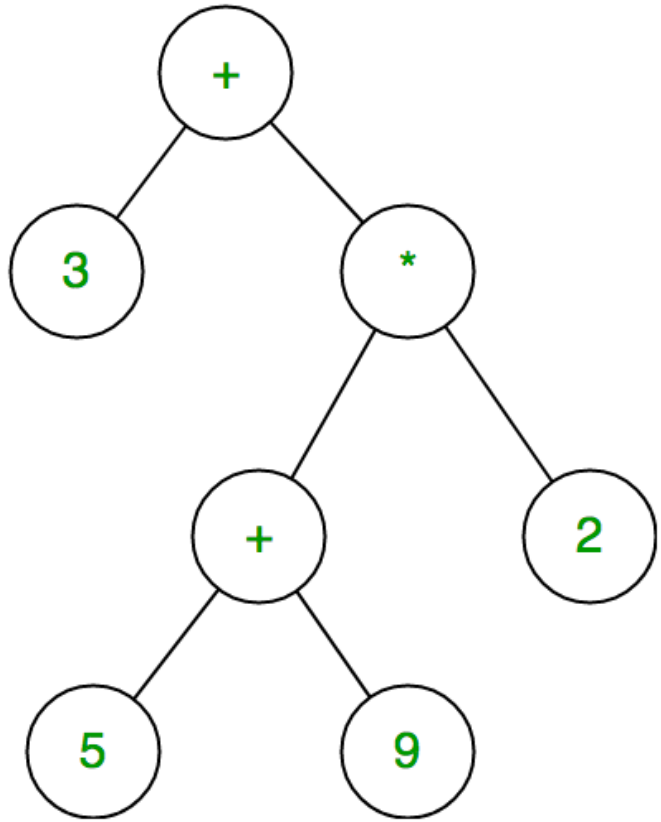


Figure: Expression Tree of $3 + ((5 + 9) * 2)$:
[https://www.geeksforgeeks.org/expression-tree/]

Tree Applications (2)

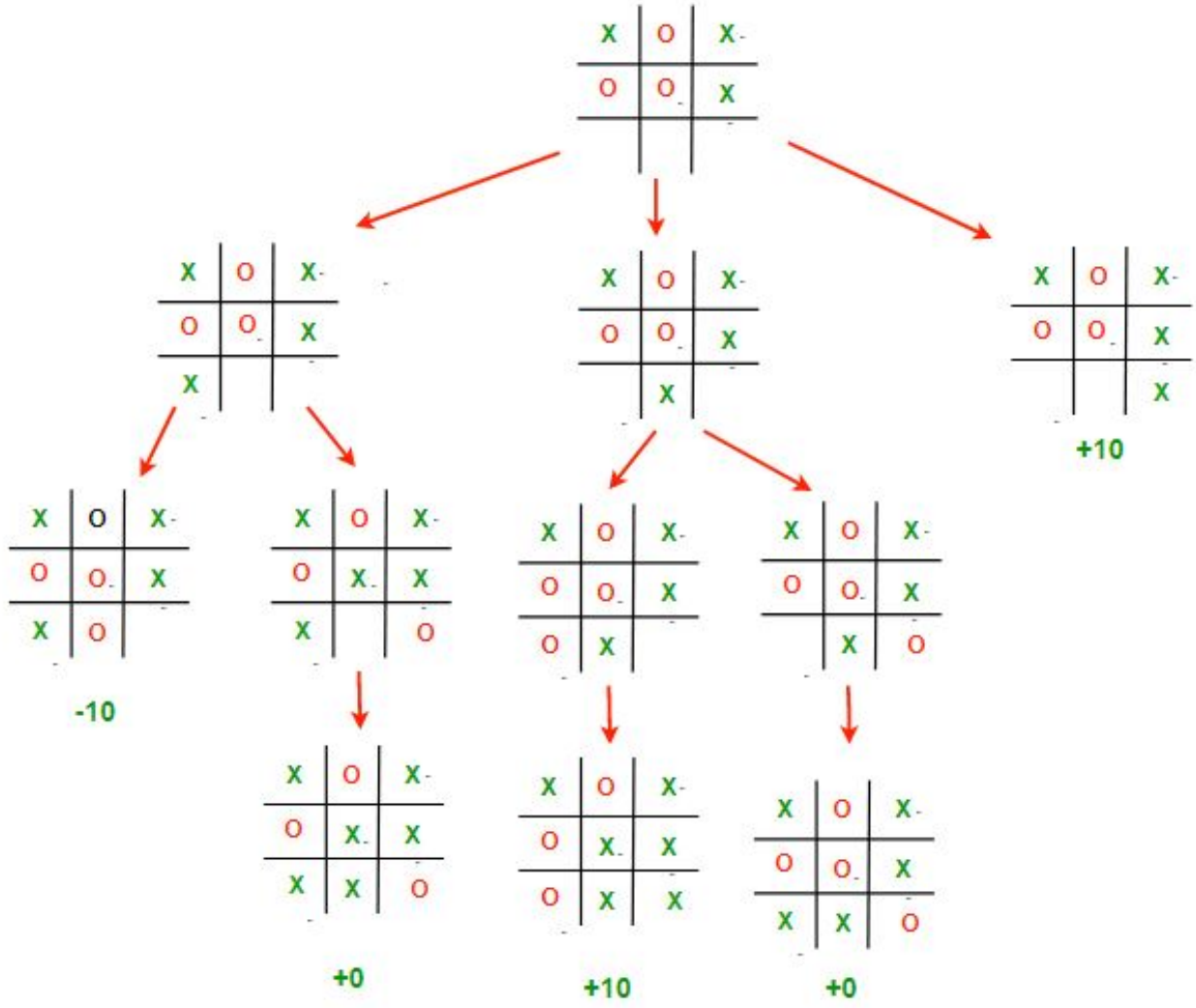
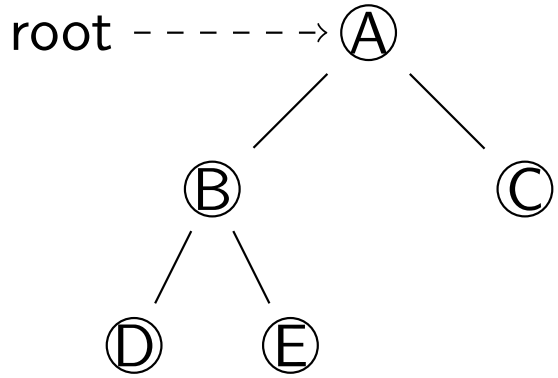


Figure: Monte Carlo Search Tree
[https://www.analyticsvidhya.com/blog/2019/01/monte-carlo-tree-search-introduction-algorithm-deepmind-alphago/]

Anatomy of a Tree



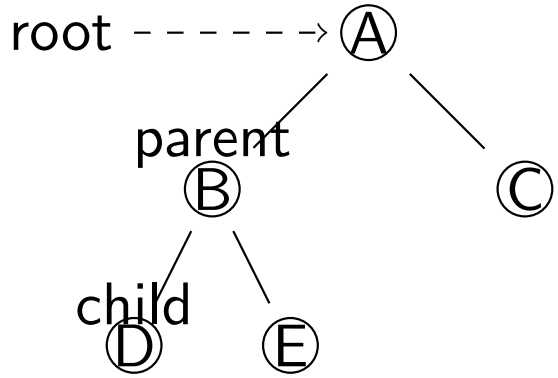
Parts:

- **root:** topmost node
- **parent:** node with child nodes
- **child:** node connected to parent
- **leaf:** node with no children
- **subtree:** all descendants from a node in a tree

Properties

- **complete:** every level but the last is entirely filled
- **balanced:** left and right subtrees do not differ than more than one

Anatomy of a Tree



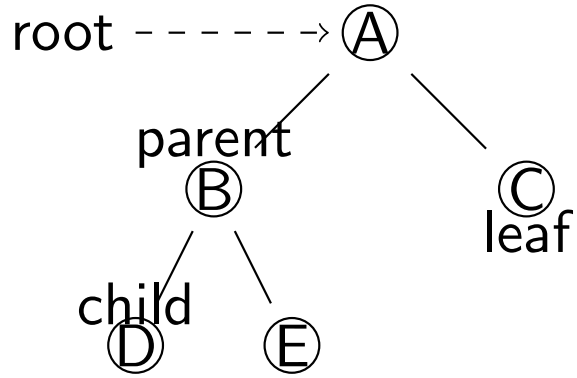
Parts:

- **root:** topmost node
- **parent:** node with child nodes
- **child:** node connected to parent
- **leaf:** node with no children
- **subtree:** all descendants from a node in a tree

Properties

- **complete:** every level but the last is entirely filled
- **balanced:** left and right subtrees do not differ than more than one

Anatomy of a Tree



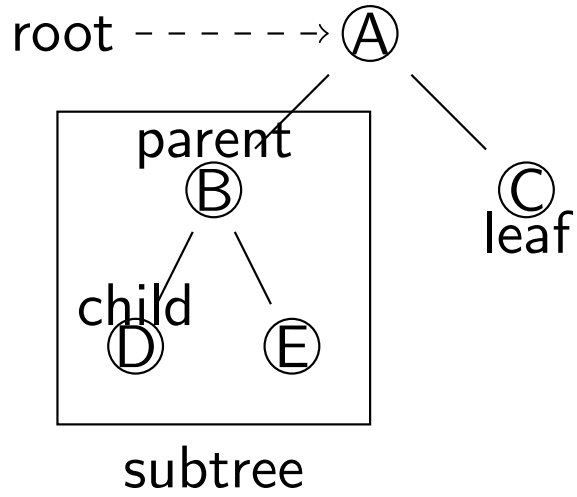
Parts:

- **root:** topmost node
- **parent:** node with child nodes
- **child:** node connected to parent
- **leaf:** node with no children
- **subtree:** all descendants from a node in a tree

Properties

- **complete:** every level but the last is entirely filled
- **balanced:** left and right subtrees do not differ than more than one

Anatomy of a Tree



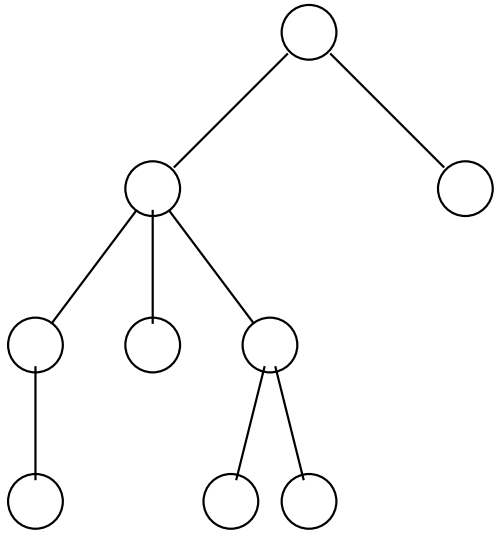
Parts:

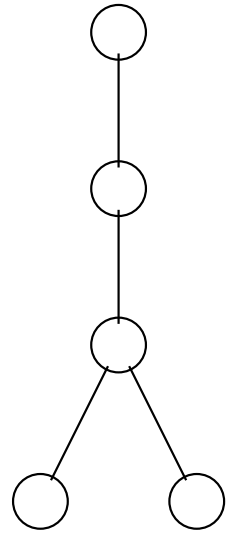
- **root:** topmost node
- **parent:** node with child nodes
- **child:** node connected to parent
- **leaf:** node with no children
- **subtree:** all descendants from a node in a tree

Properties

- **complete:** every level but the last is entirely filled
- **balanced:** left and right subtrees do not differ than more than one

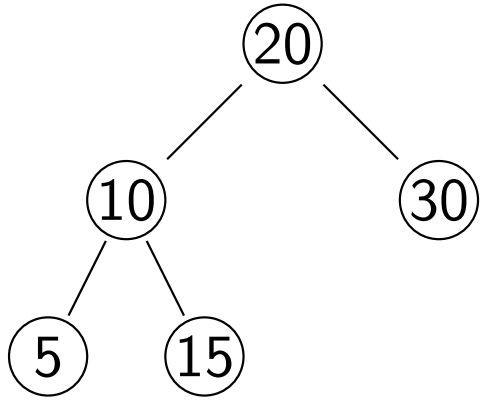
Trees





- trees can be fairly unstructured
  - no restriction on number of children
  - no ordering structure
- we will apply structure and go over some special properties

Binary Search Tree



Ordering Property

- each node has at most two children
- left children have values less than parent
- right children have values greater than parent

Overview:

- structure allows for efficient searching ($O(\log n)$)
- in-order traversal

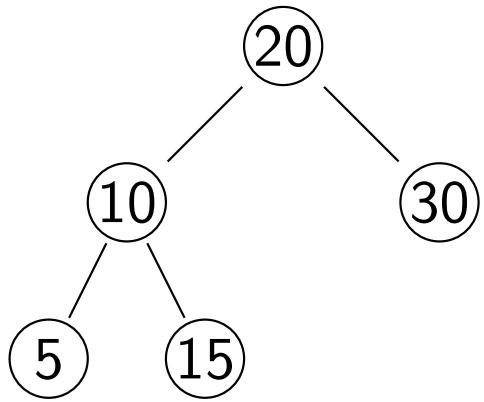
Heaps

Overview:

- min/max is always found at root
- complete binary tree
- last level filled left to right

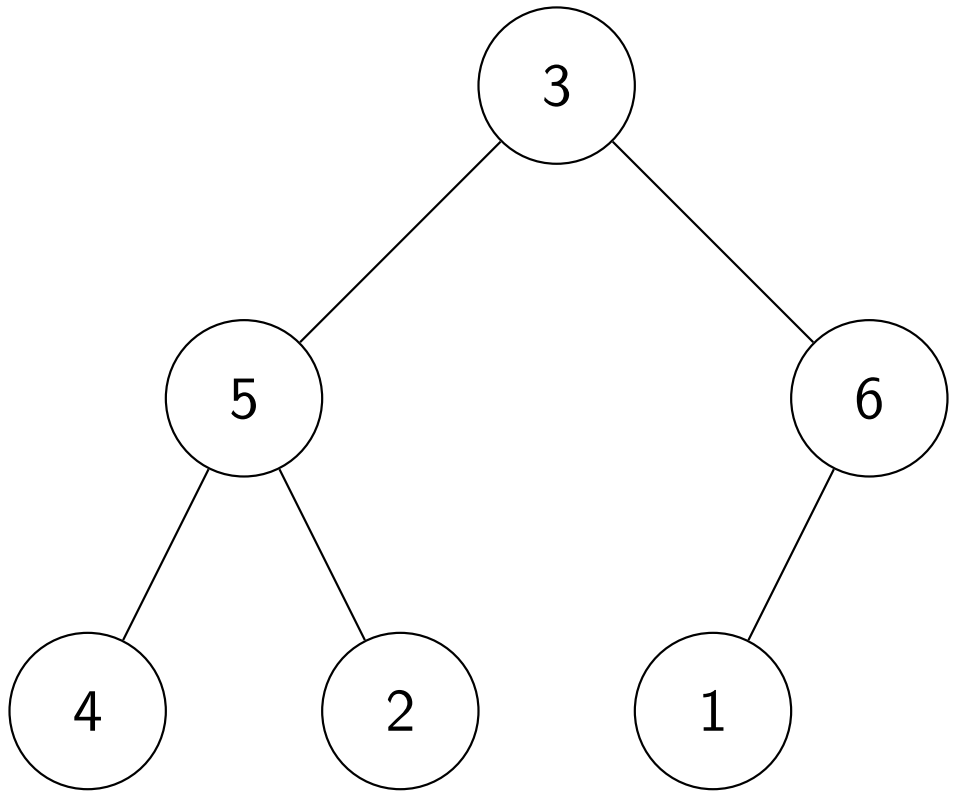
Applications:

- priority queue: parallel computing
- shortest path search



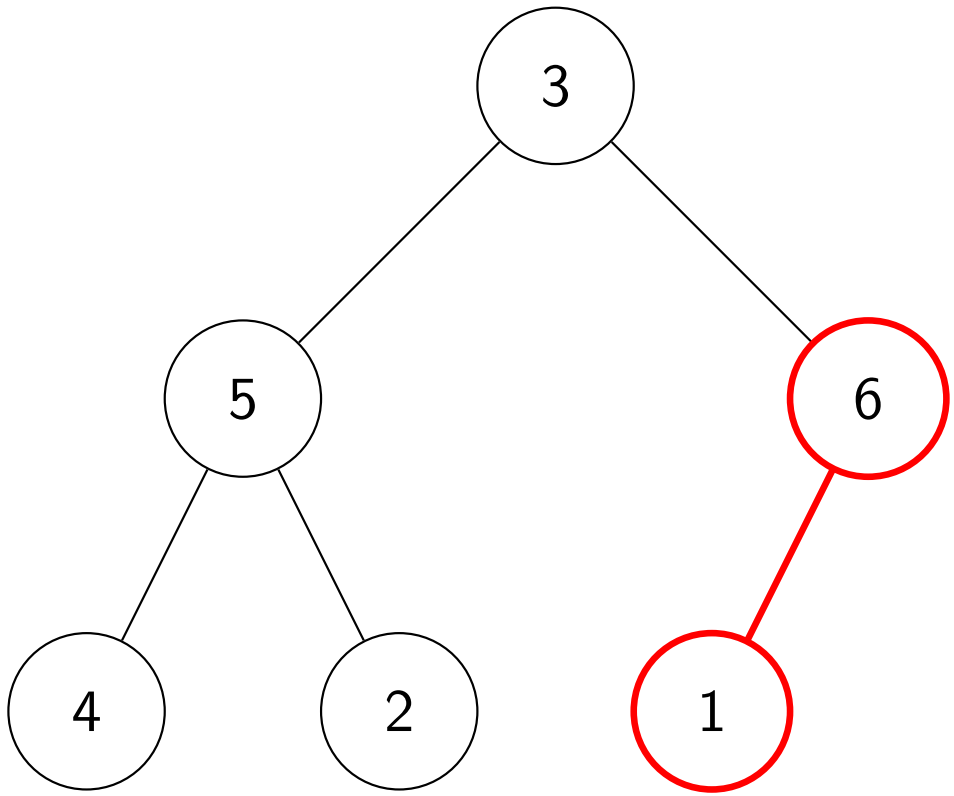
Heapify (Min Heap)

x = [3, 5, 6, 4, 2, 1]
Build complete binary tree



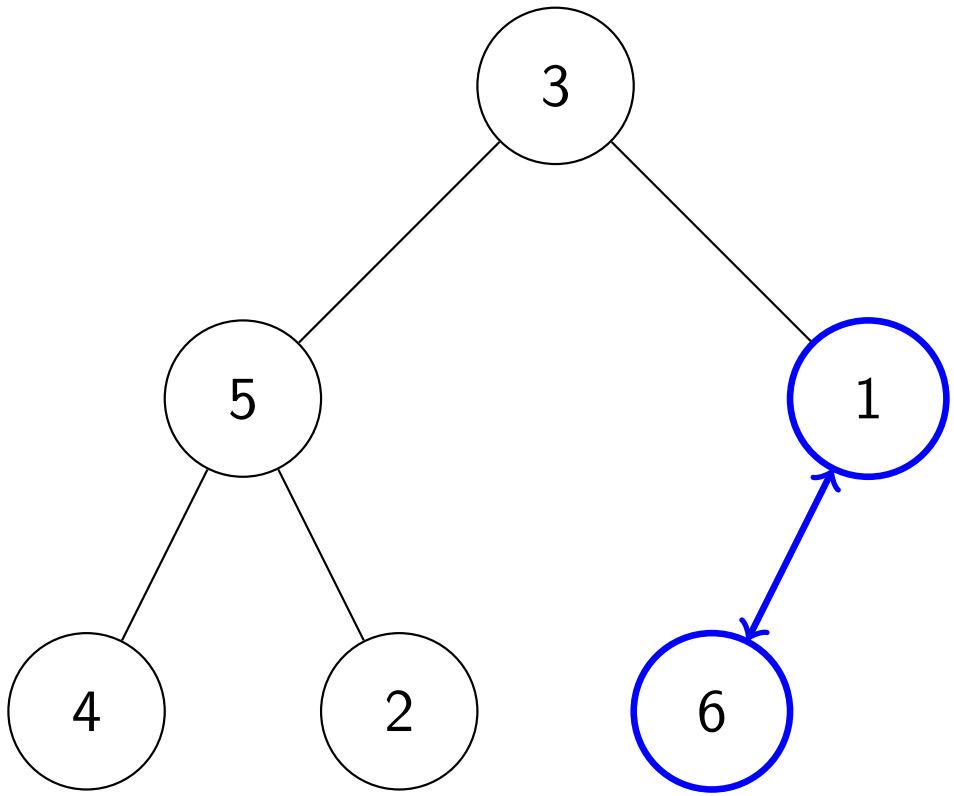
Heapify (Min Heap)

Start at last parent



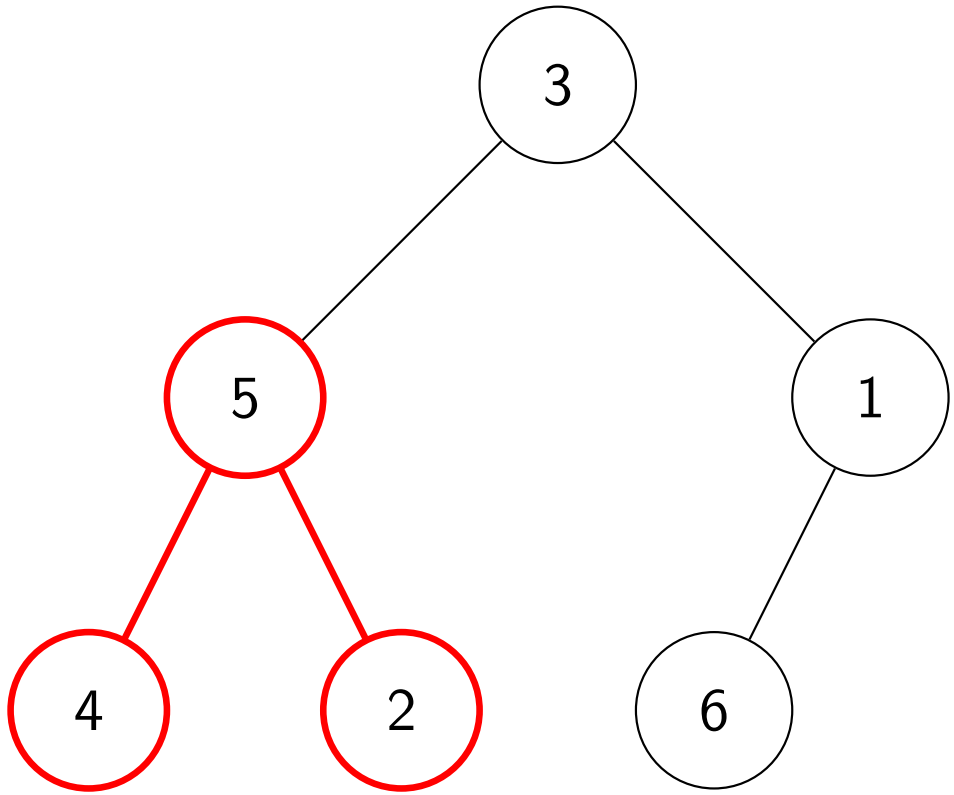
Heapify (Min Heap)

Swap



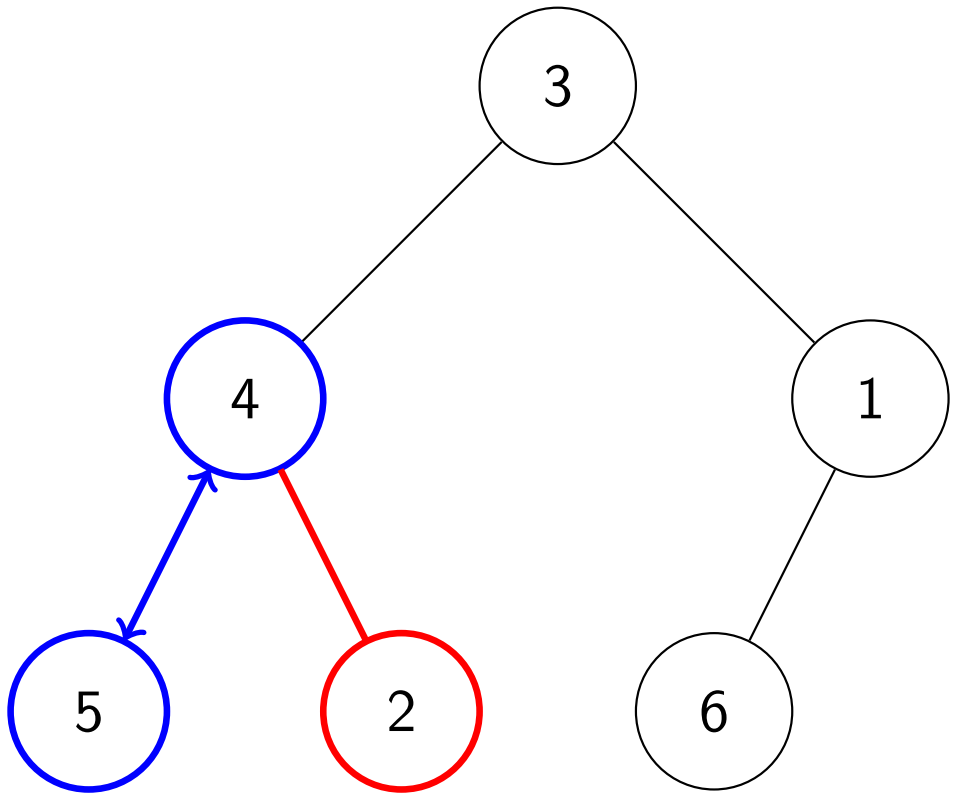
Heapify (Min Heap)

Move to next subtree



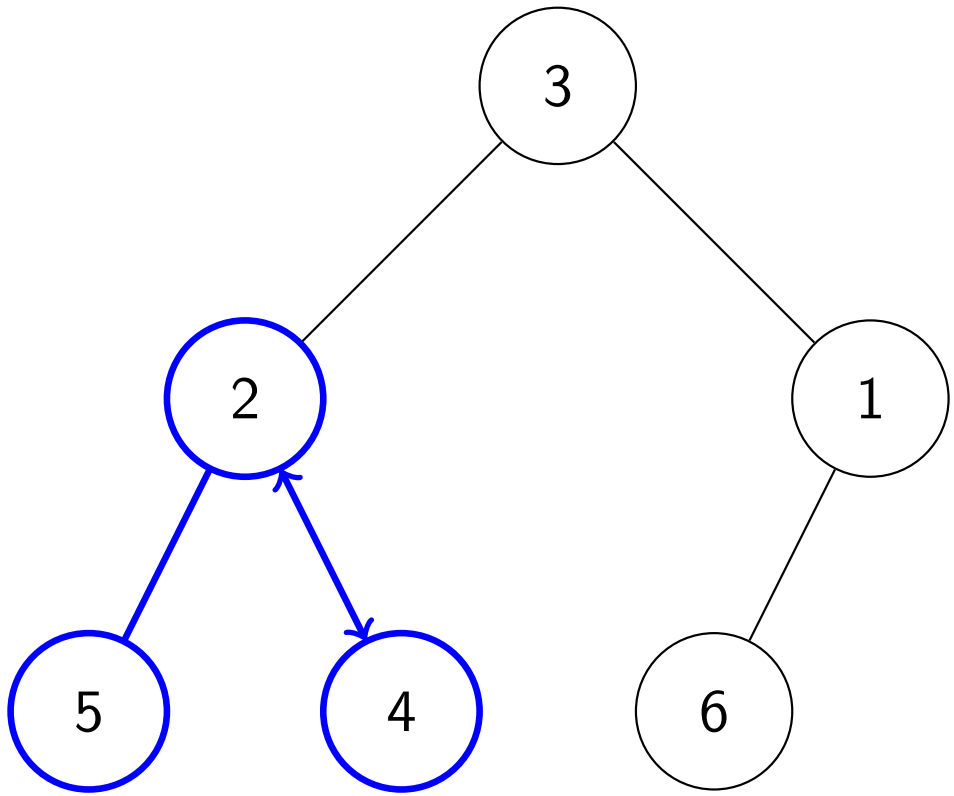
Heapify (Min Heap)

Swap left child



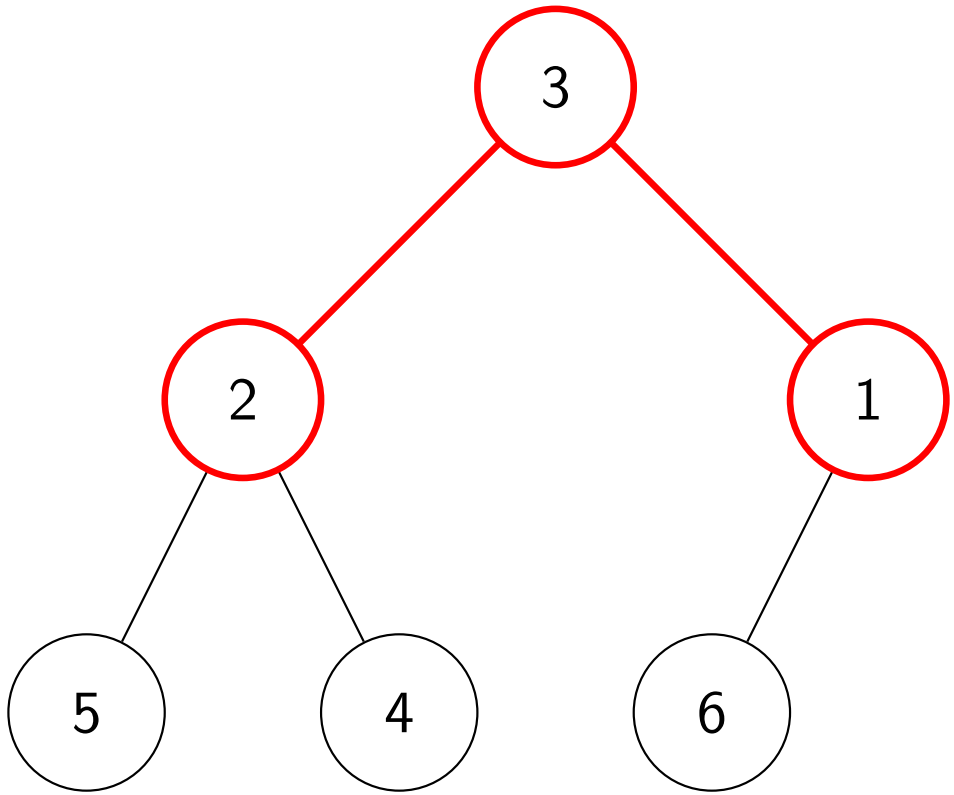
Heapify (Min Heap)

Swap right child



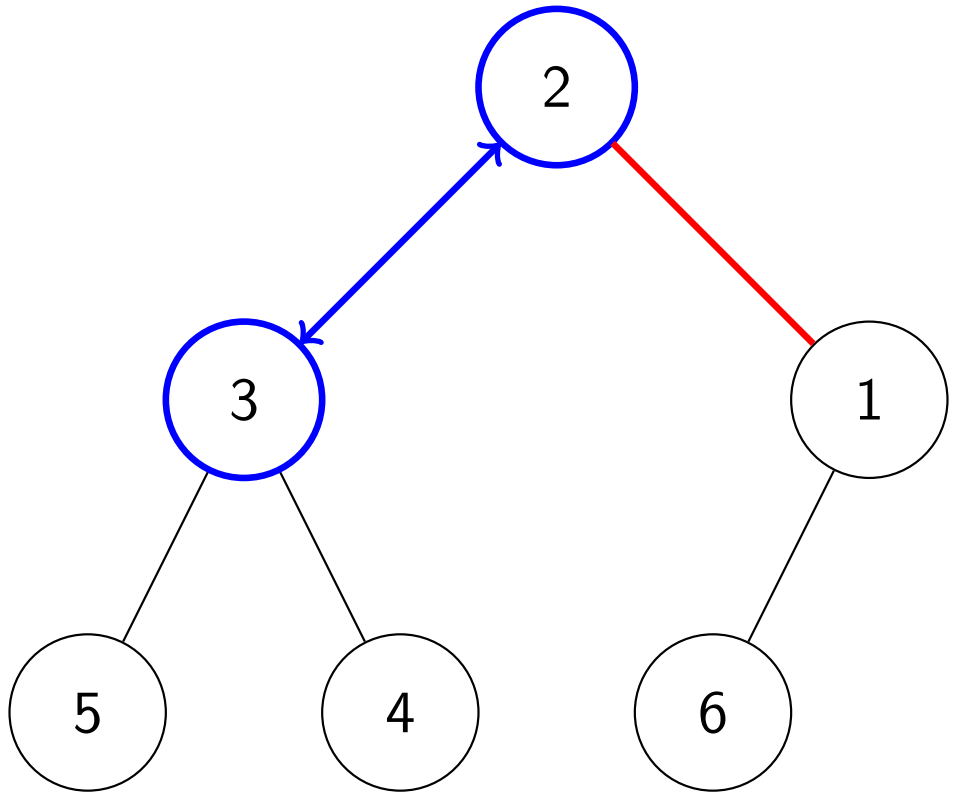
Heapify (Min Heap)

Move to next subtree



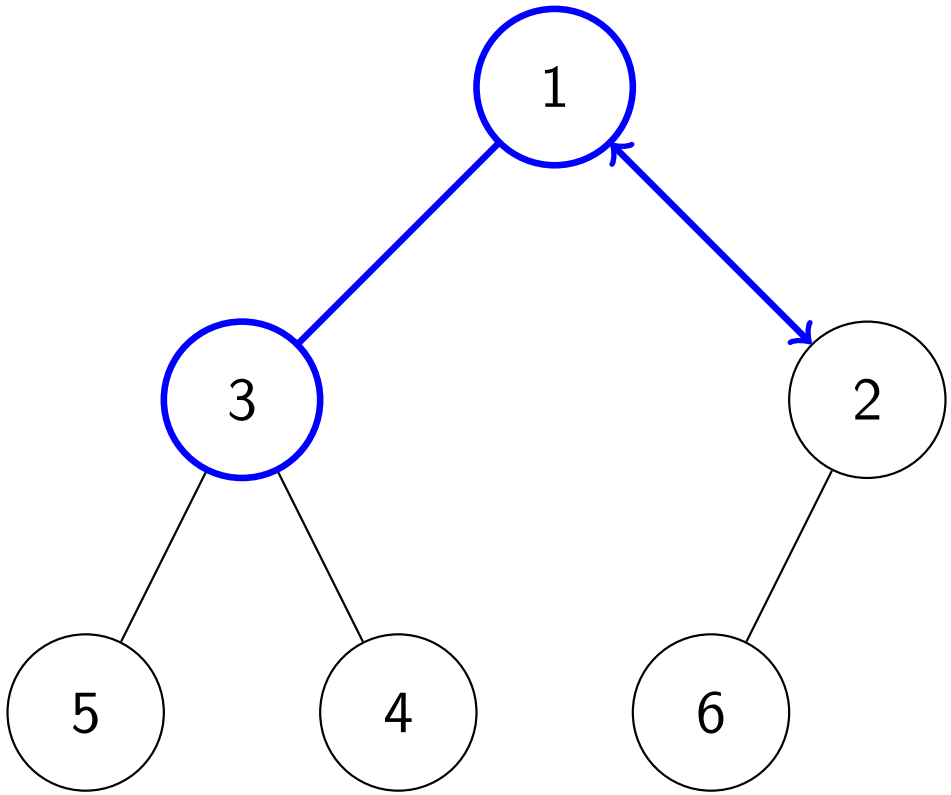
Heapify (Min Heap)

Check left child

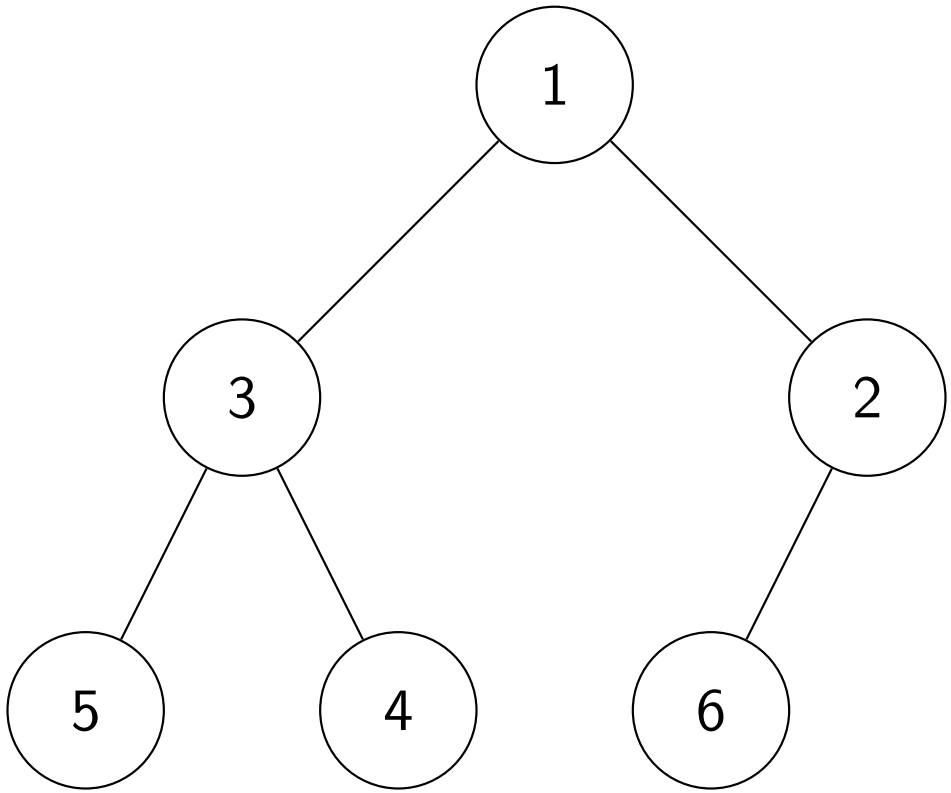


Heapify (Min Heap)

Swap right child

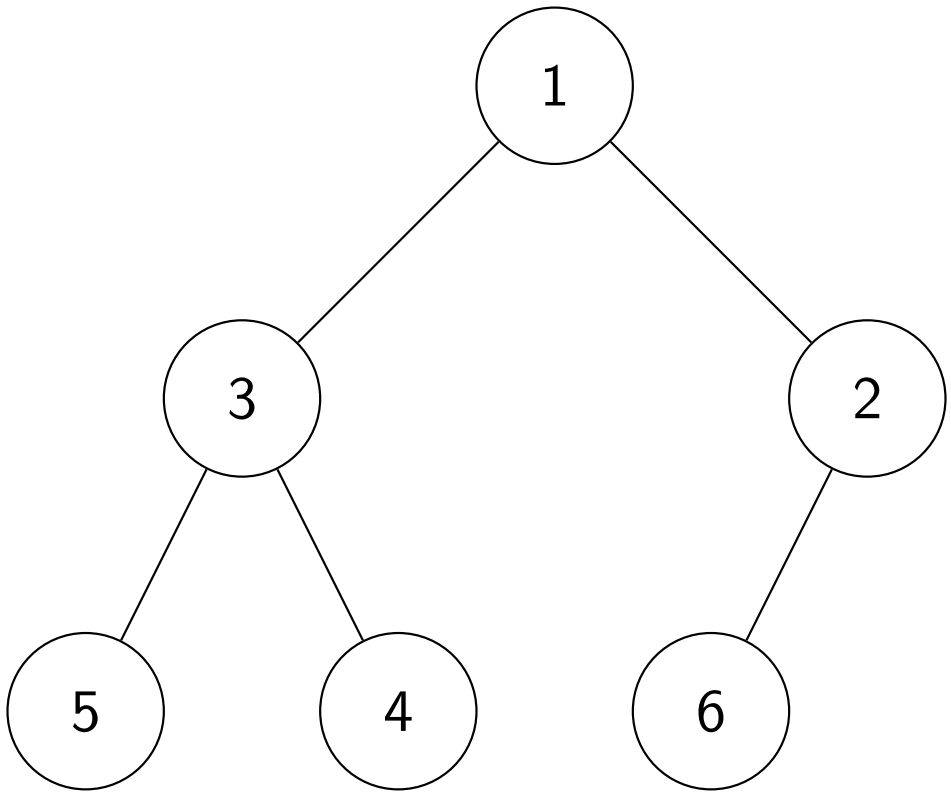


Heapify (Min Heap)



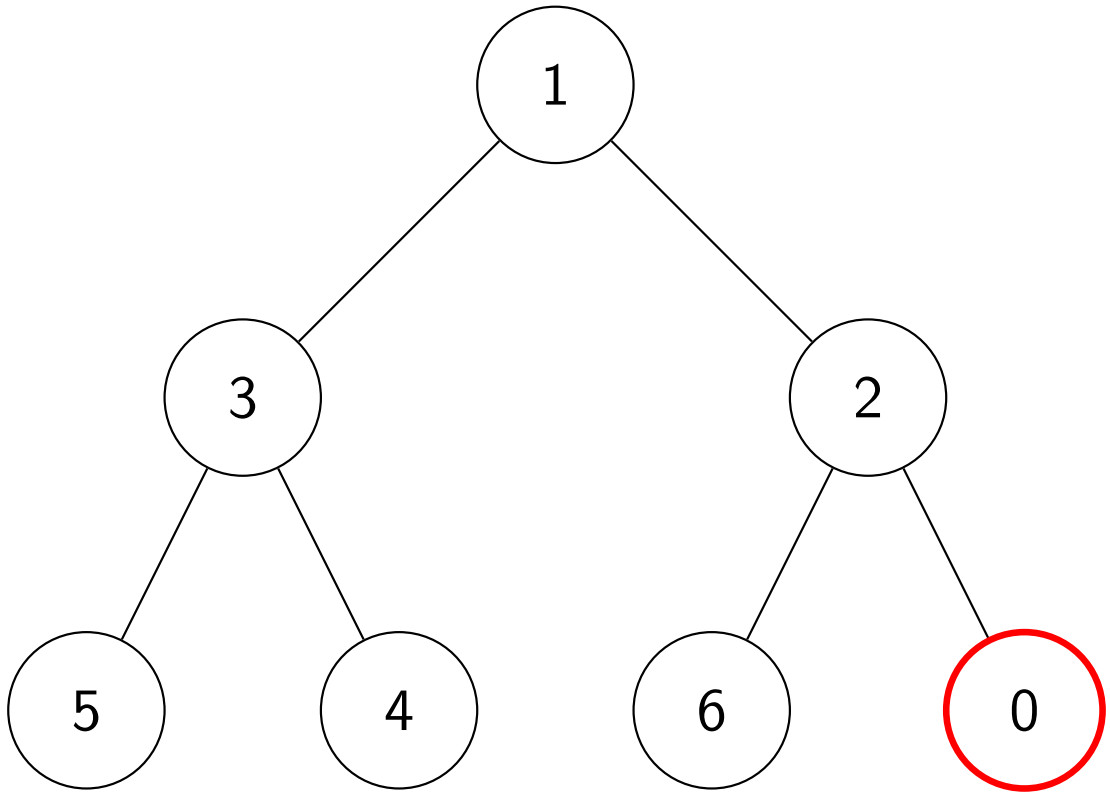
Heap: Insert

insert 0 to heap



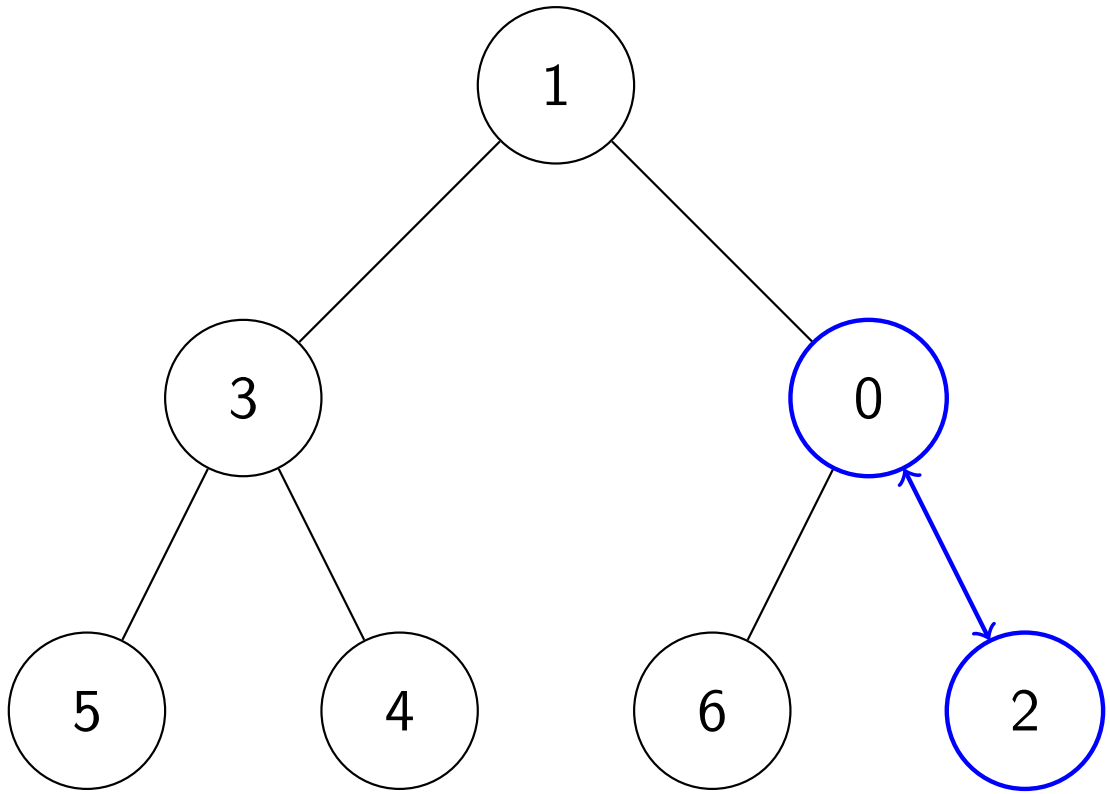
Heap: Insert

insert 0 to heap



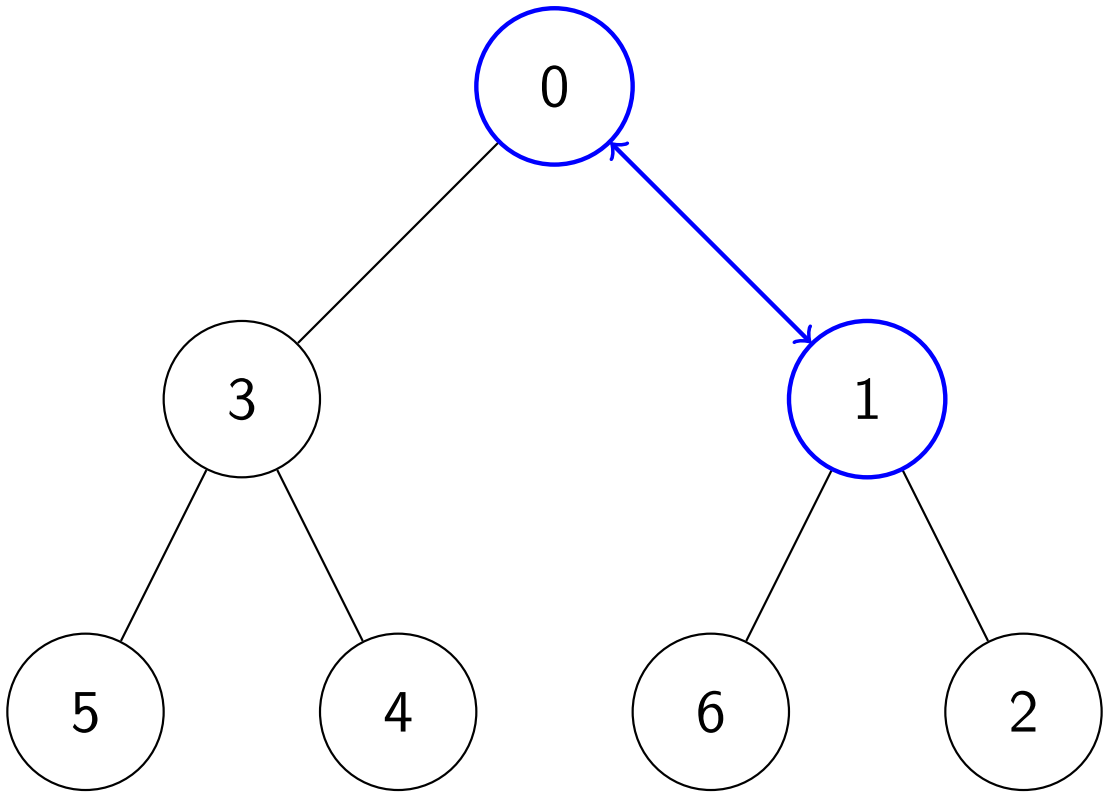
Heap: Insert

heapify



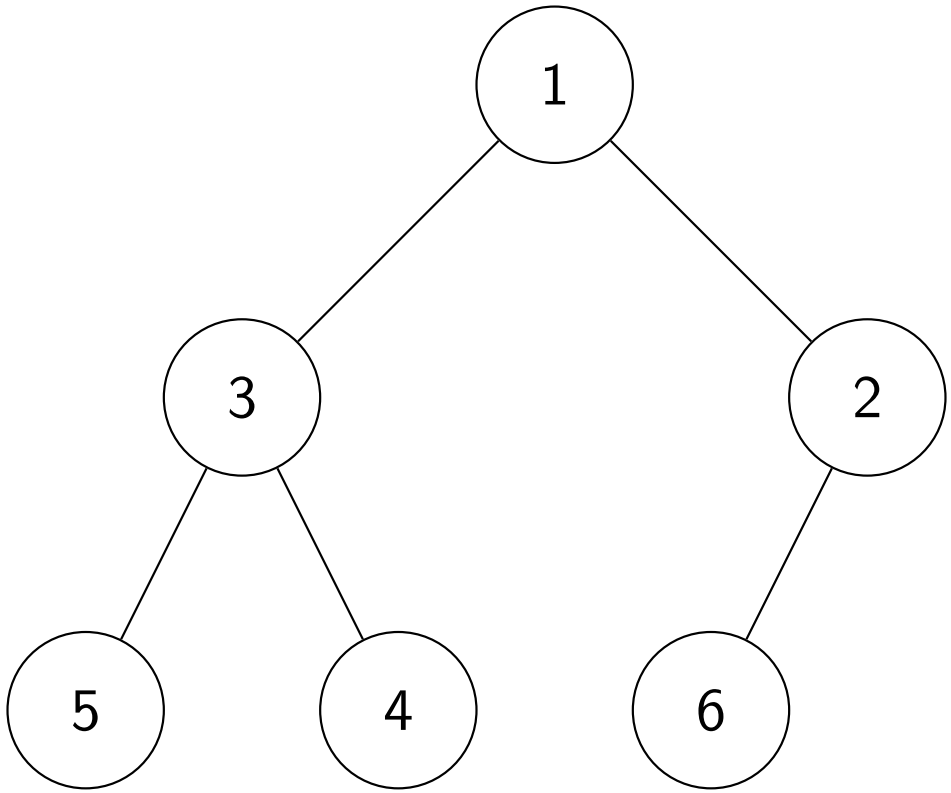
Heap: Insert

heapify



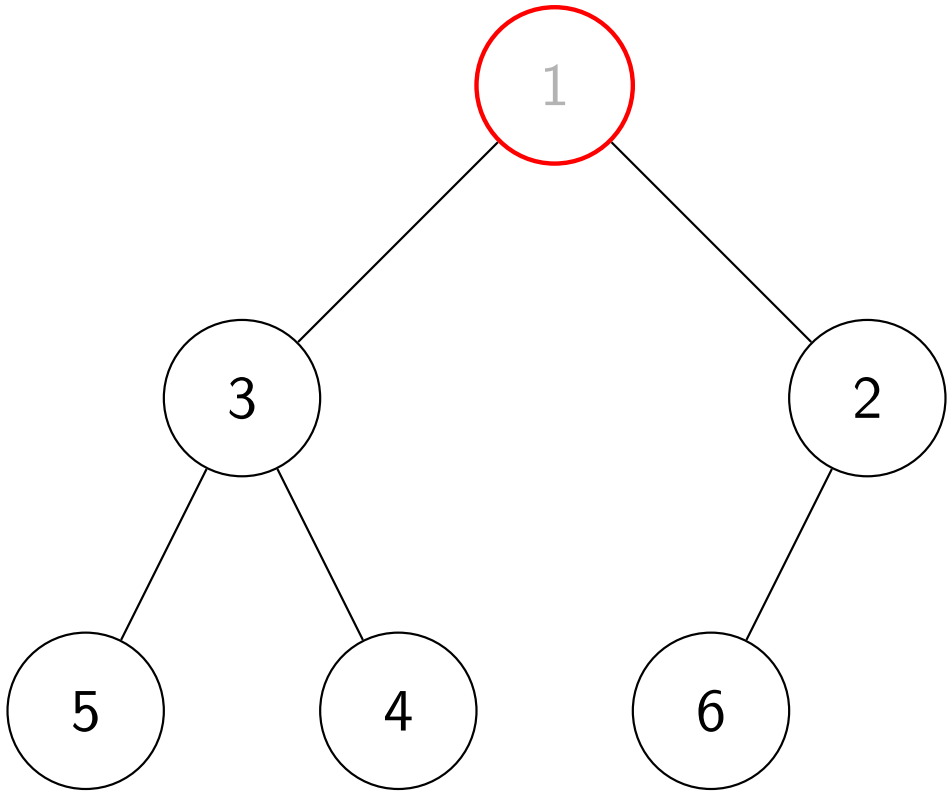
Heap: Delete

Delete root



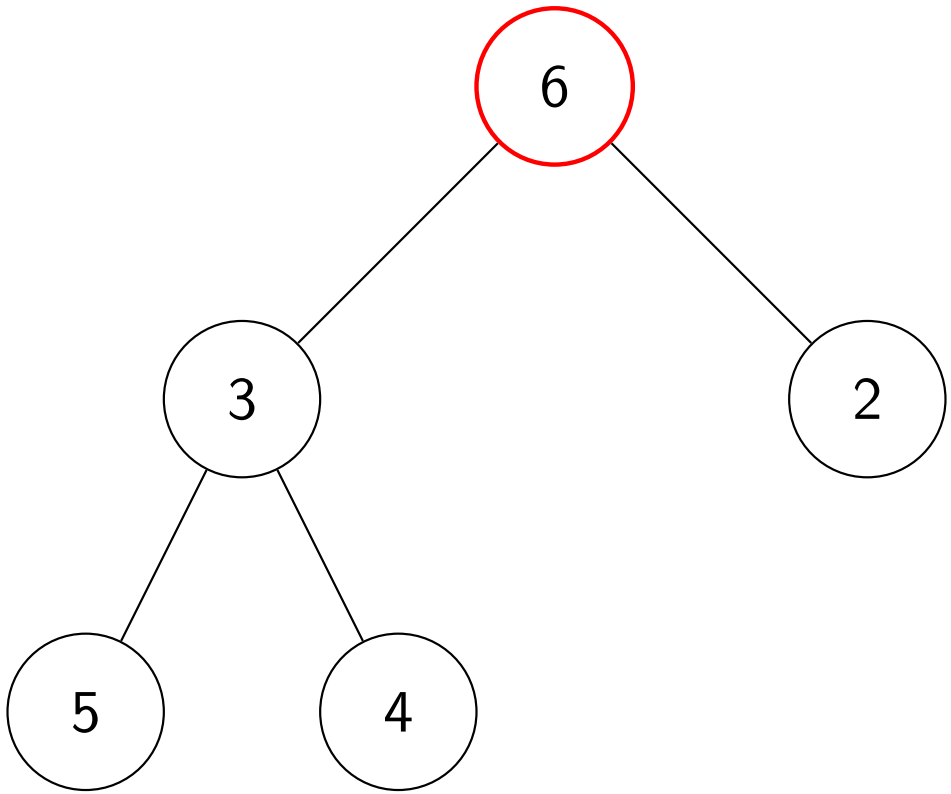
Heap: Delete

Delete root



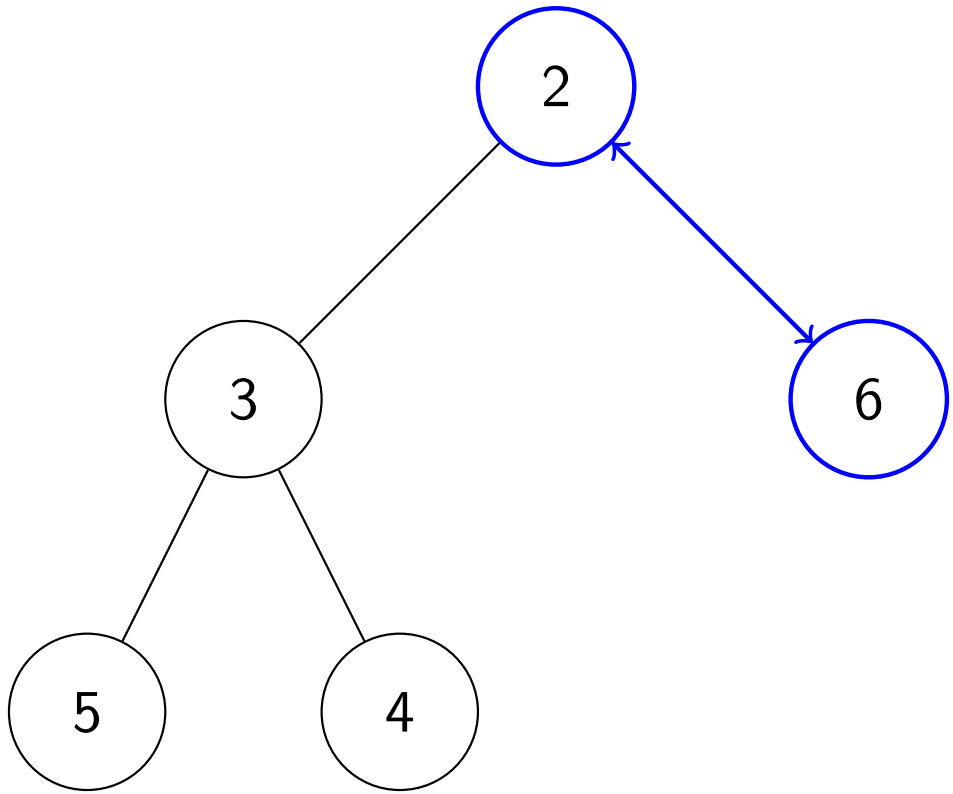
Heap: Delete

Replace with last node



Heap: Delete

Heapify smaller child



Heaps: Overview

- Build Heap: $O(n)$
- Insertion: $O(\log n)$
- Deletion: $O(\log n)$
- Fast to find min value
- Fast to reorganize heap after insertion/deletion

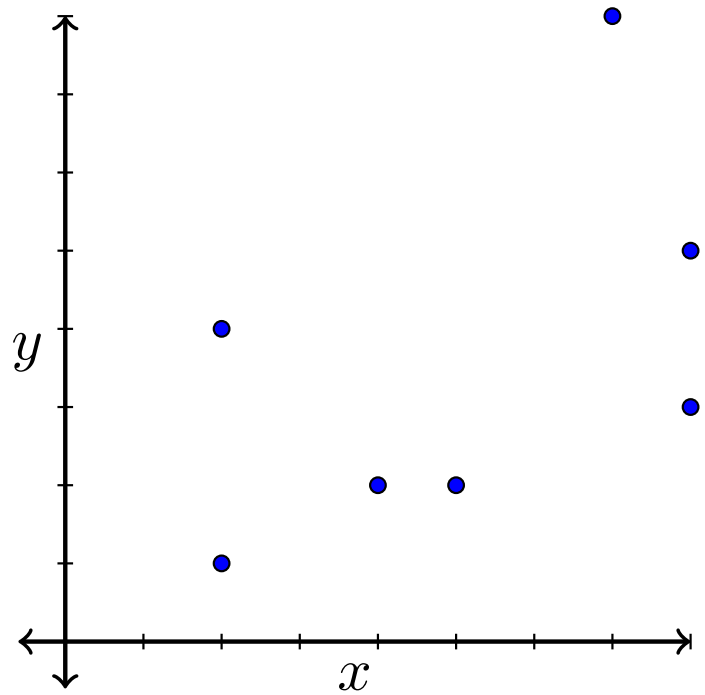
K-Dimensional Trees

Overview

- encodes spatial information
- efficient search for data points
- used for k-nearest neighbors

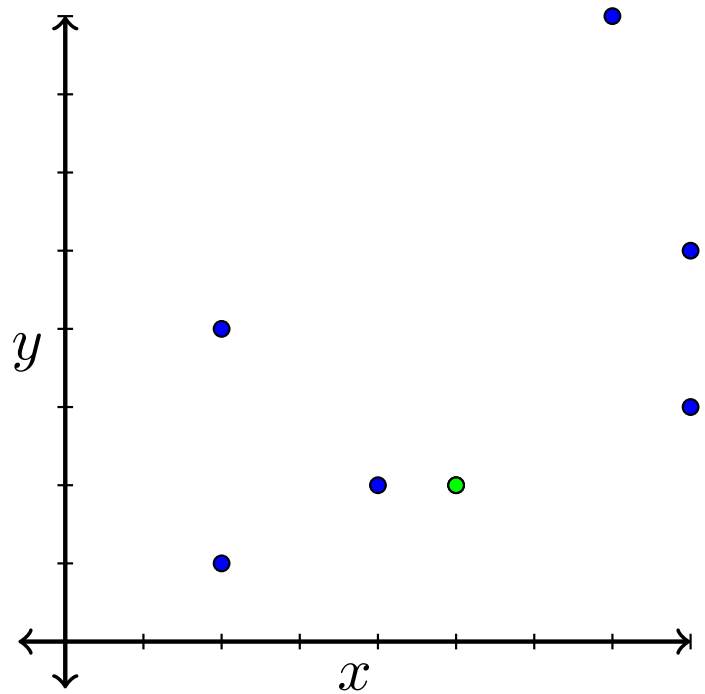
K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)



K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

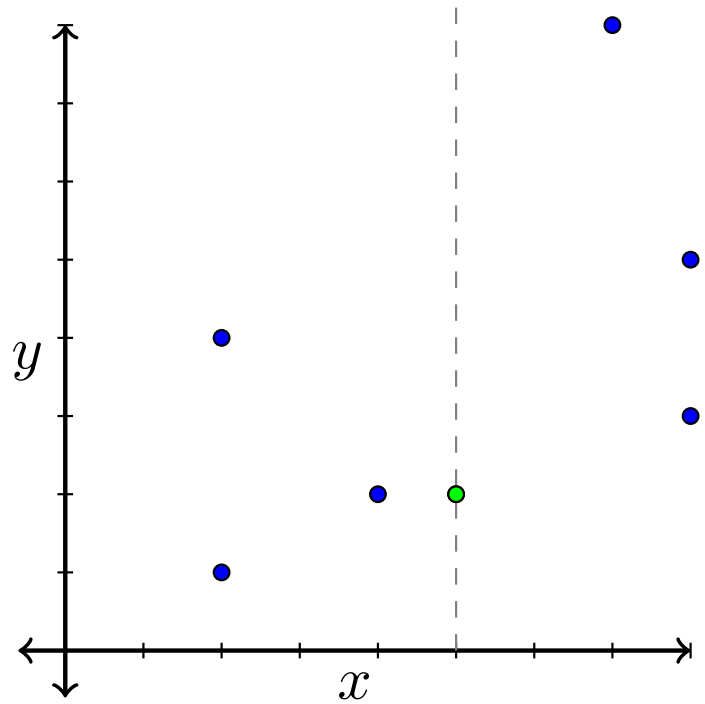


K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

make root

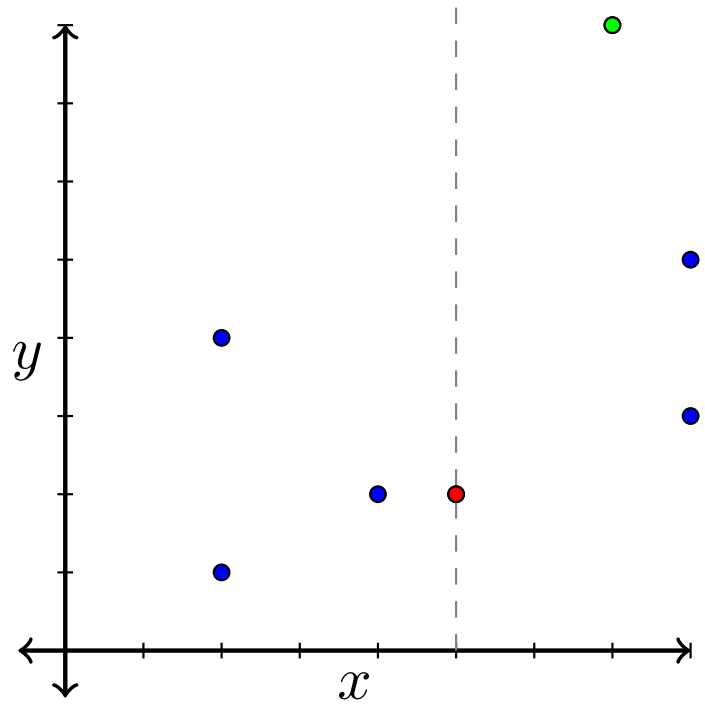
x ..... $(5, 2)$ .....



K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

x ..... $(5, 2)$ .....

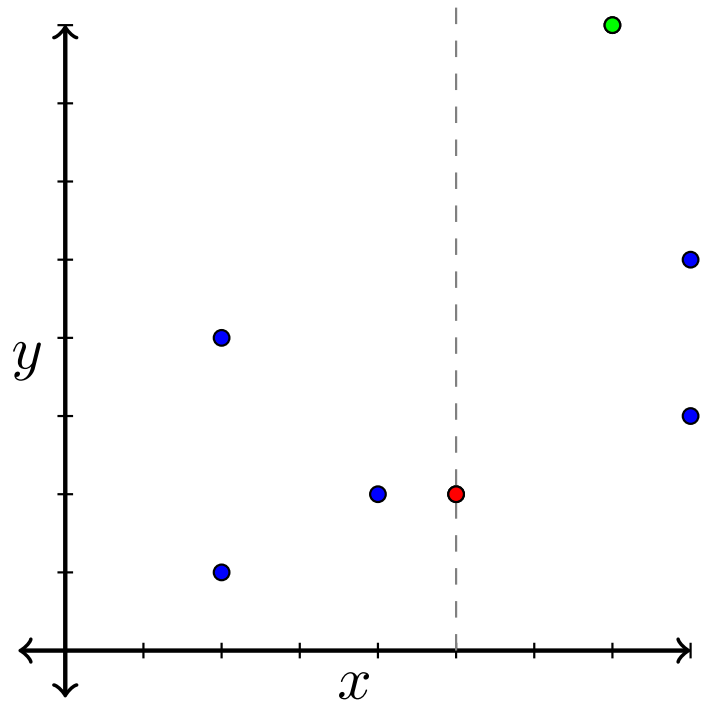


K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

$$(5, 2) \leq (7, 8)$$

x ..... $(5, 2)$ .....



K-Dimensional Trees

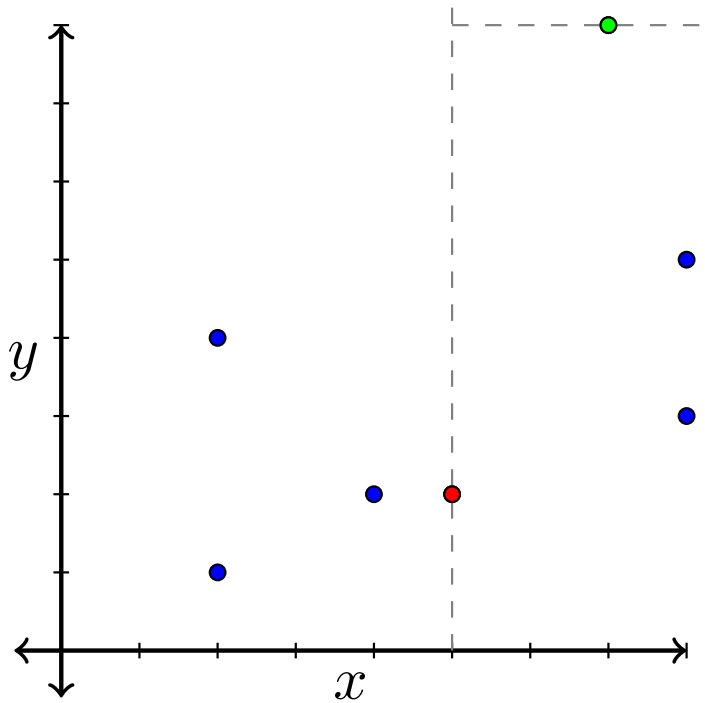
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

$$(5, 2) \leq (7, 8)$$

x ..... $(5, 2)$ .....

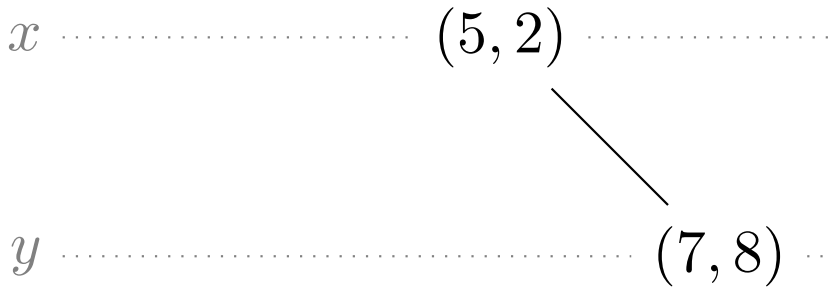
make child

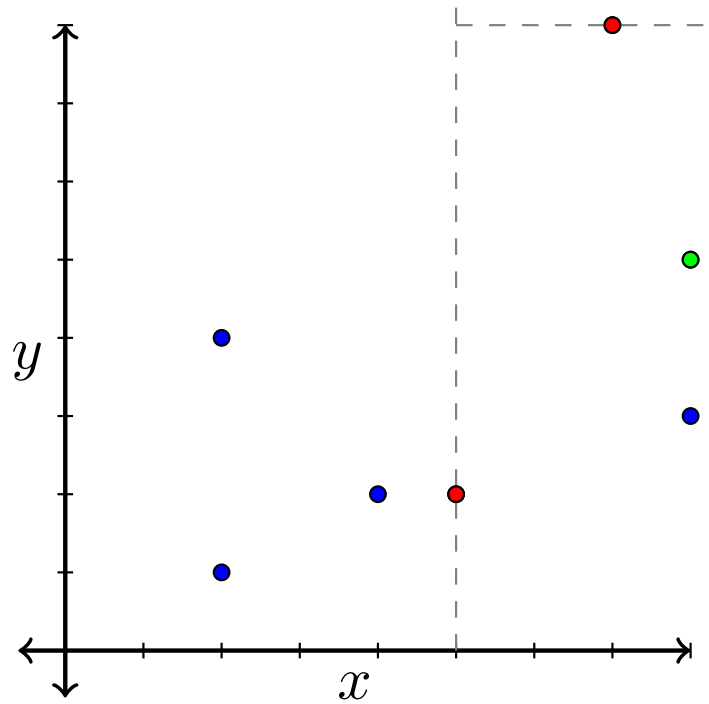
y ..... $(7, 8)$ ..



K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

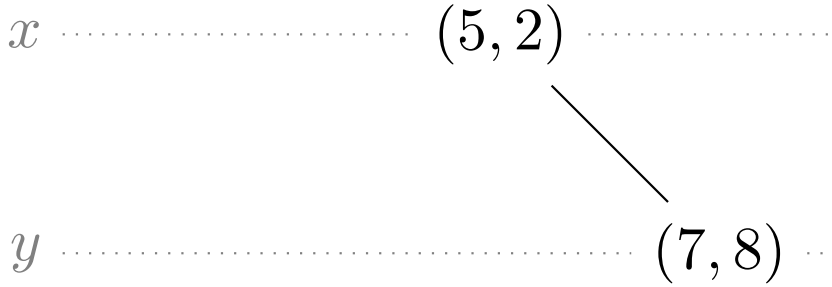


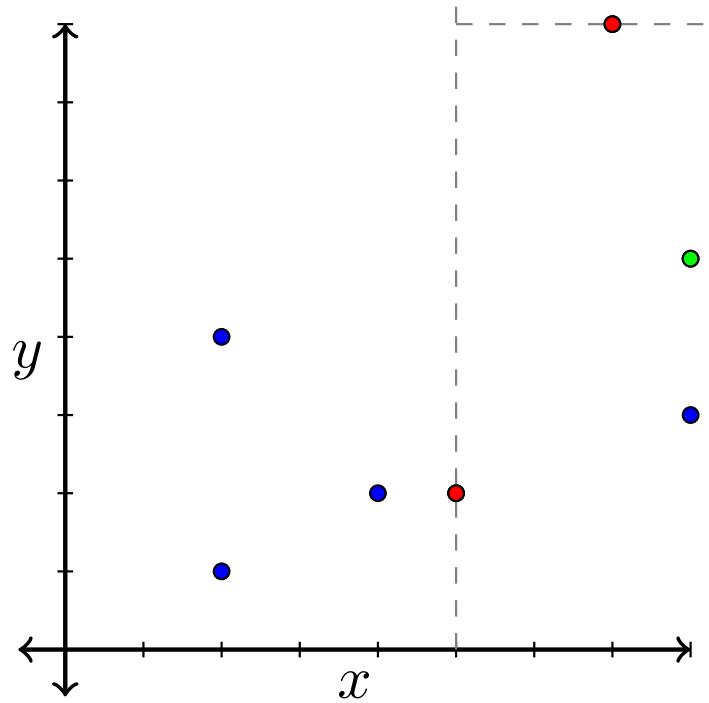


K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

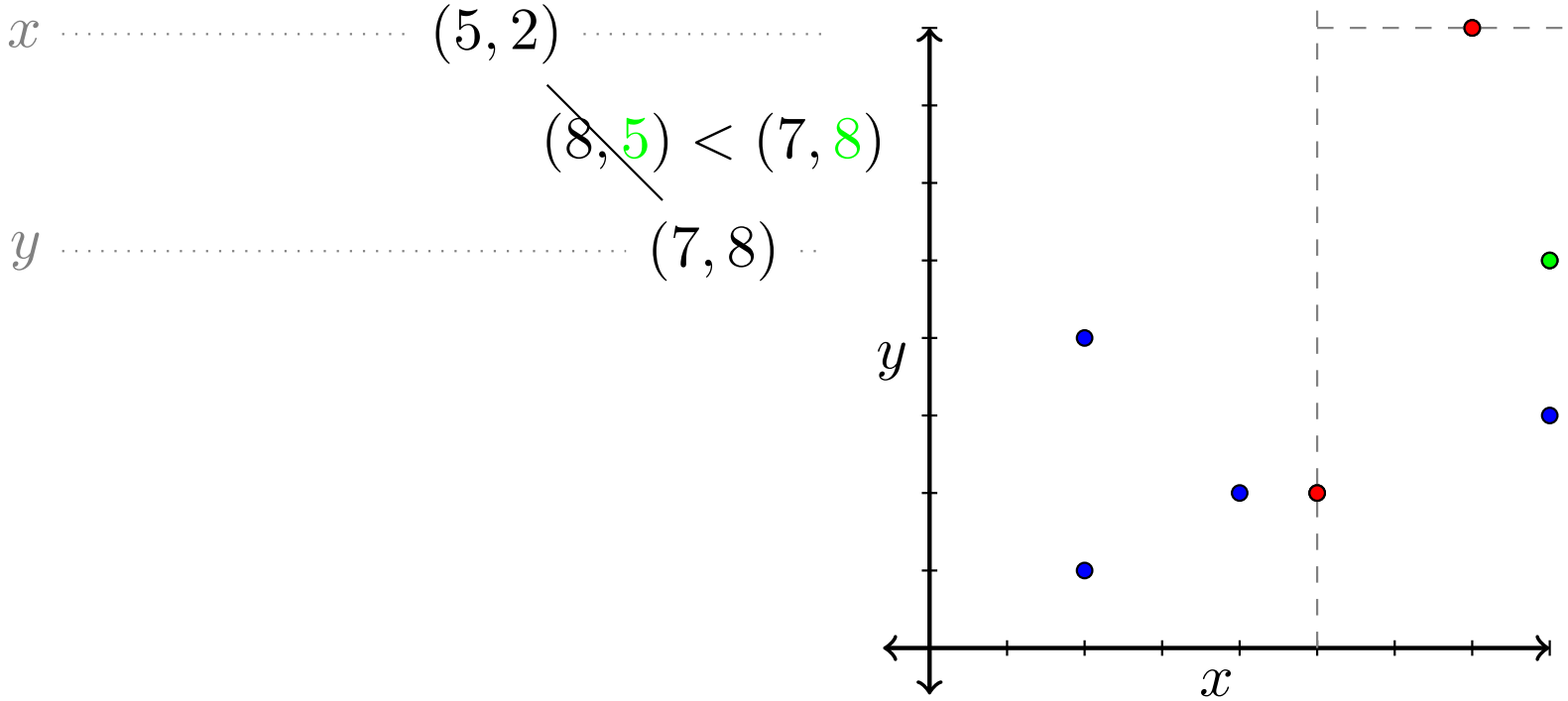
$$(5, 2) \leq (8, 5)$$





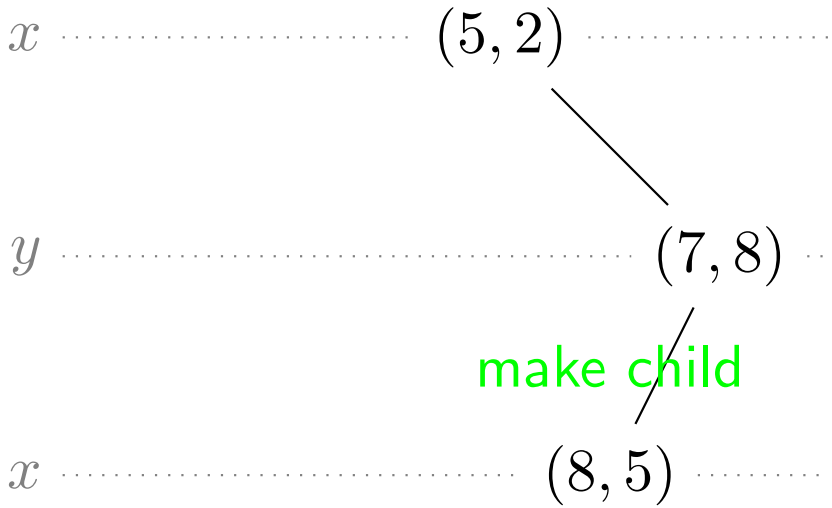
K-Dimensional Trees

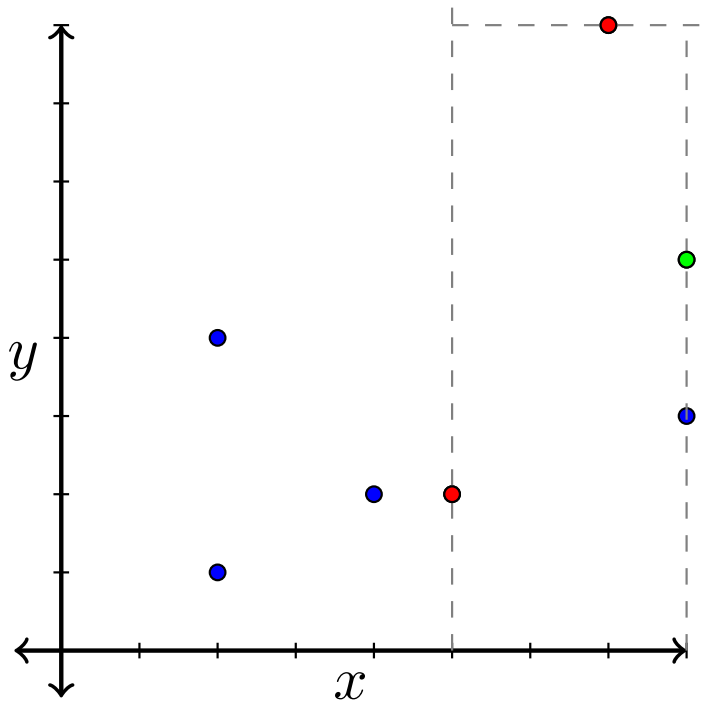
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)



K-Dimensional Trees

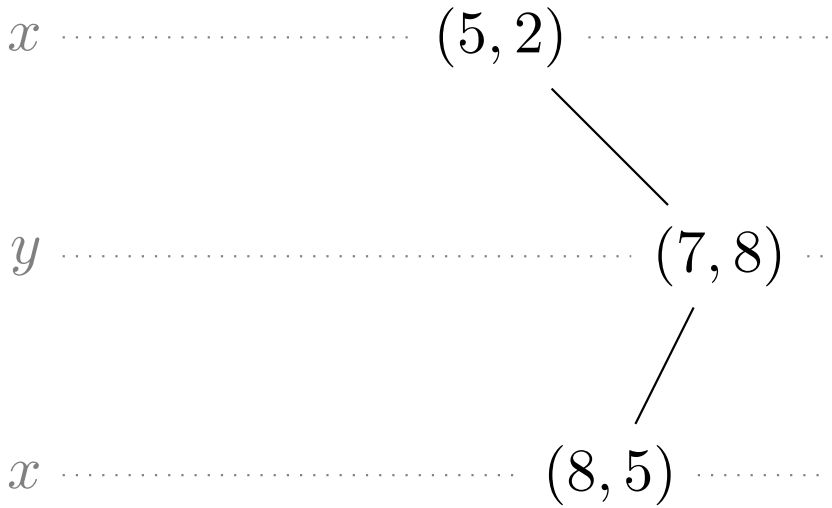
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

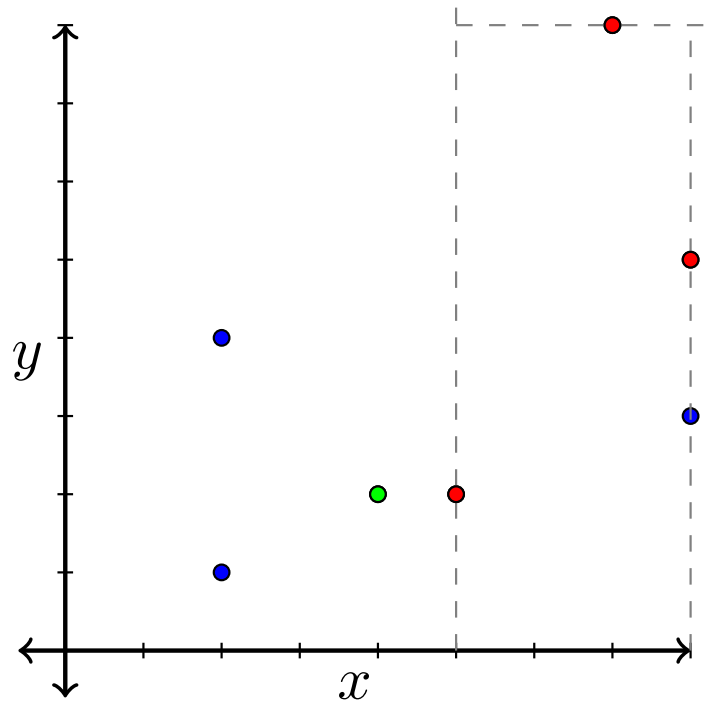




K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

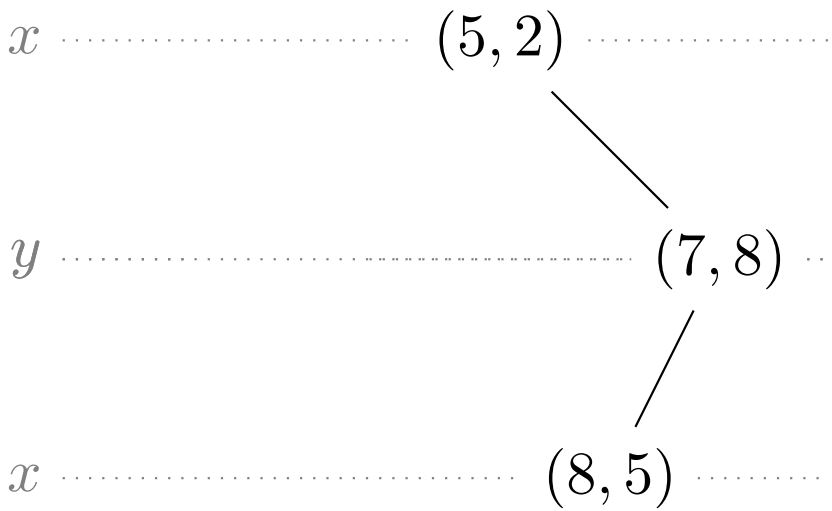


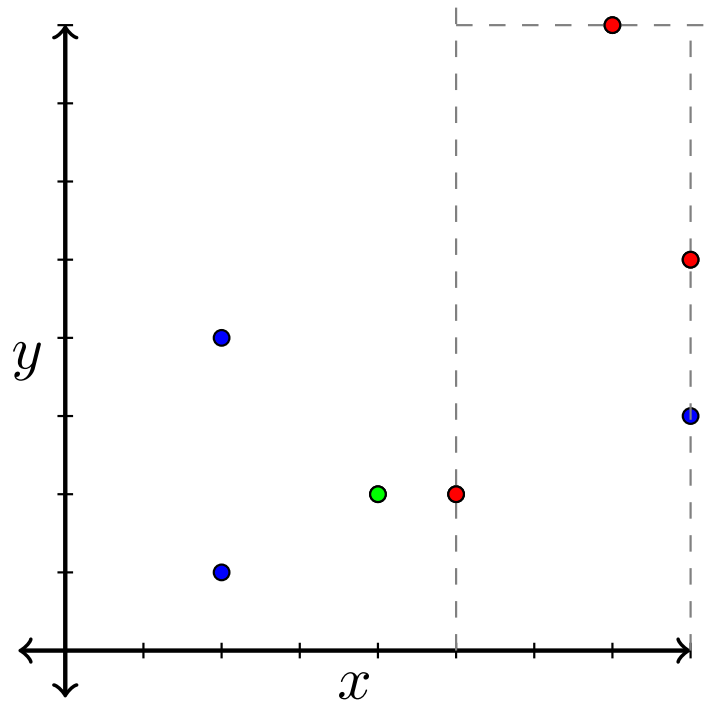


K-Dimensional Trees

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

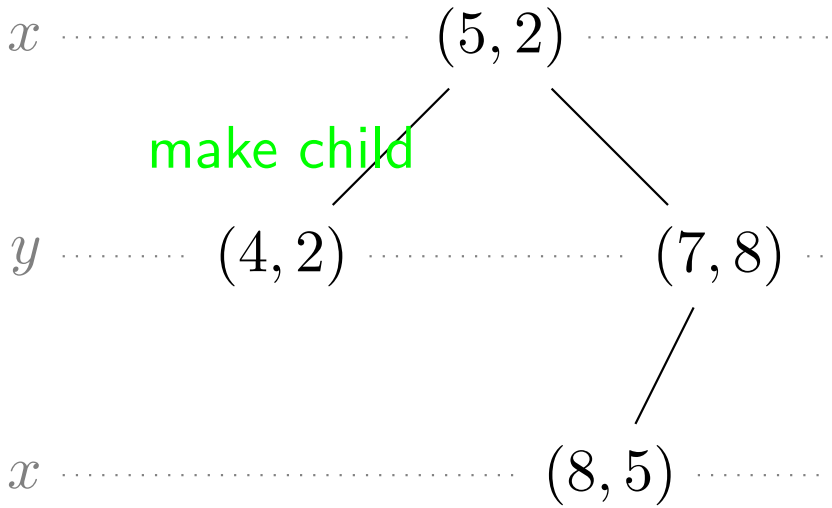
$(4, 2) < (5, 2)$

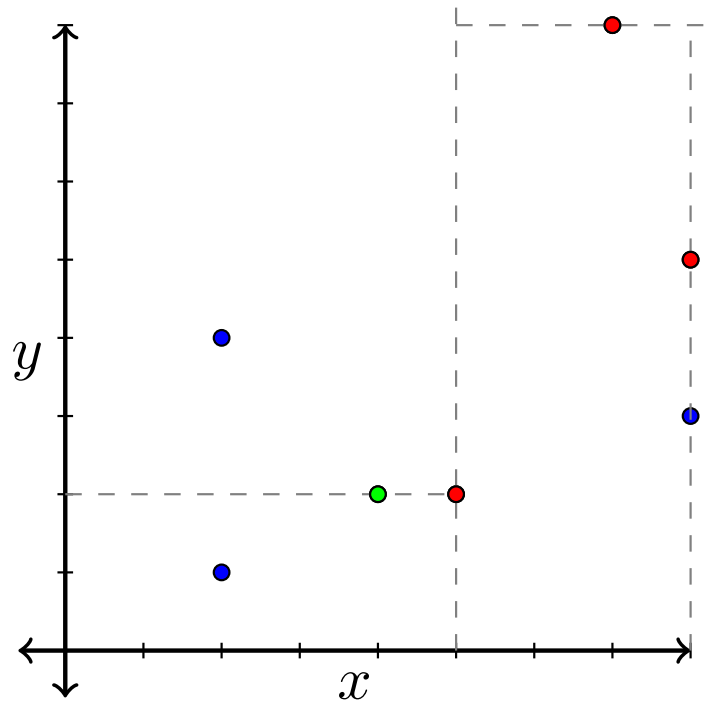




K-Dimensional Trees

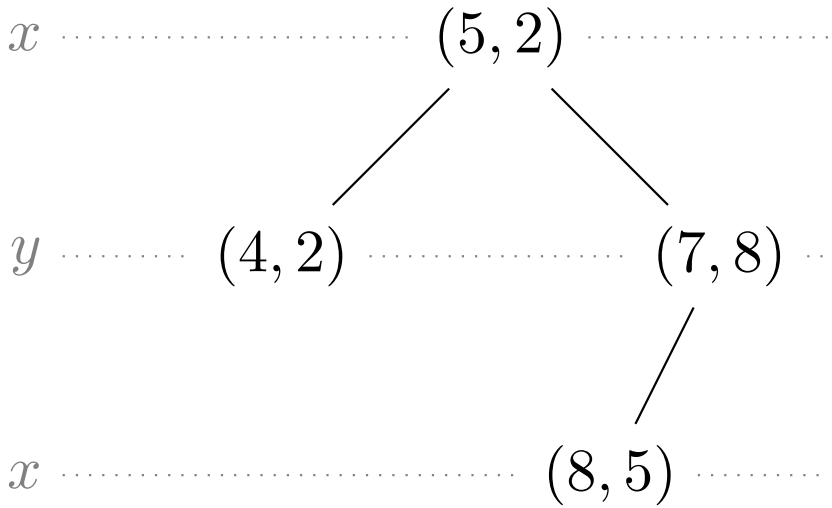
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

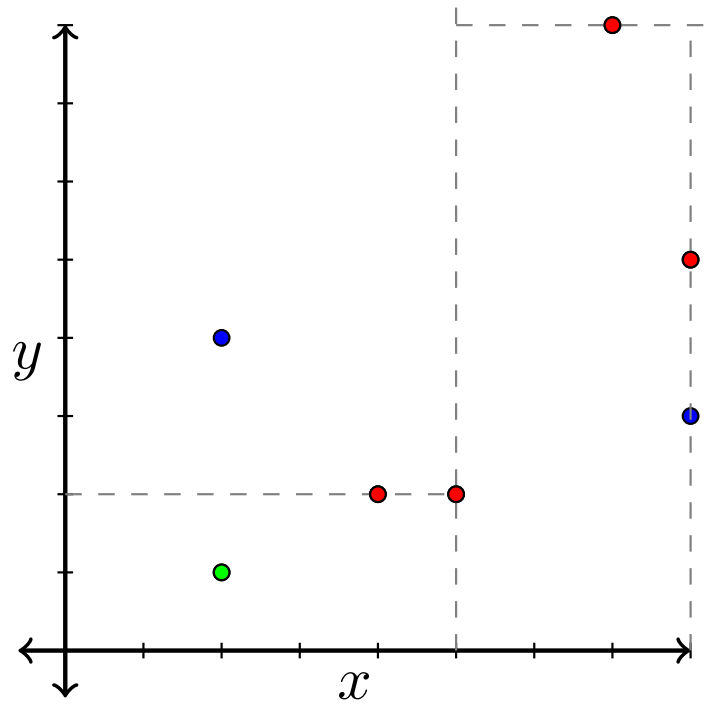




K-Dimensional Trees

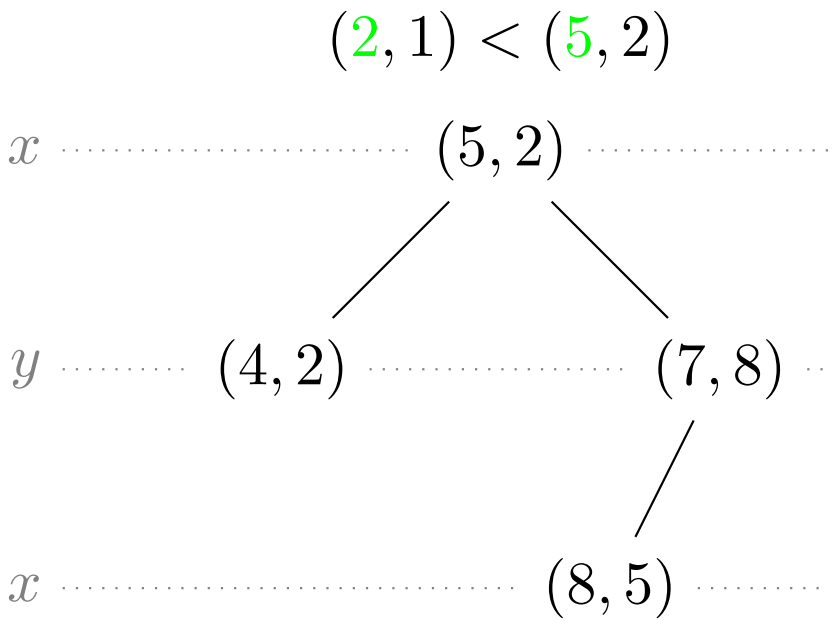
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

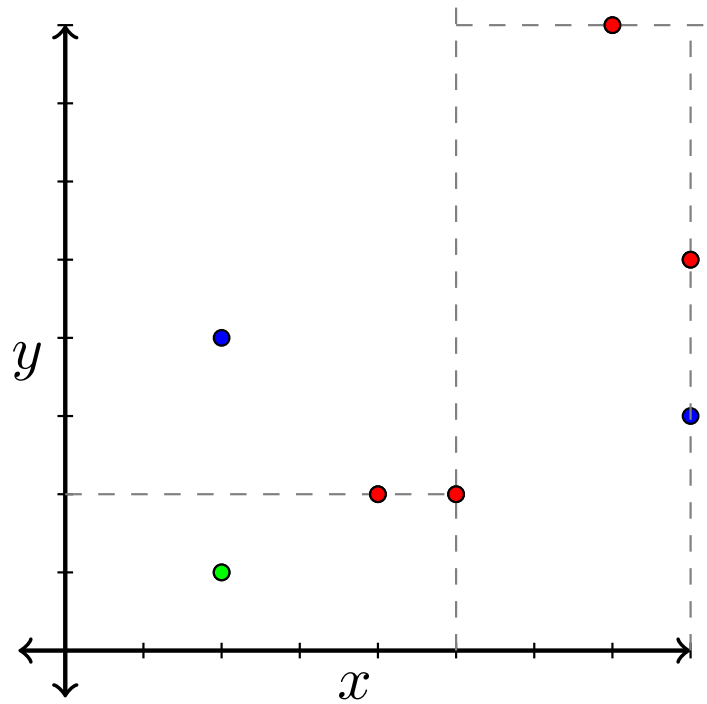




K-Dimensional Trees

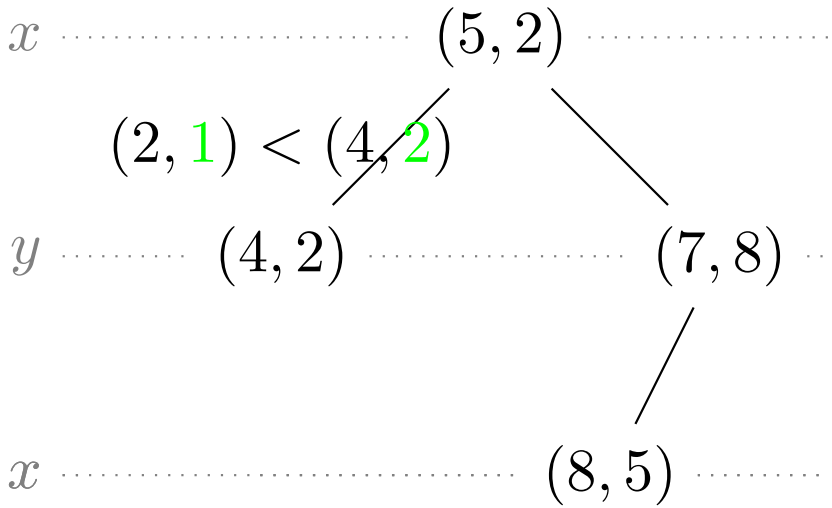
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

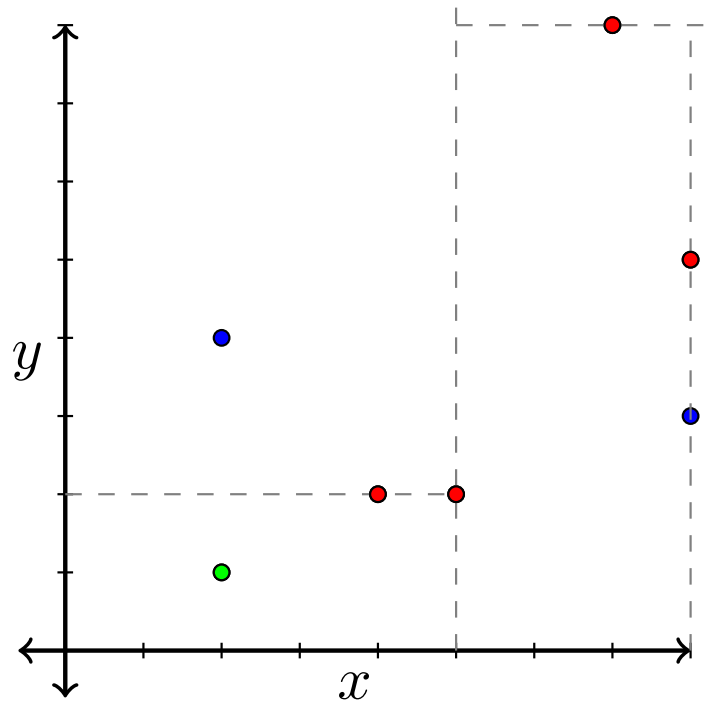




K-Dimensional Trees

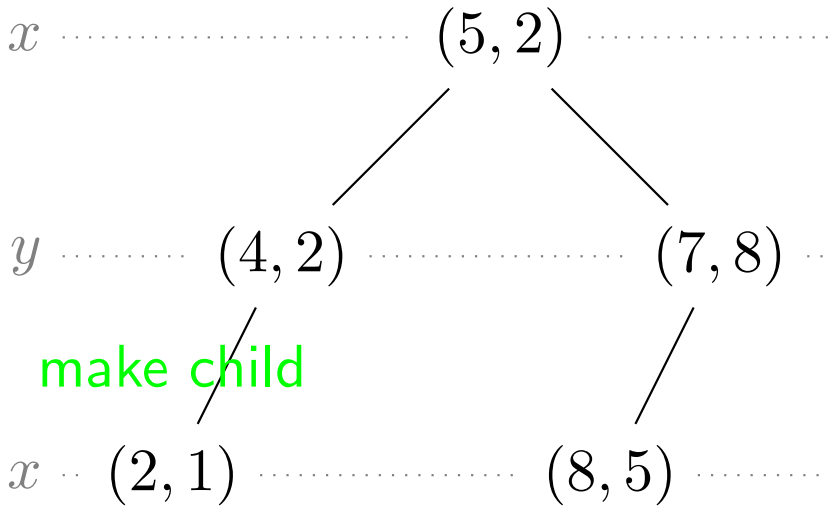
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

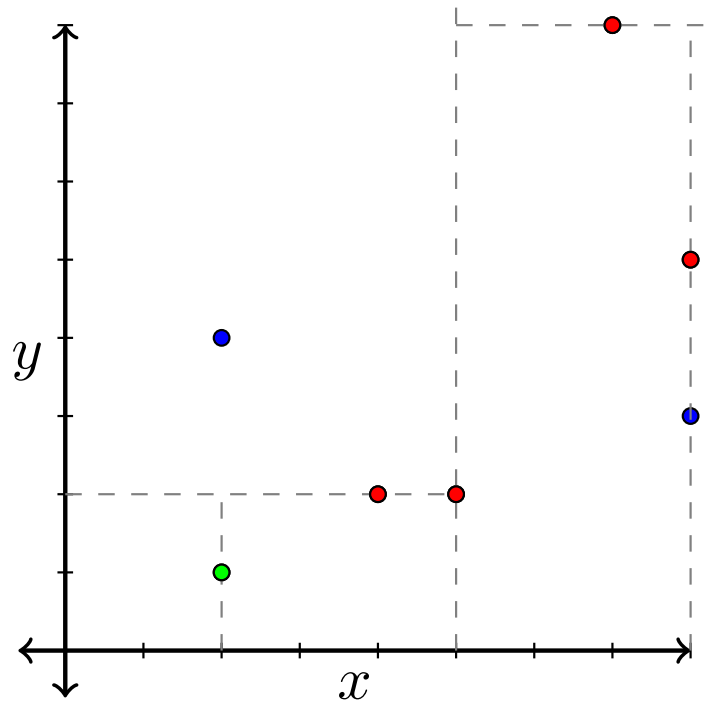




K-Dimensional Trees

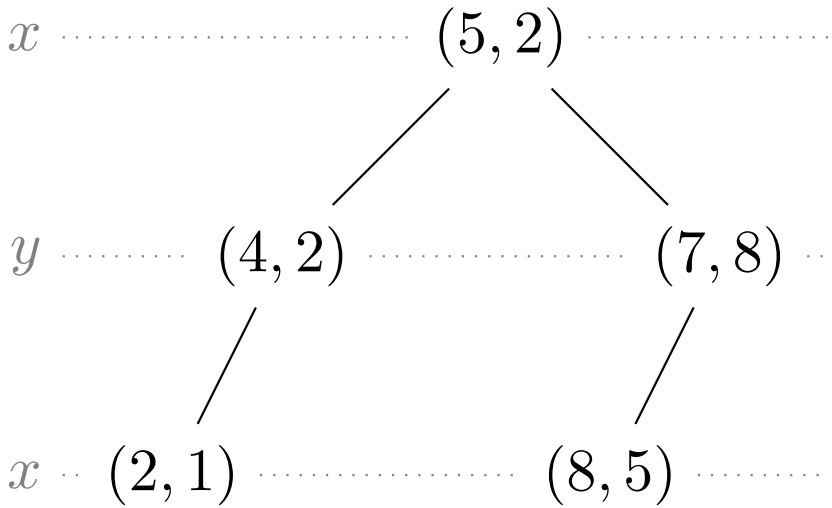
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

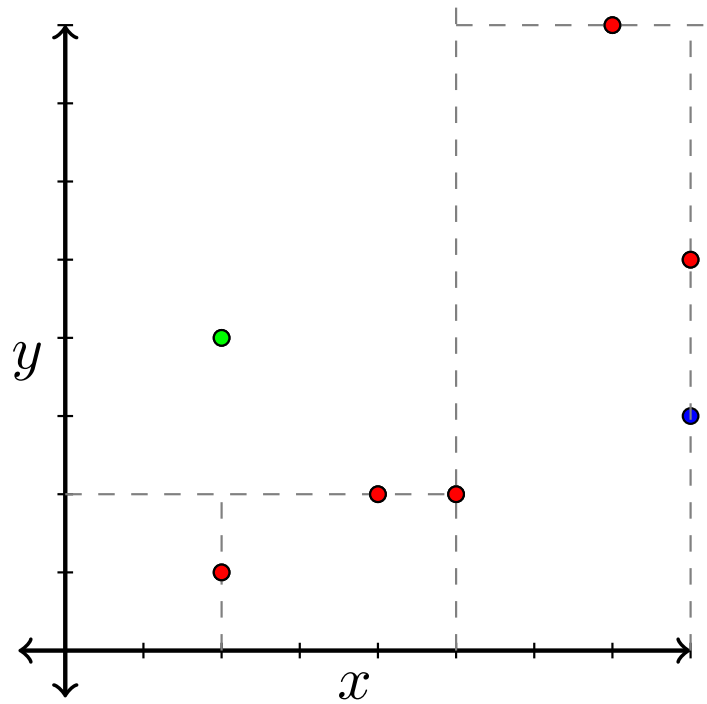




K-Dimensional Trees

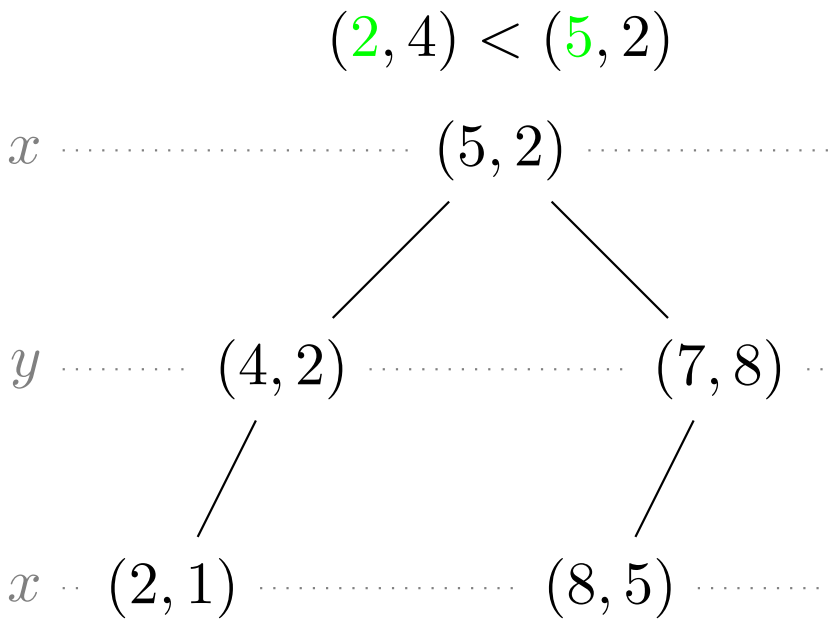
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

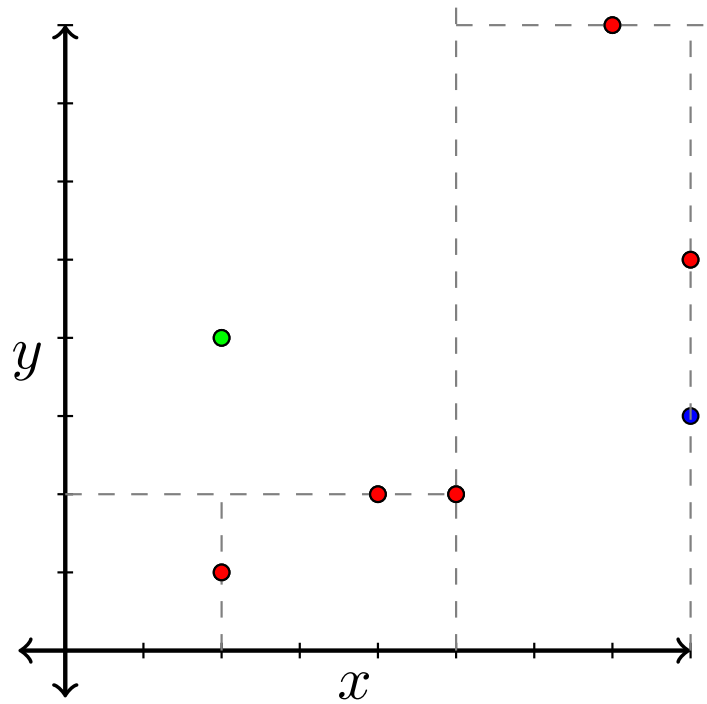




K-Dimensional Trees

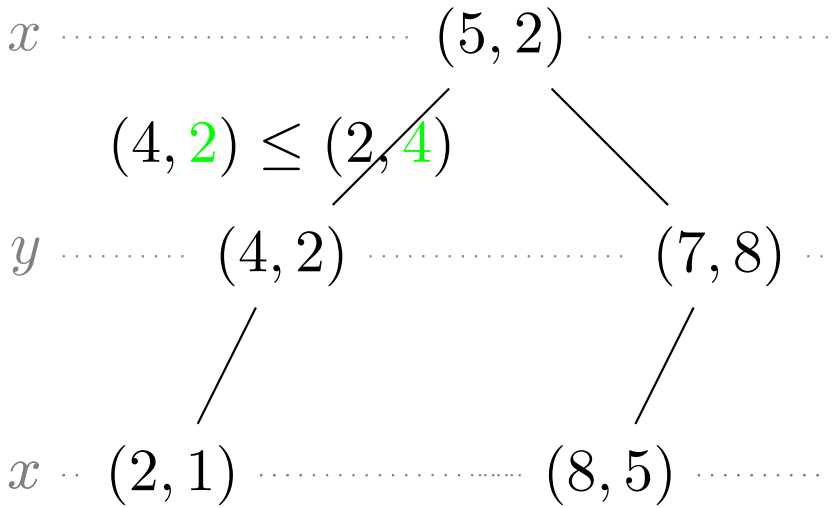
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

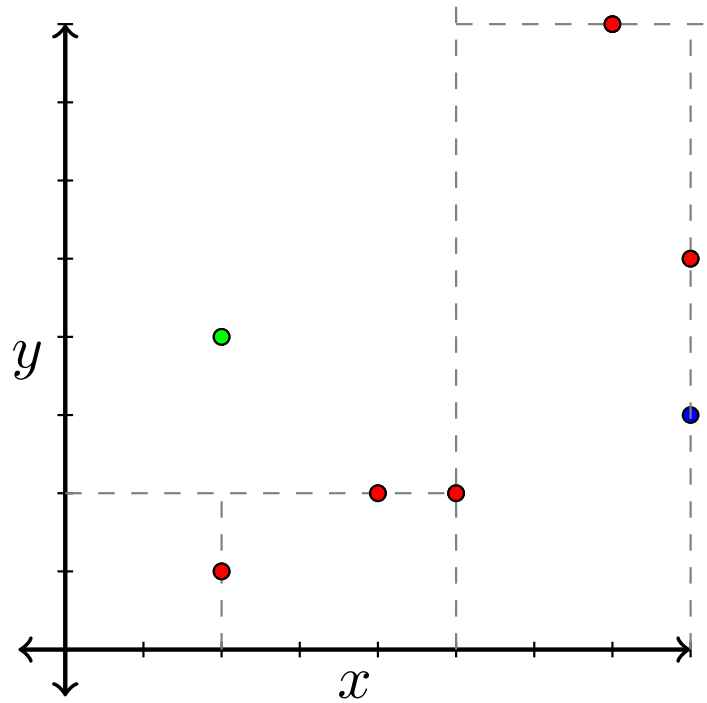




K-Dimensional Trees

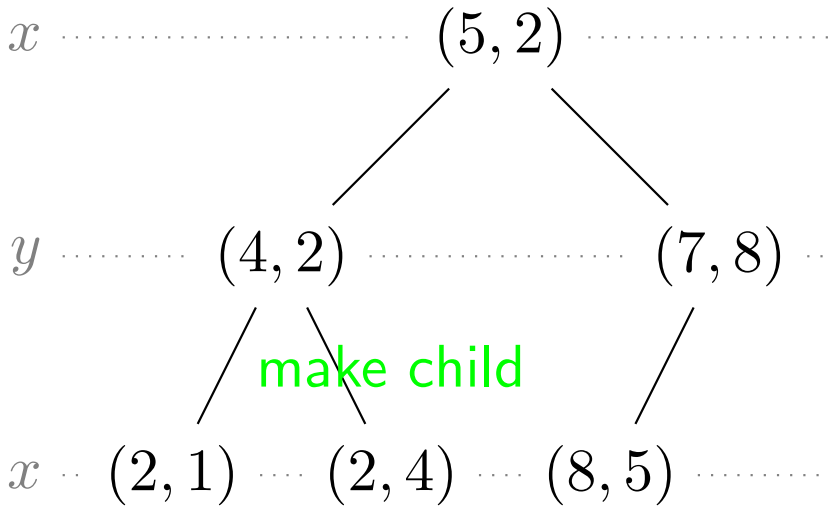
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

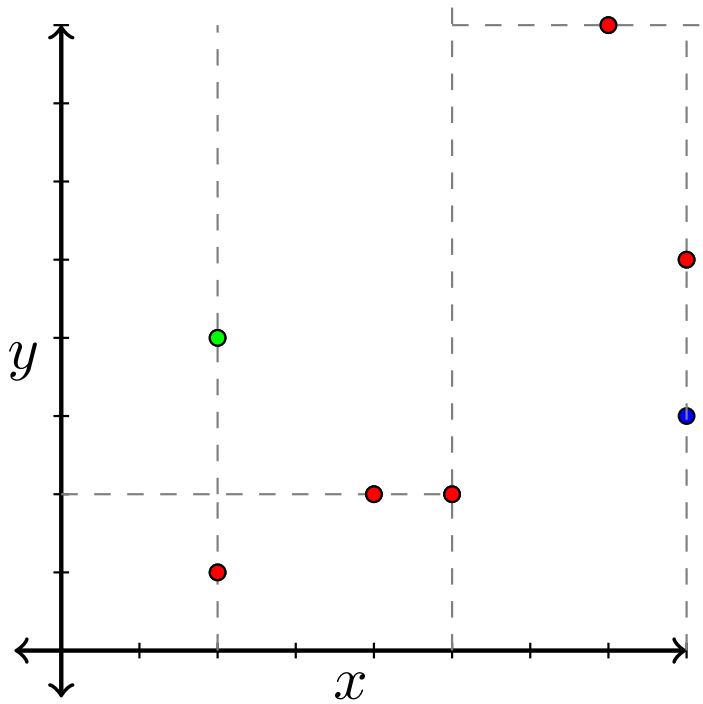




K-Dimensional Trees

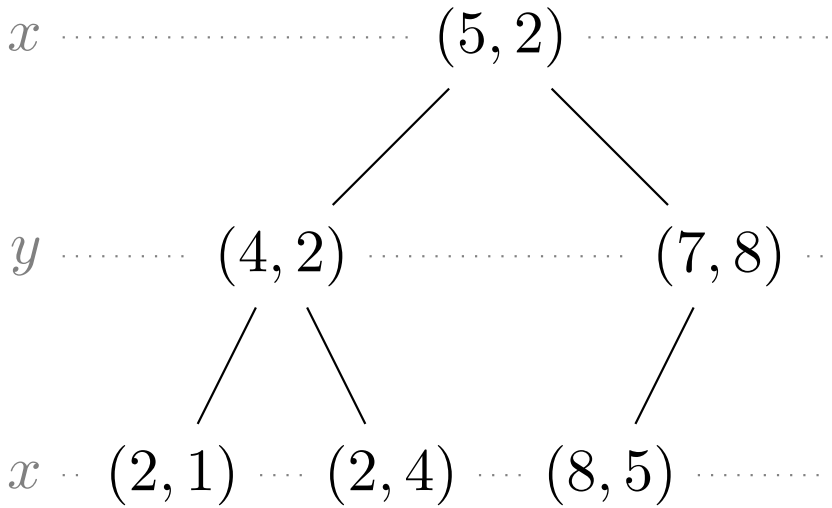
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

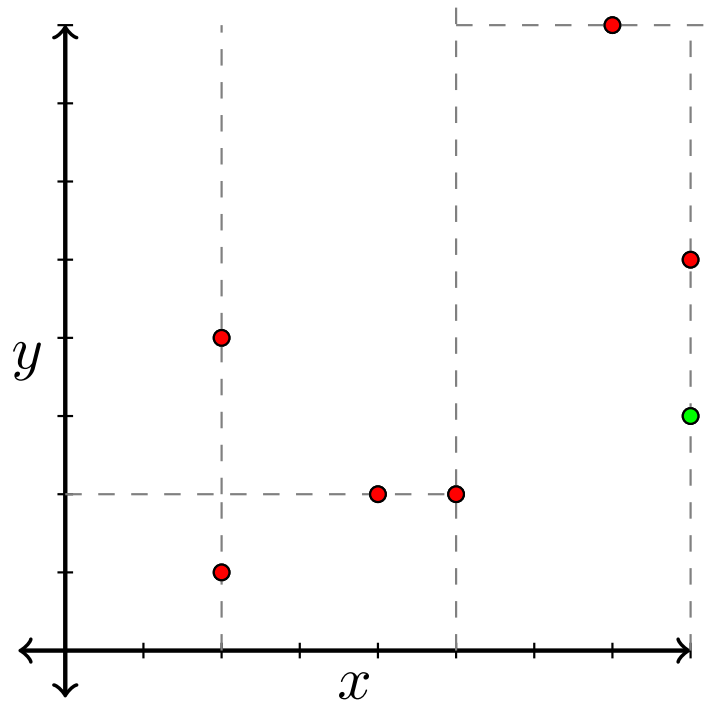




K-Dimensional Trees

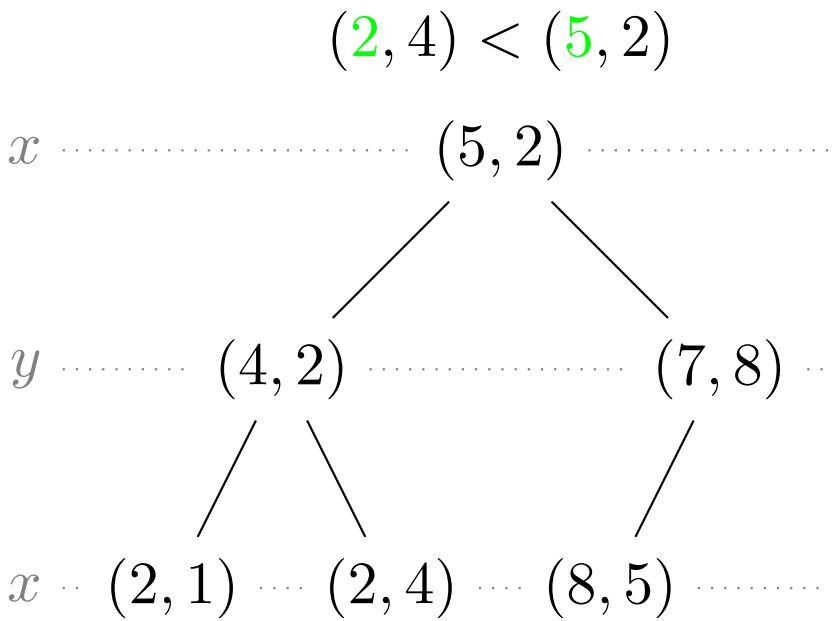
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

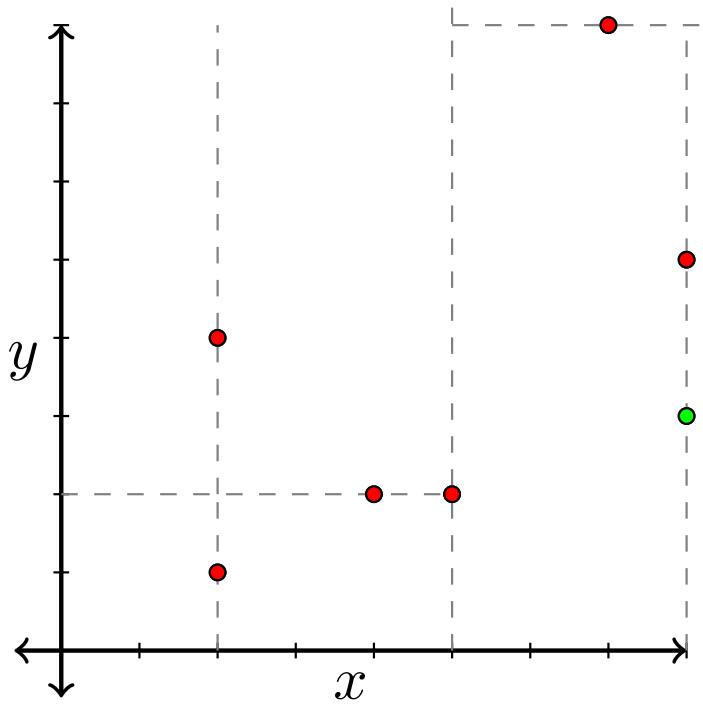




K-Dimensional Trees

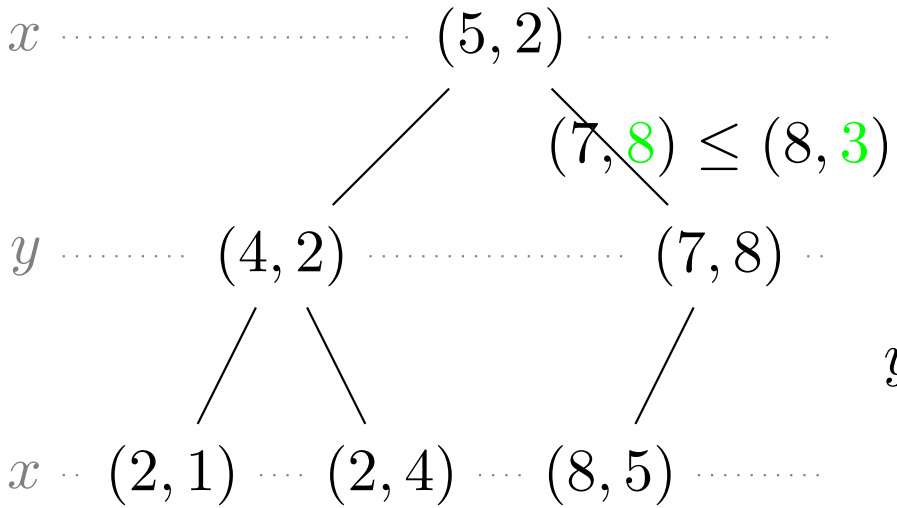
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

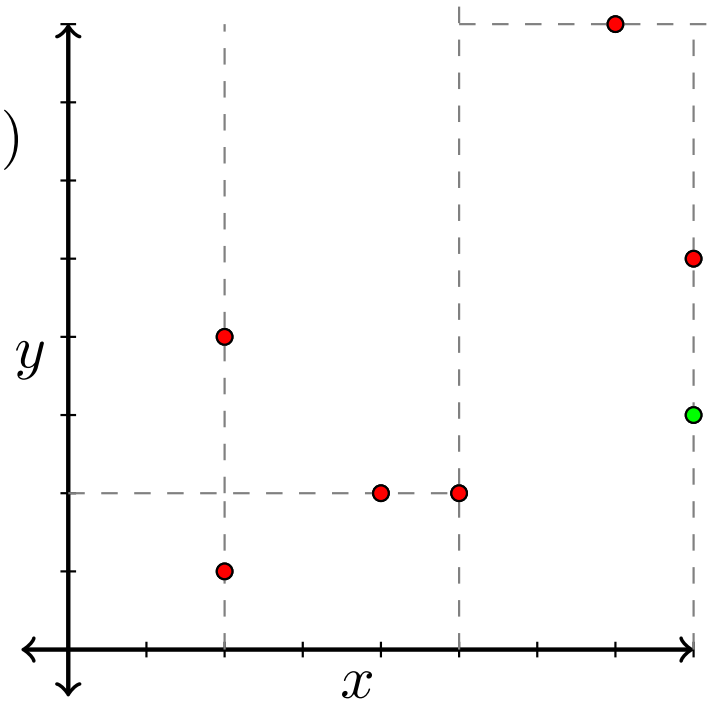




K-Dimensional Trees

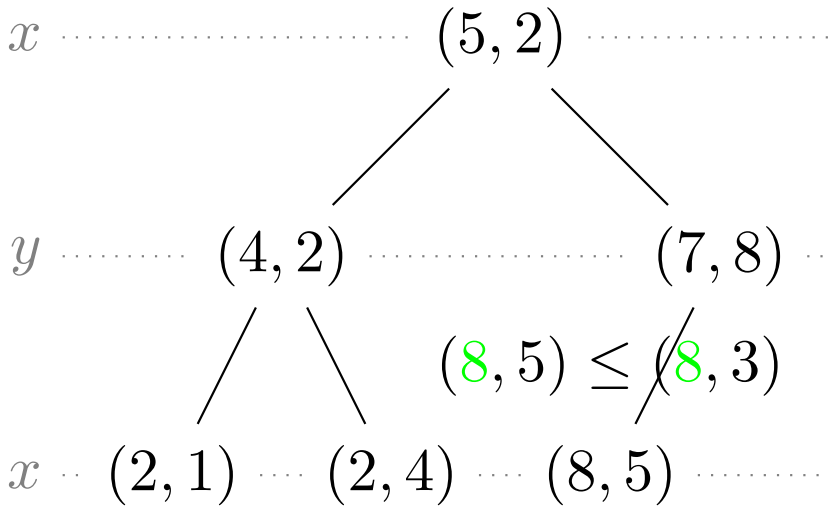
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

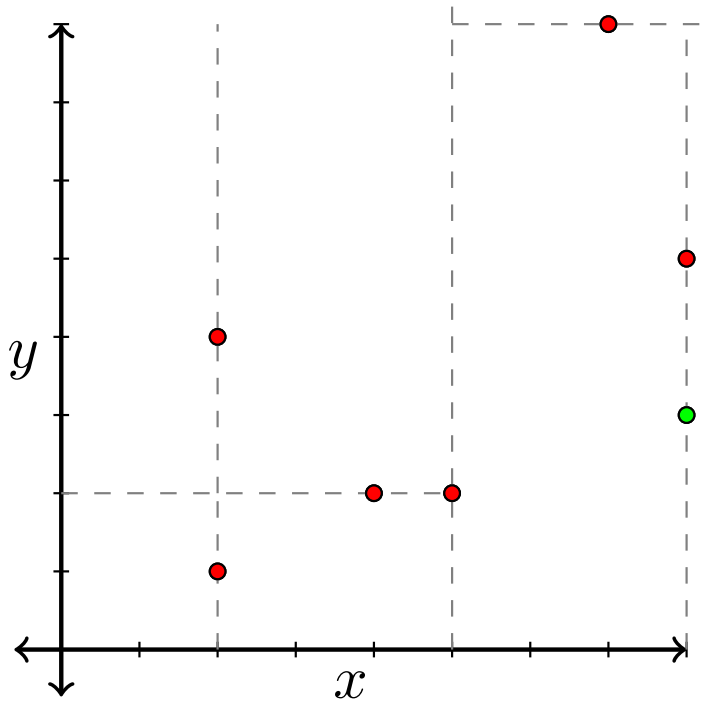




K-Dimensional Trees

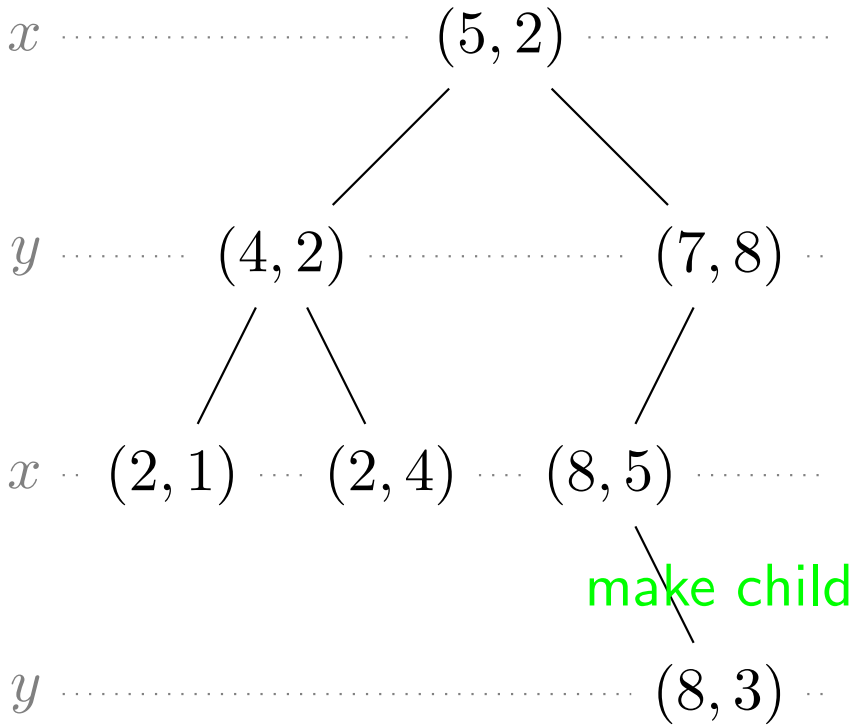
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

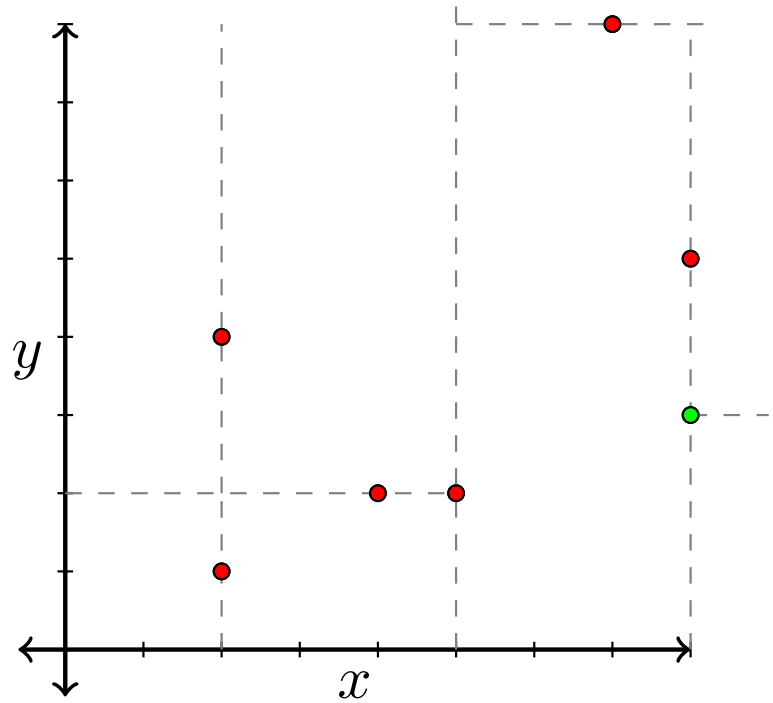




K-Dimensional Trees

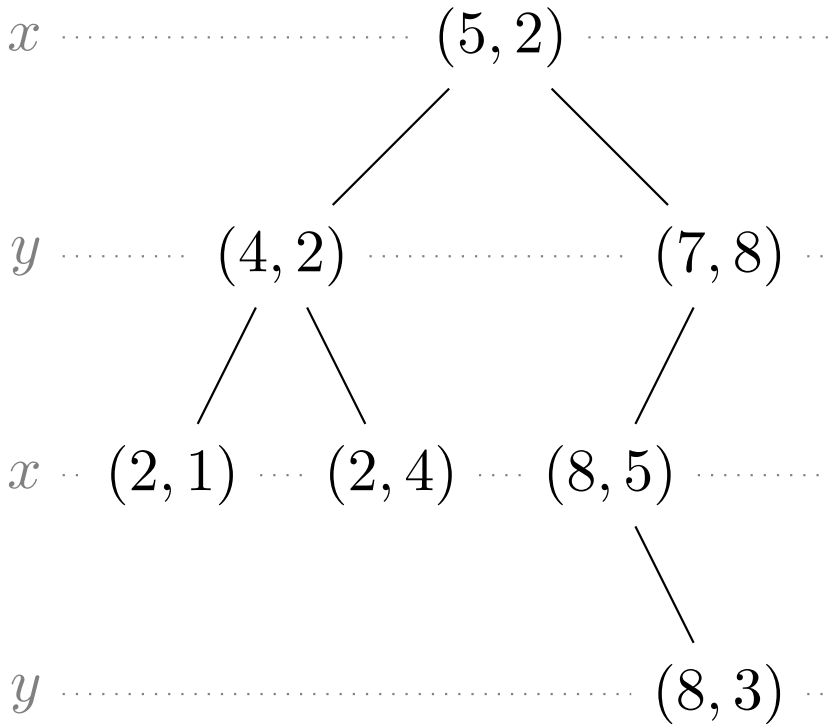
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

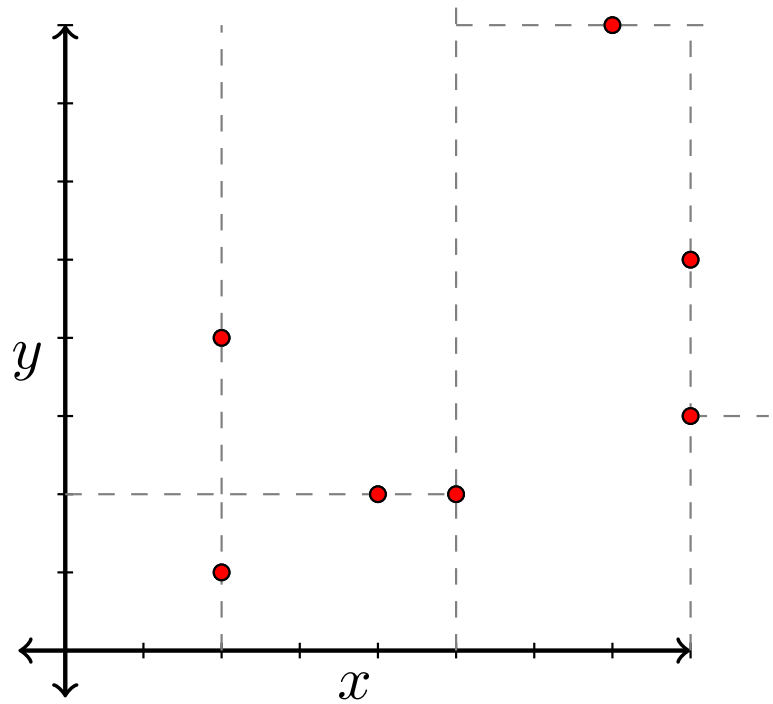




K-Dimensional Trees

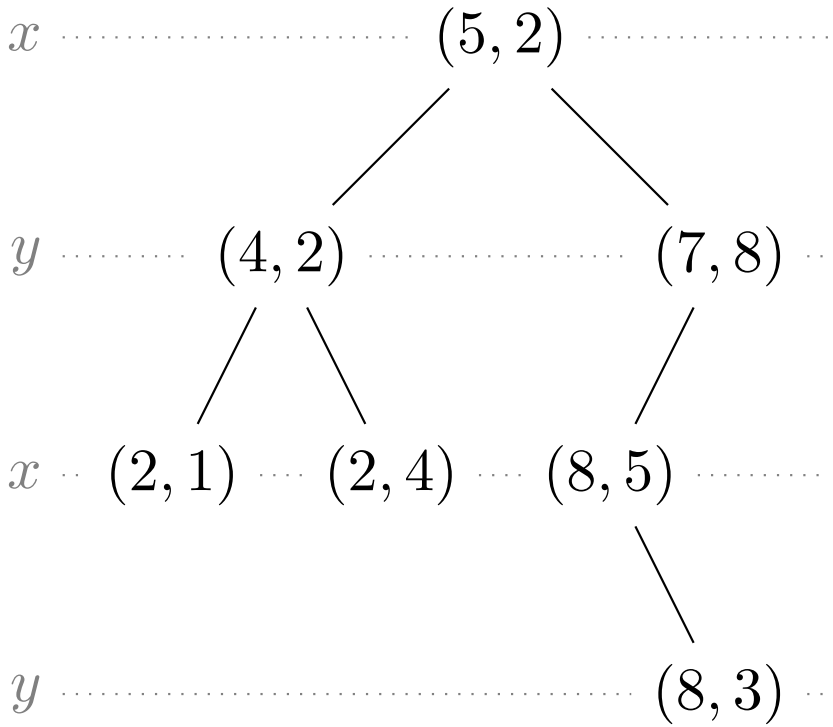
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

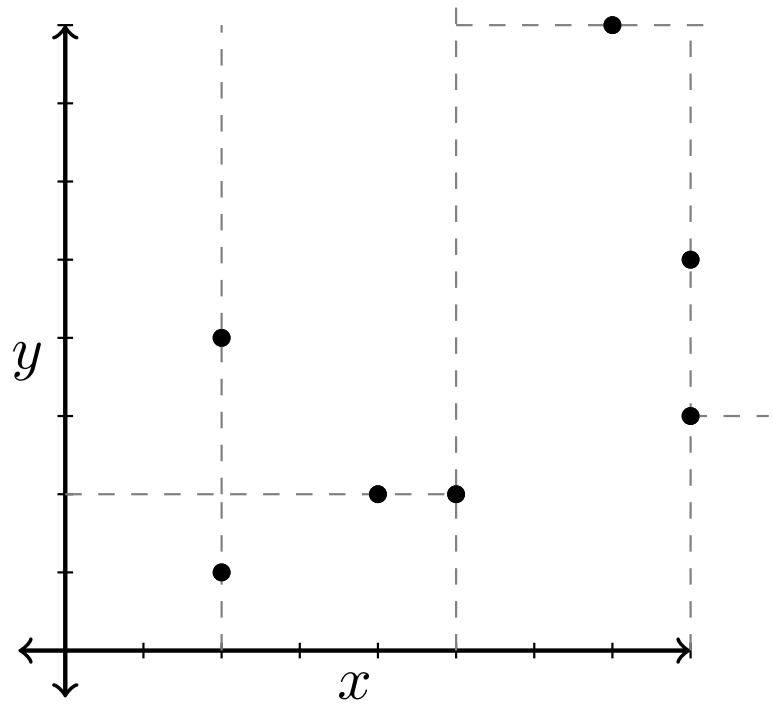




K-Dimensional Trees: Nearest Neighbor

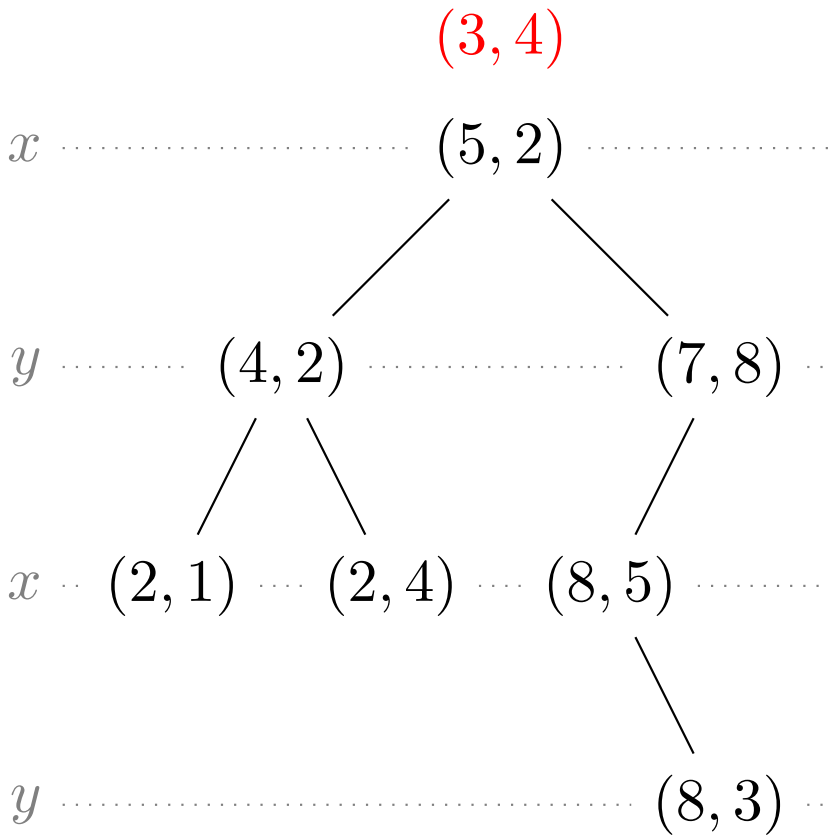
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

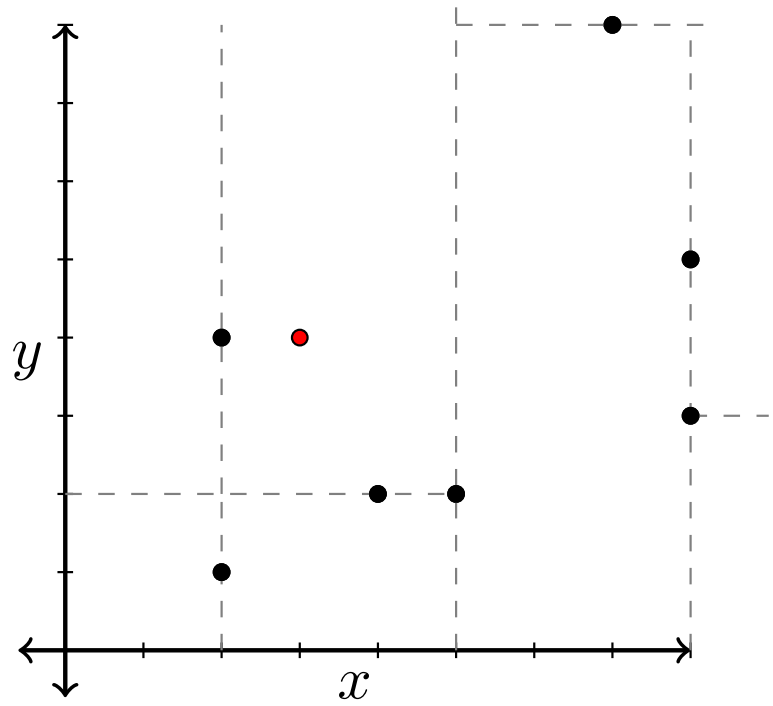




K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

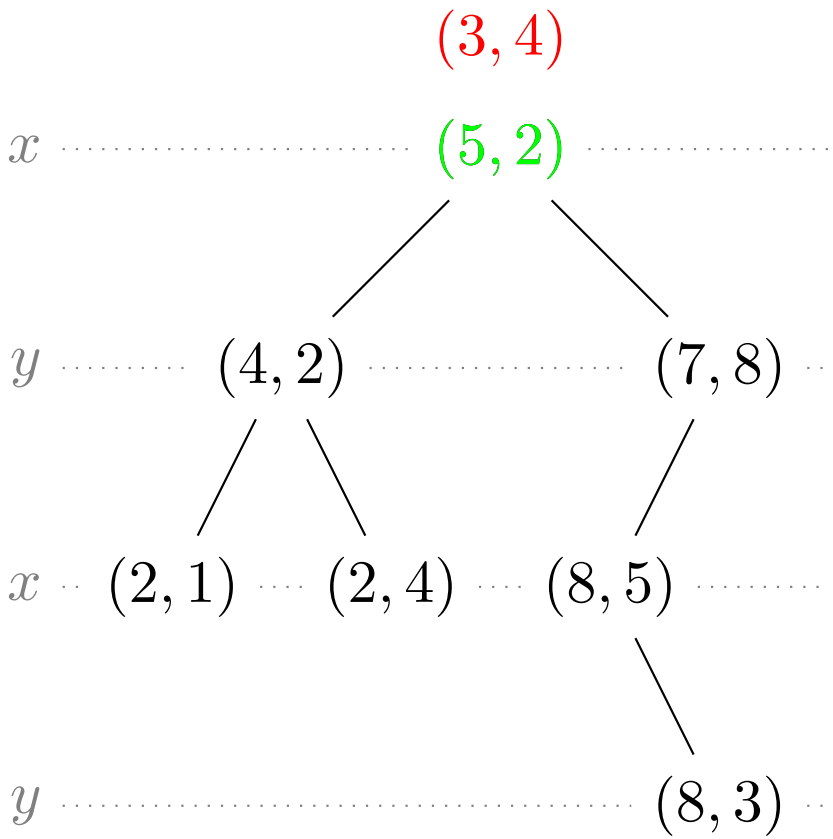


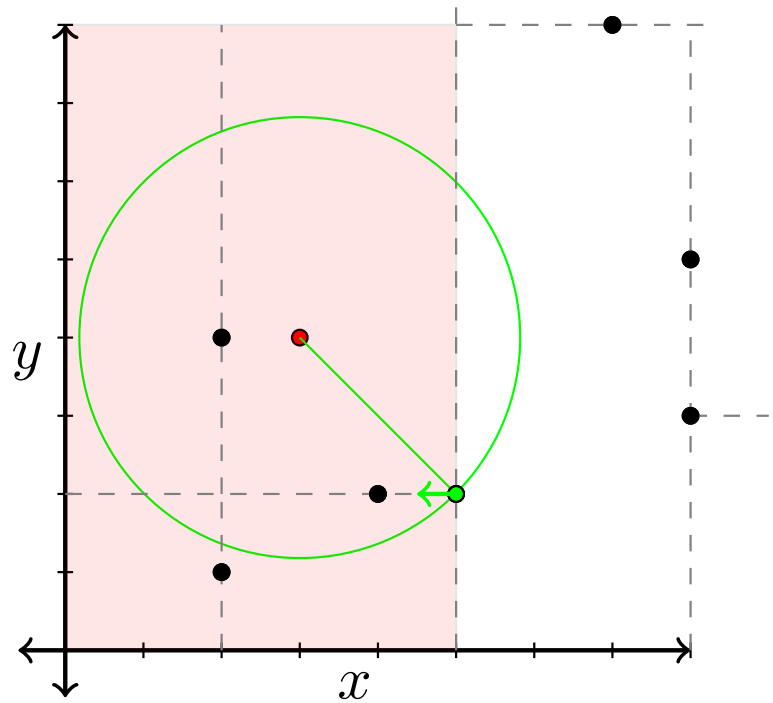


Find nearest neighbor to point $(3, 4)$

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

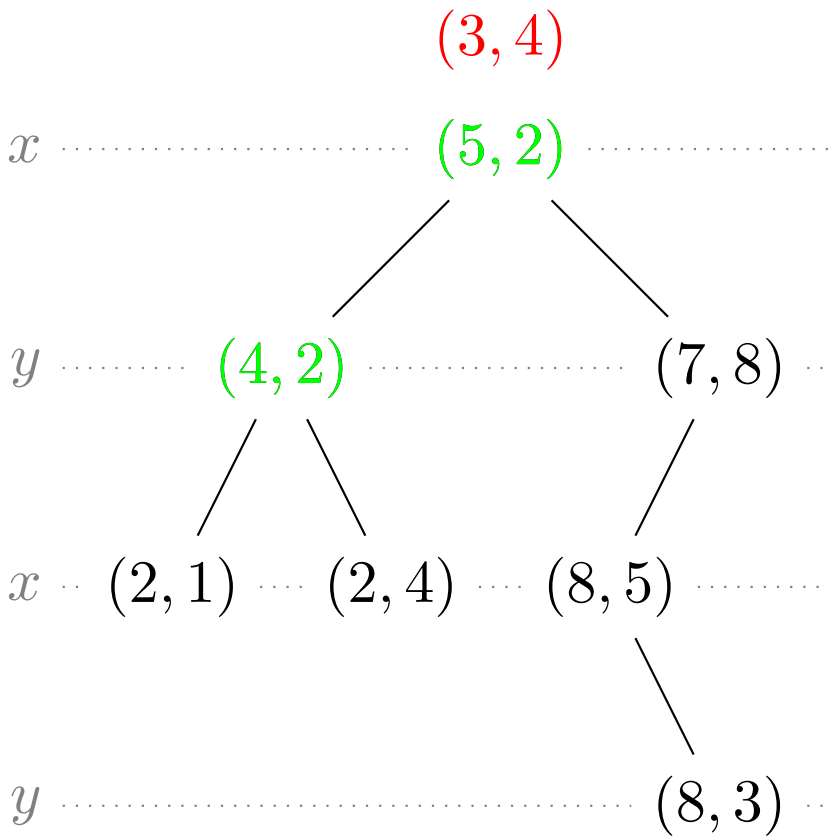


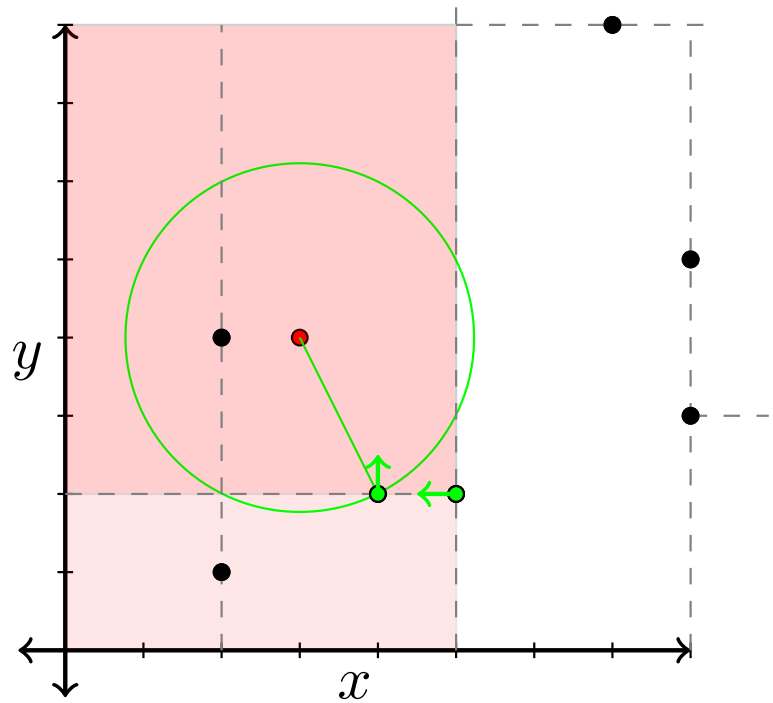


Save minimum distance to (5, 4)

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

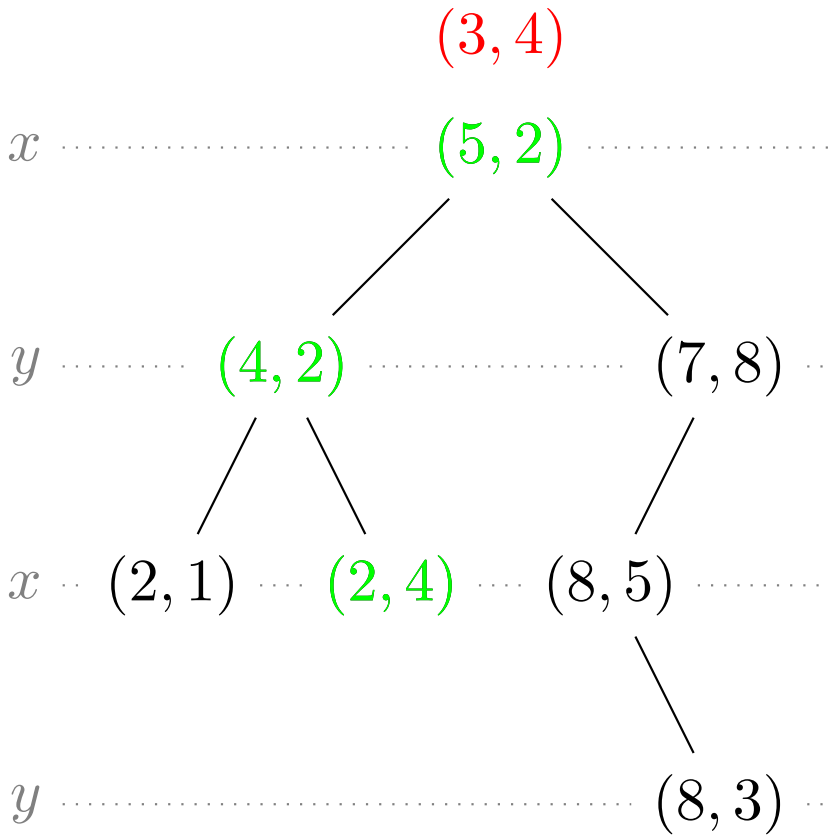


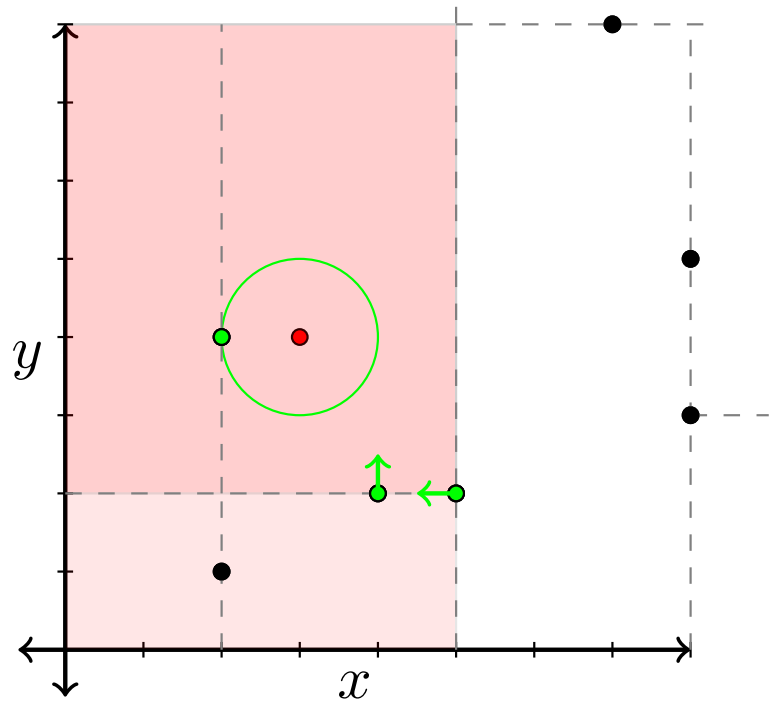


Save minimum distance to (4, 2)

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

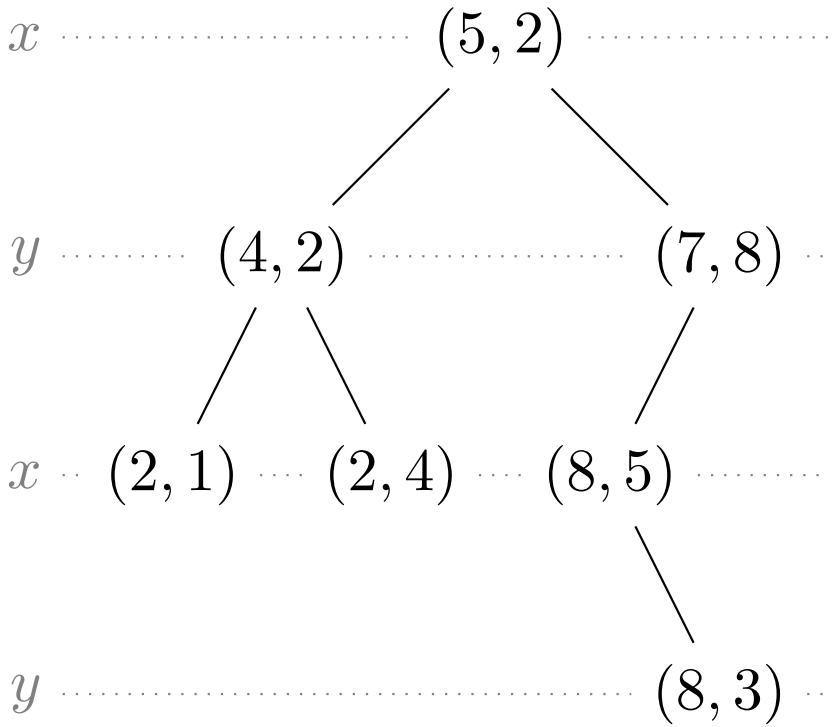


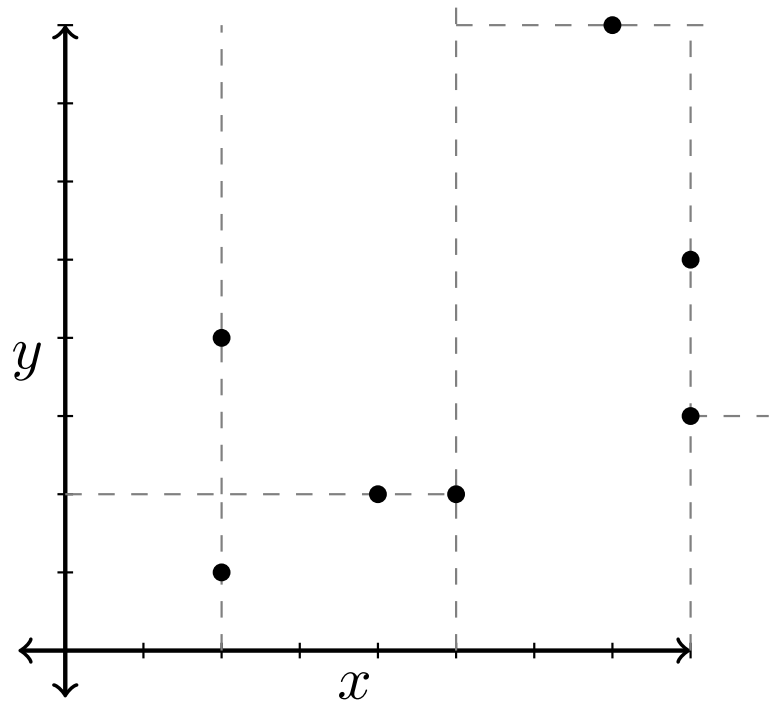


Save minimum distance to (2, 4) Is last leaf $(2, 4)$ always the nearest neighbor?

K-Dimensional Trees: Nearest Neighbor

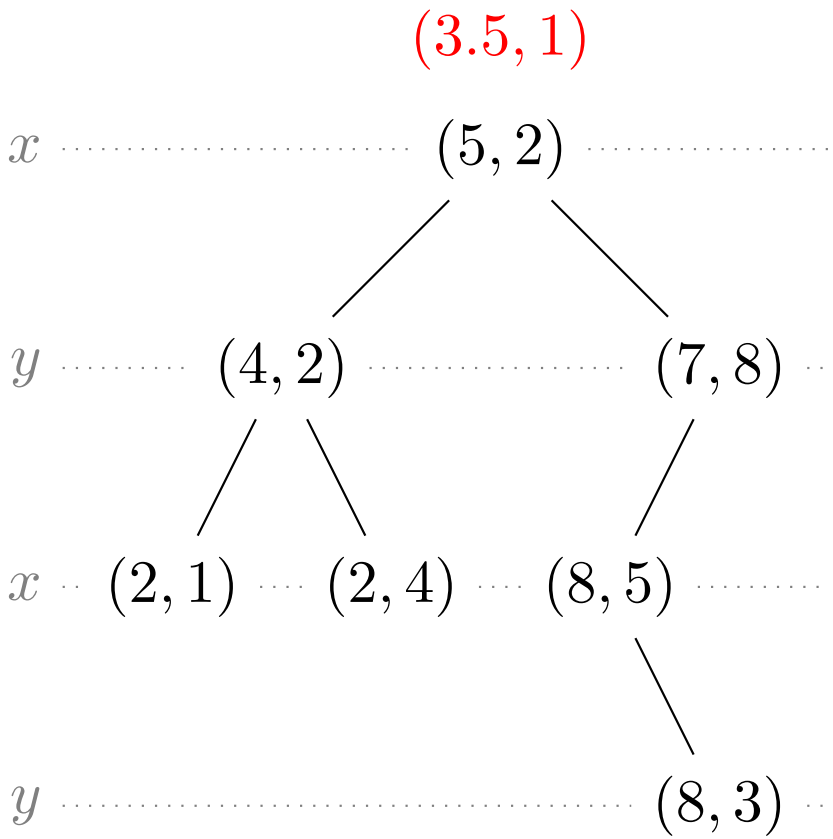
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

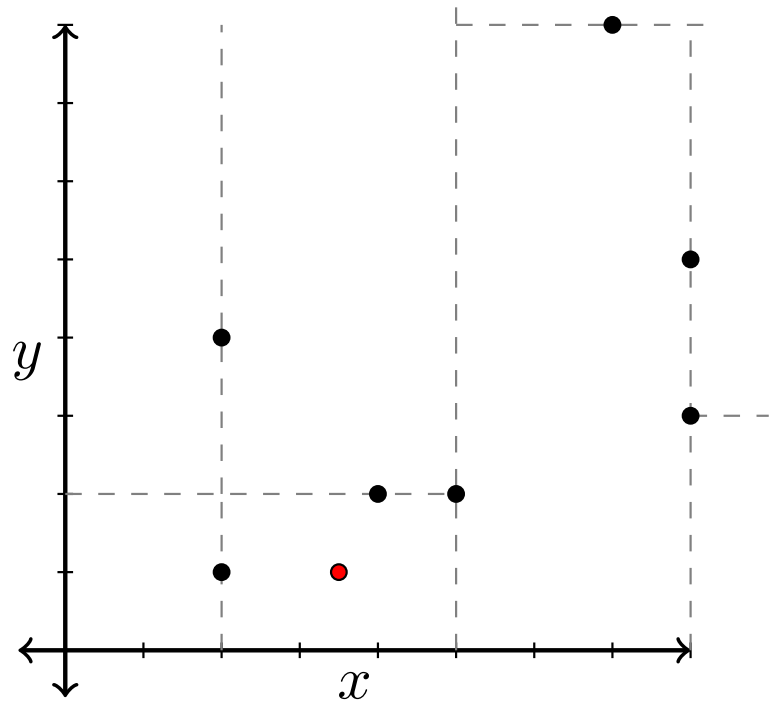




K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

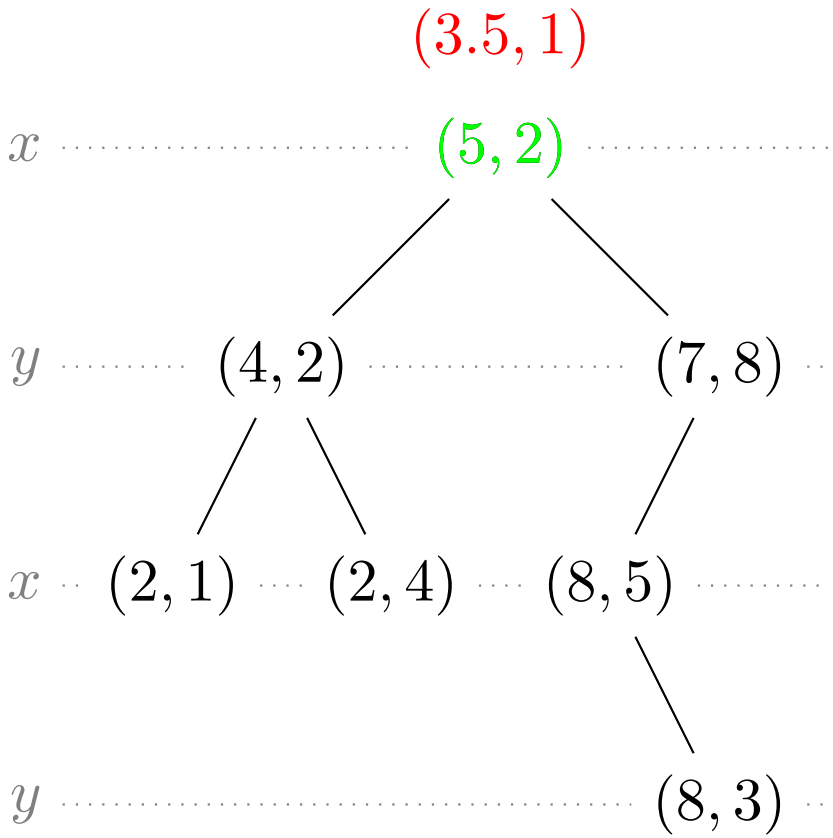


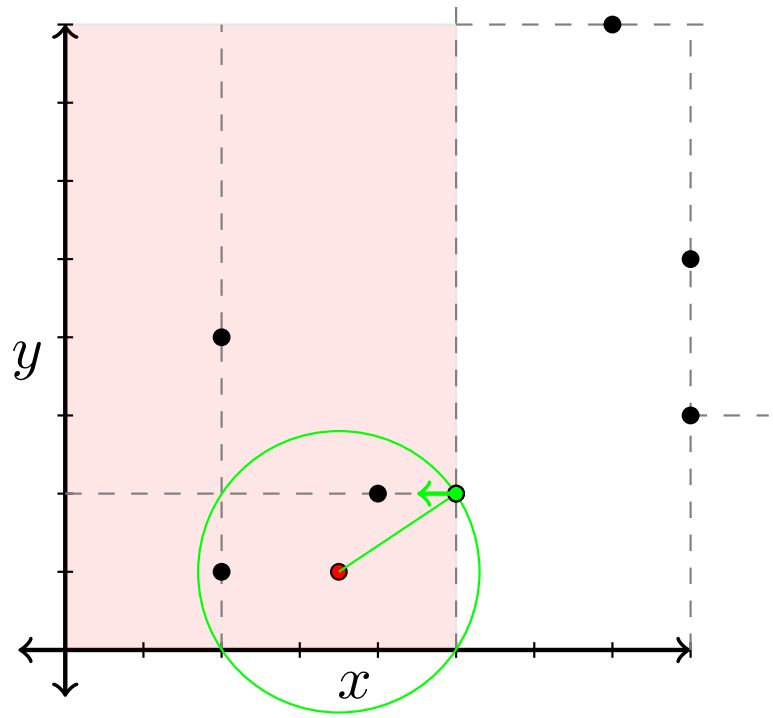


Find nearest neighbor to (3.5, 1)

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

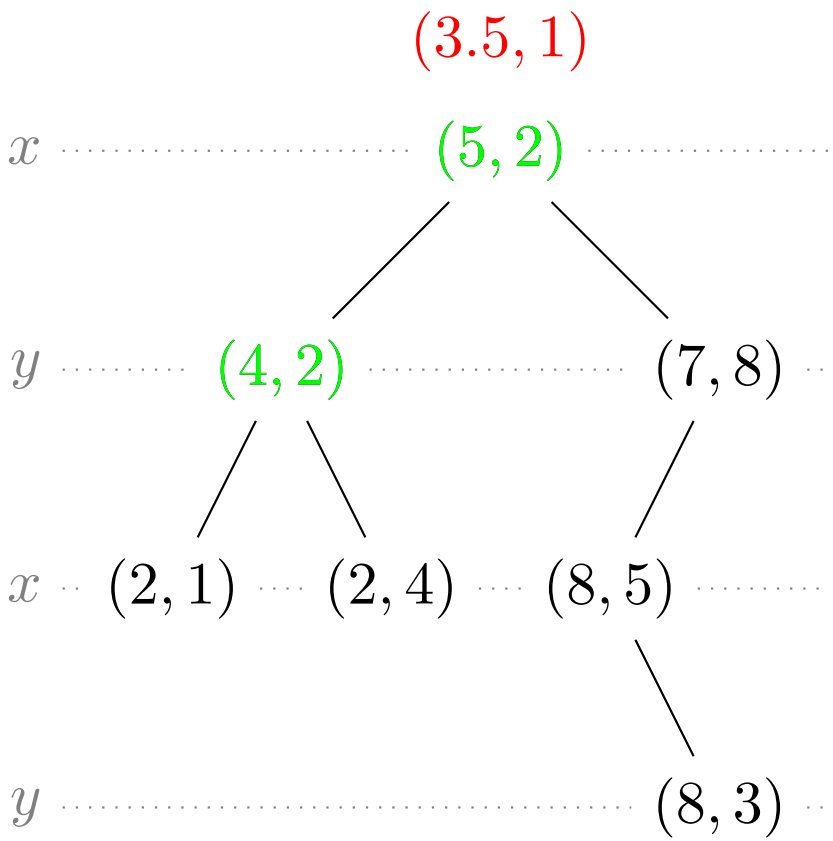


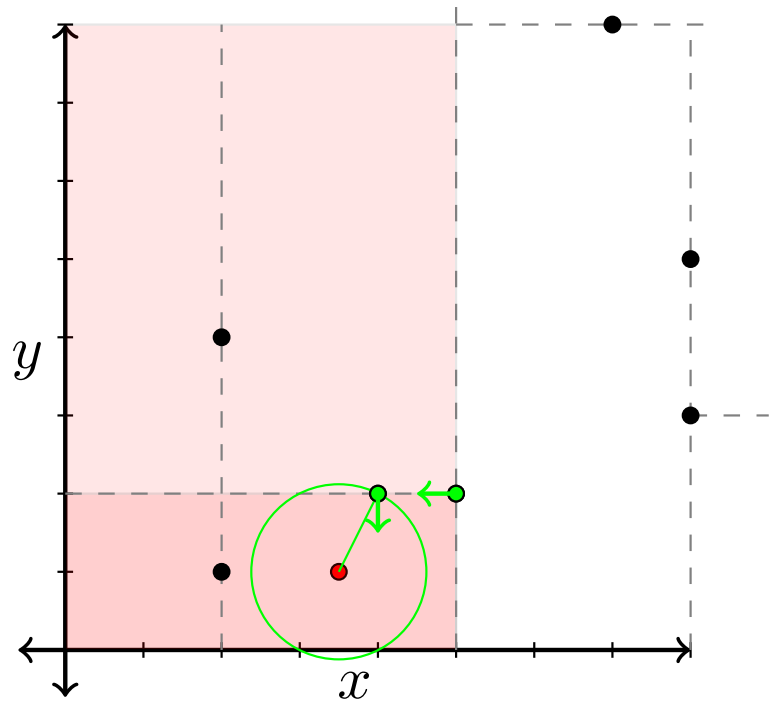


Set minimum distance; cannot prune

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

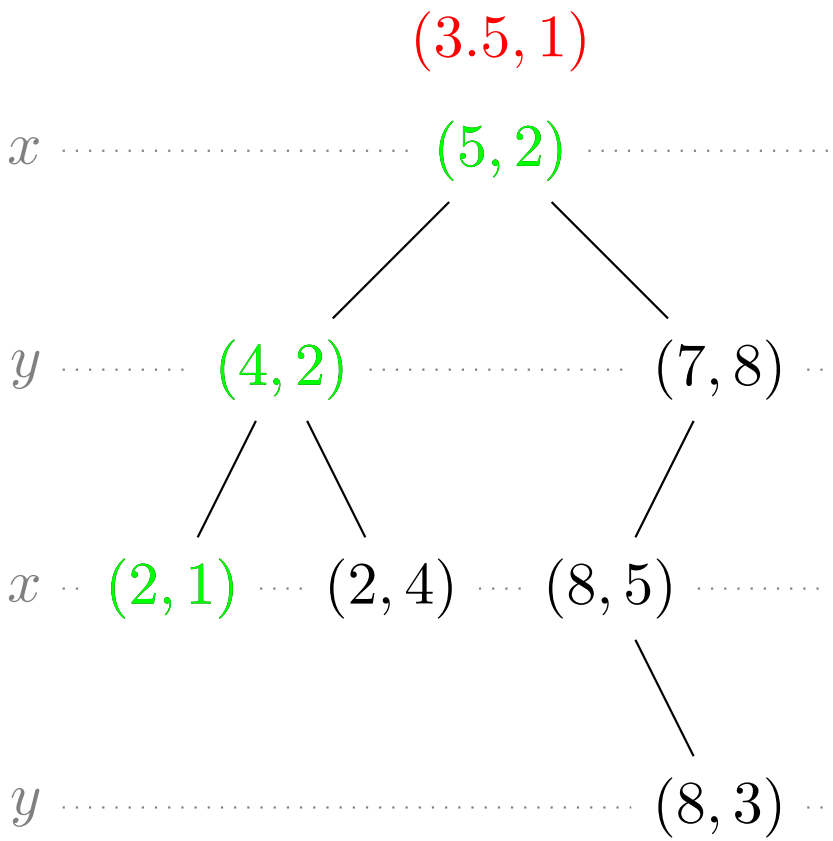


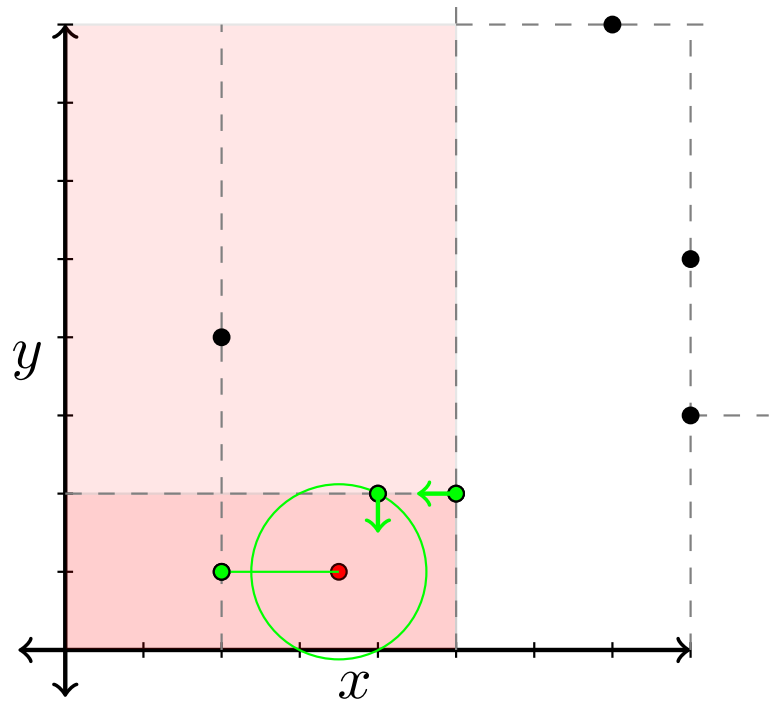


Set minimum distance: prune right subtree

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

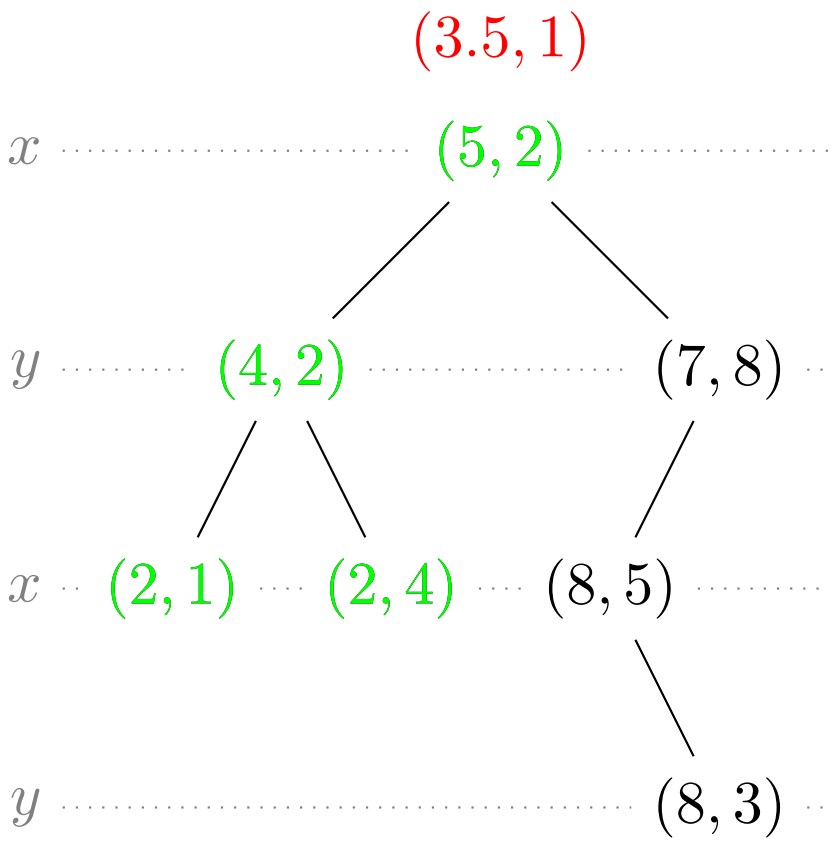


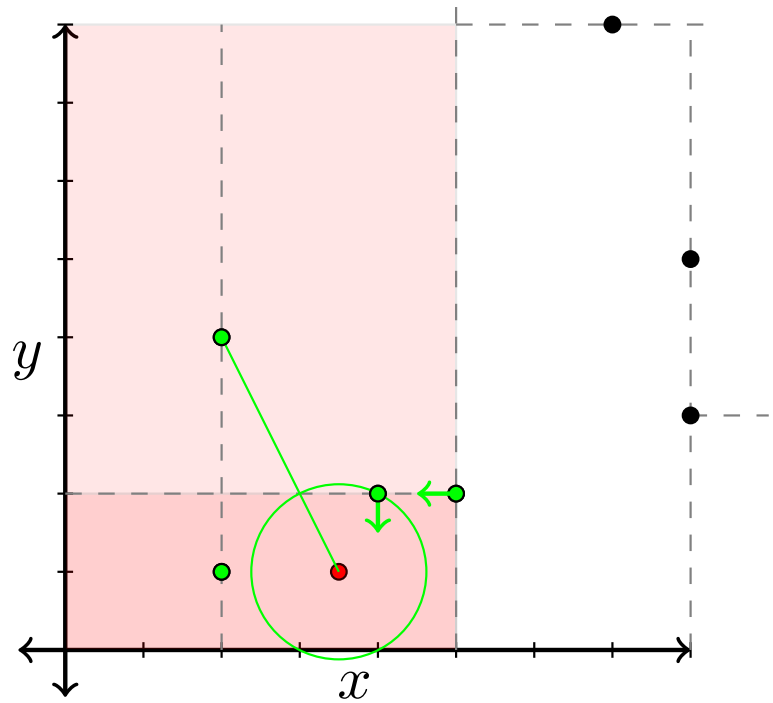


Do not change minimum distance

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3)

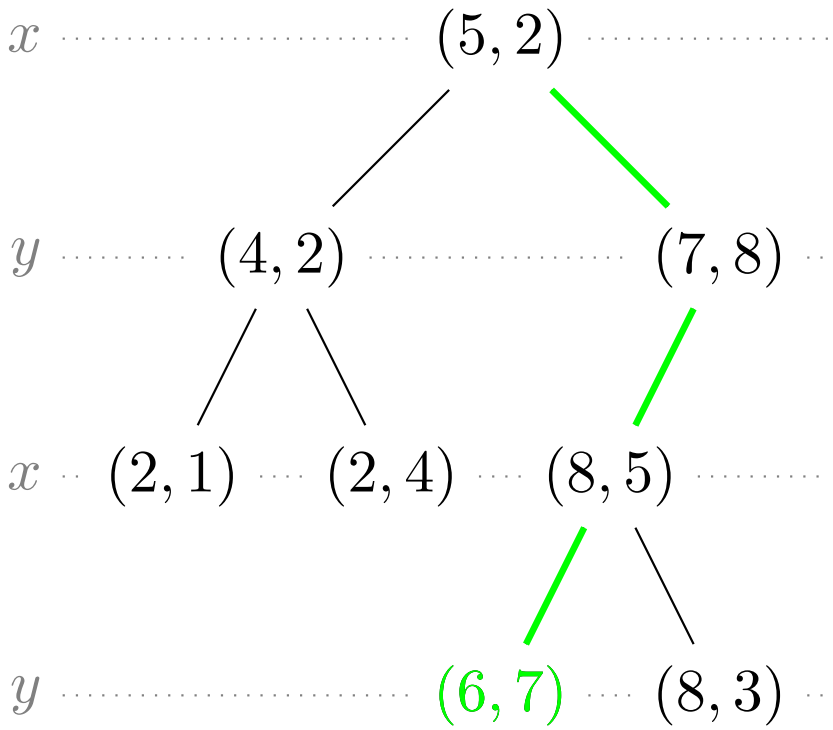


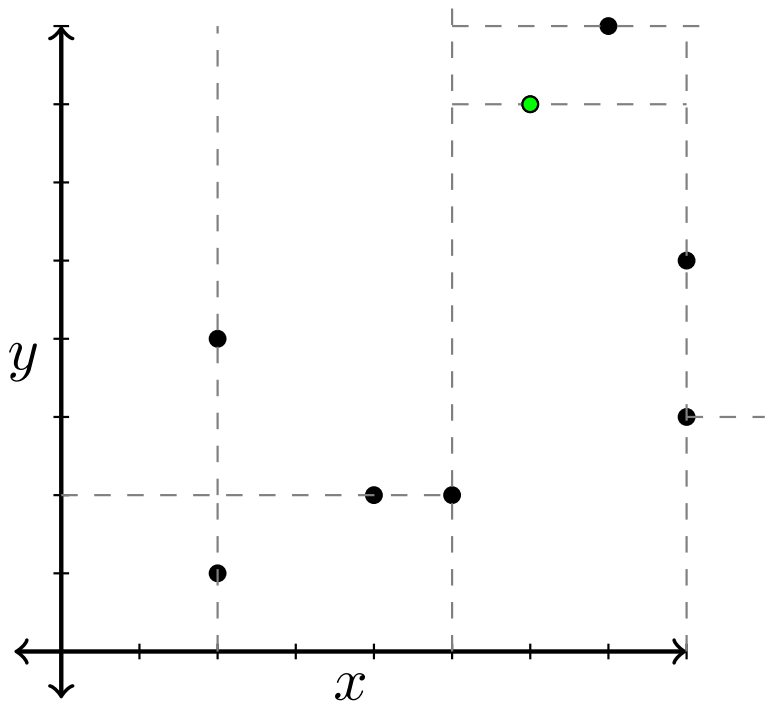


Do not change minimum distance

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)

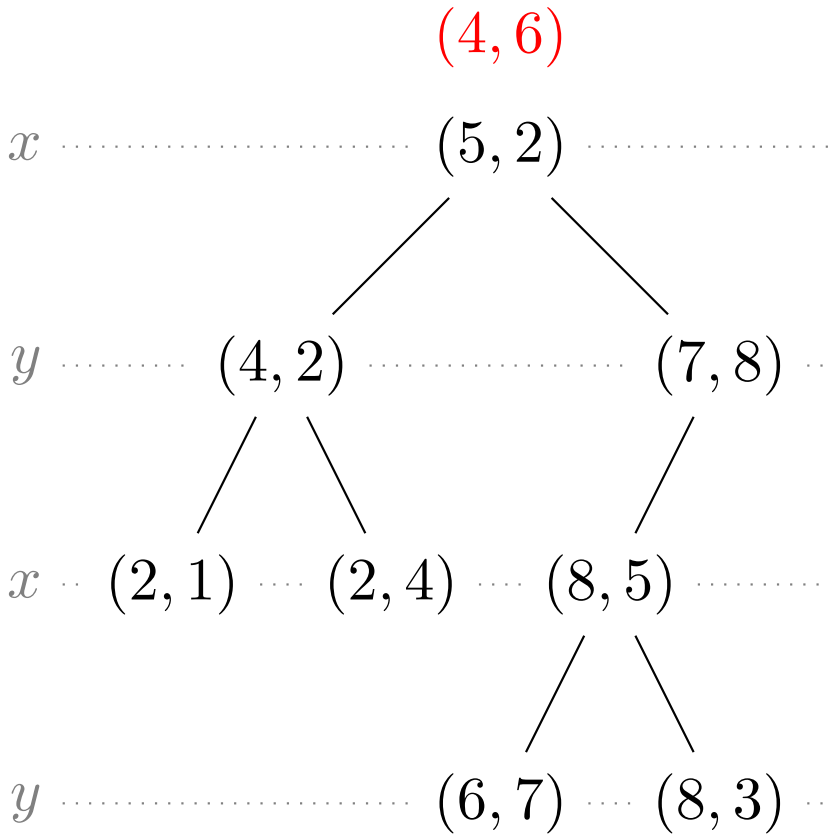


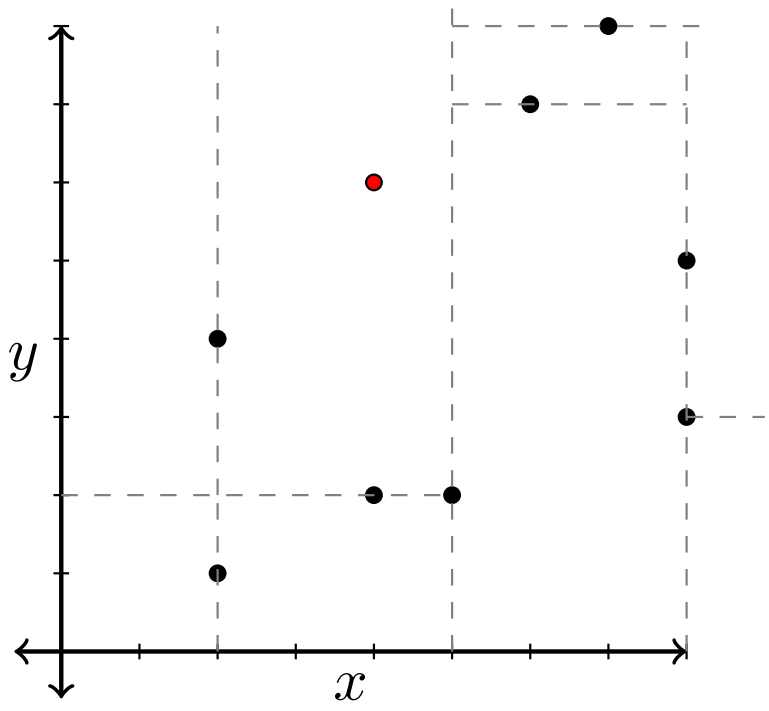


Consider new point (6, 7)

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)

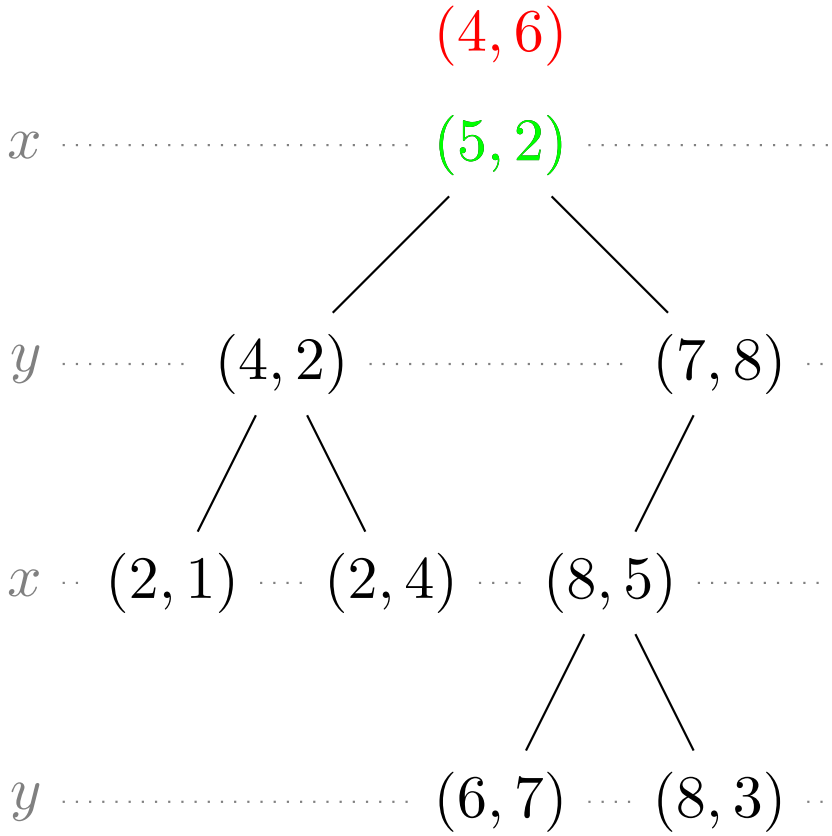


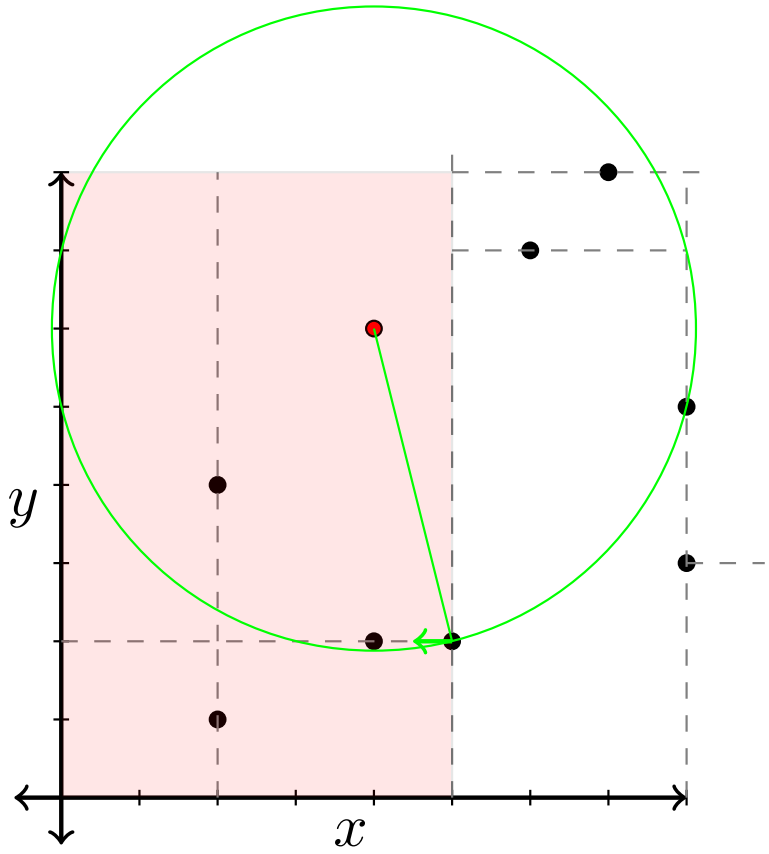


Find nearest neighbor to (4, 6)

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)

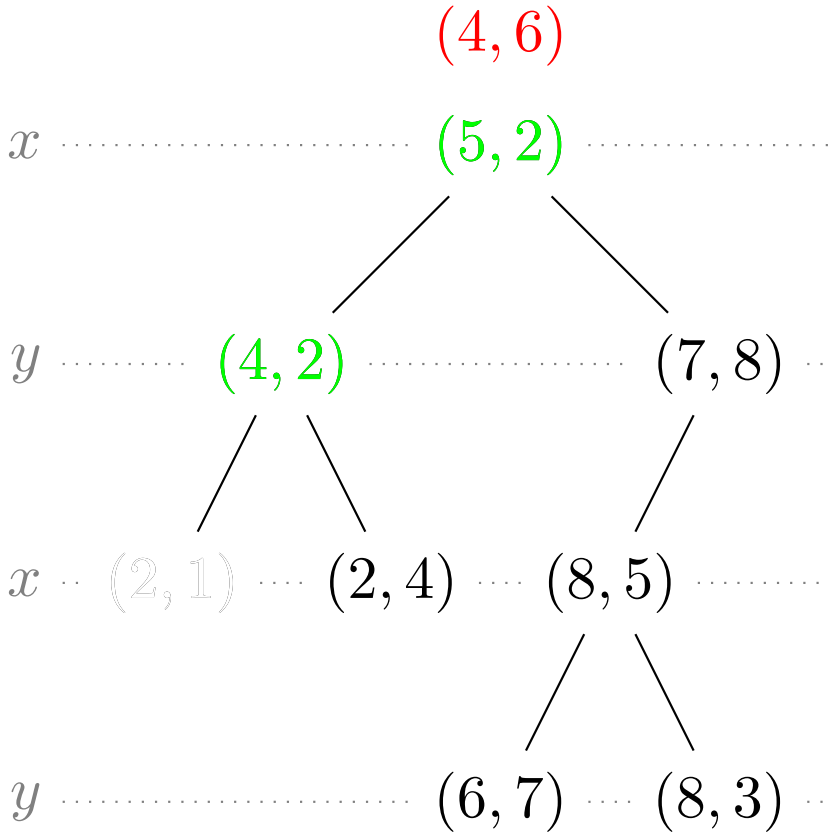


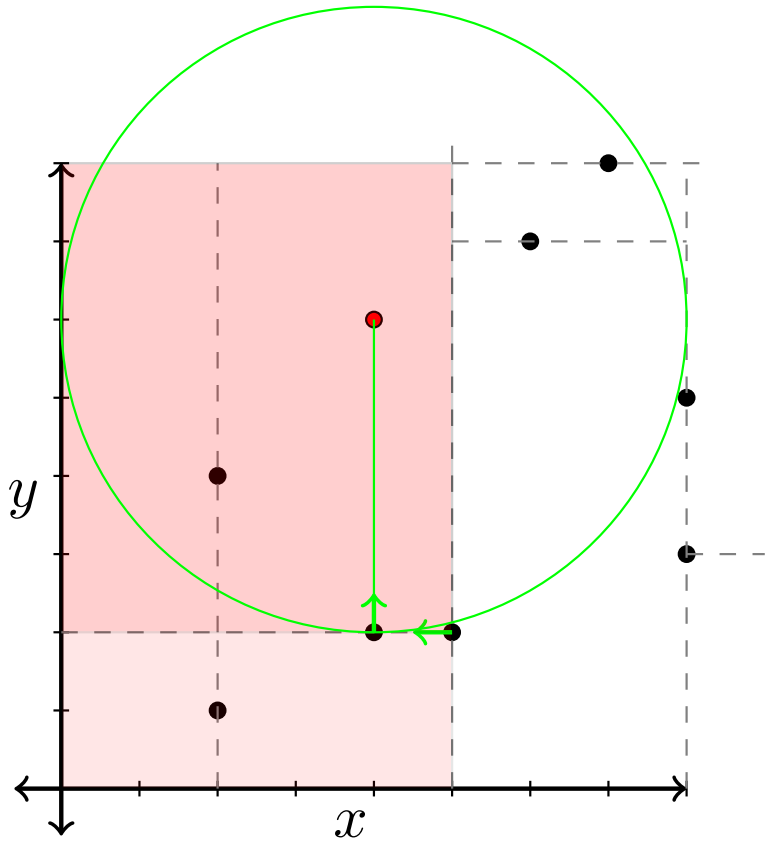


Set minimum distance: cannot prune

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)

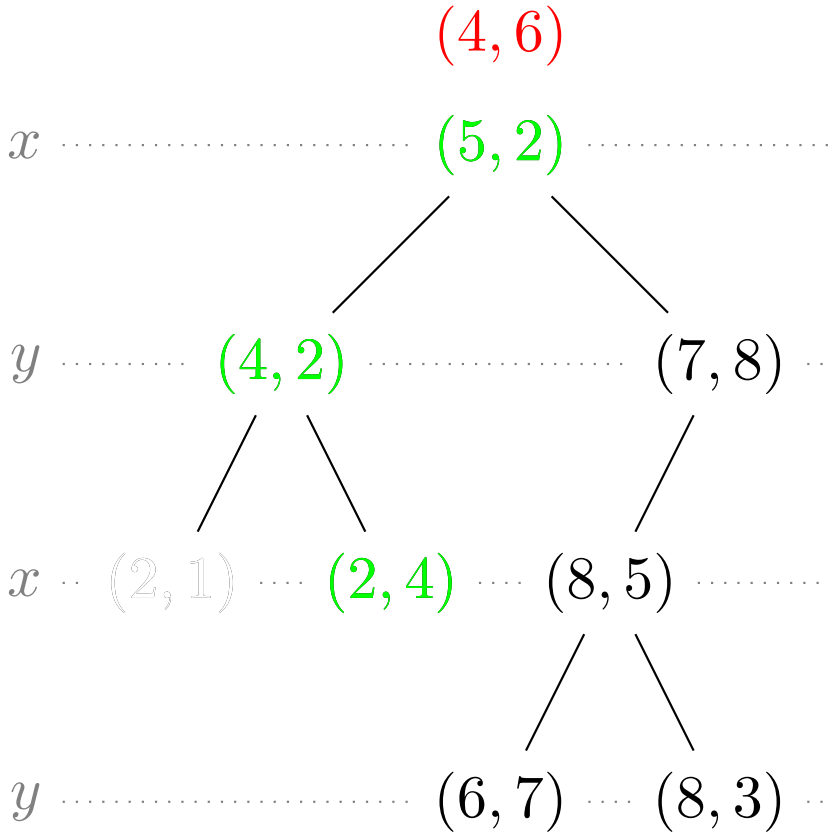


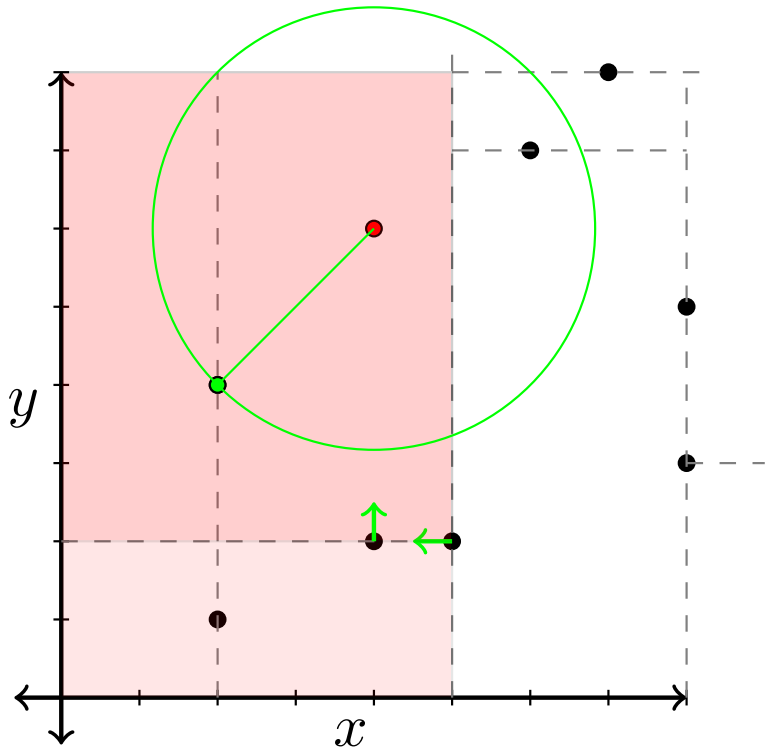


Set new minimum distance: prune

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)

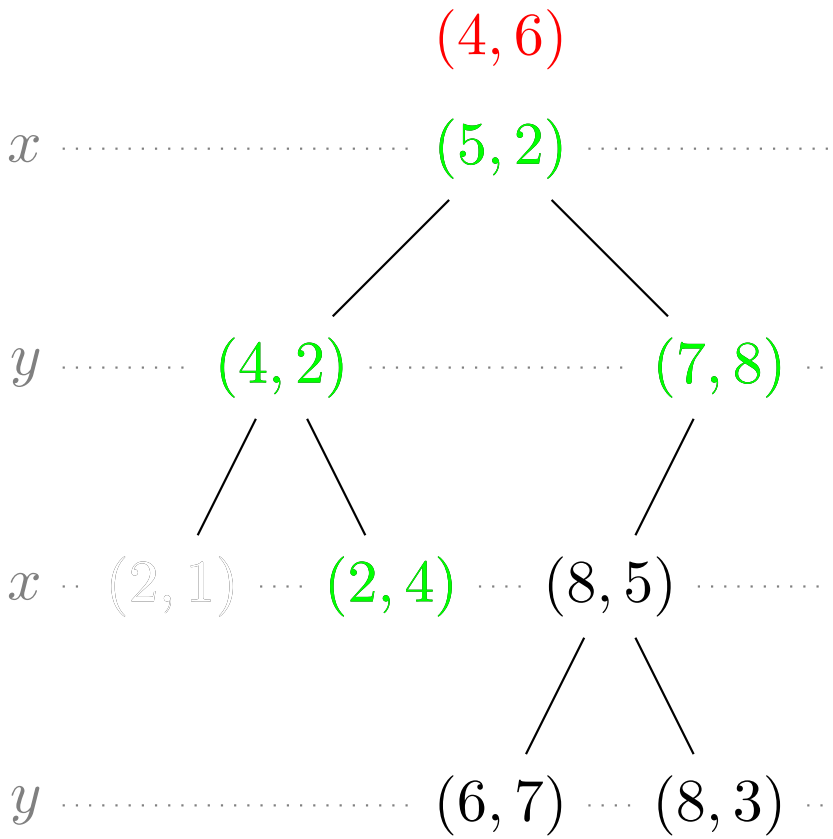


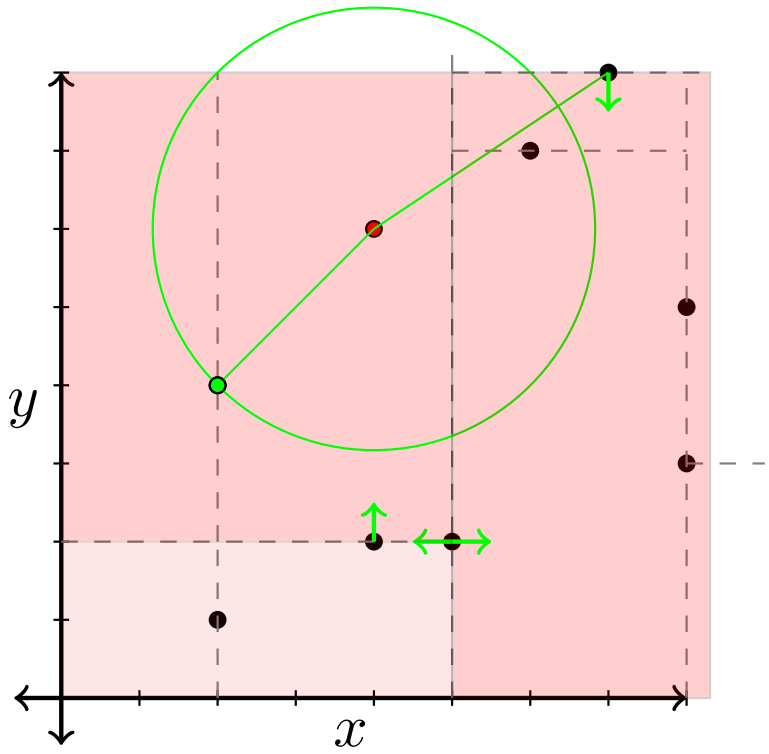


Set new minimum distance: cannot prune

K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)

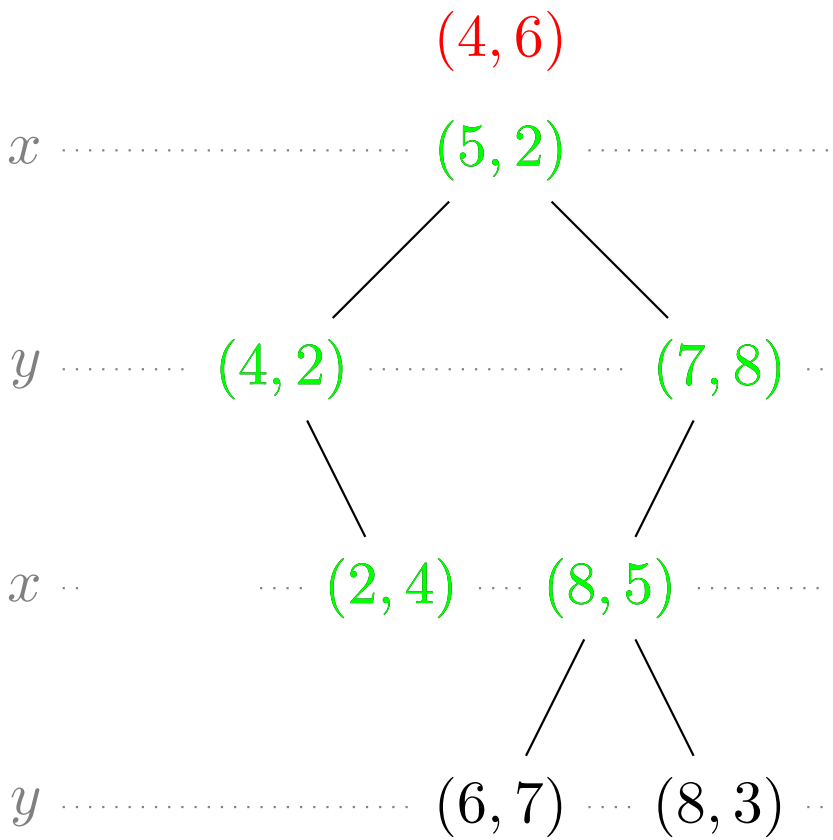


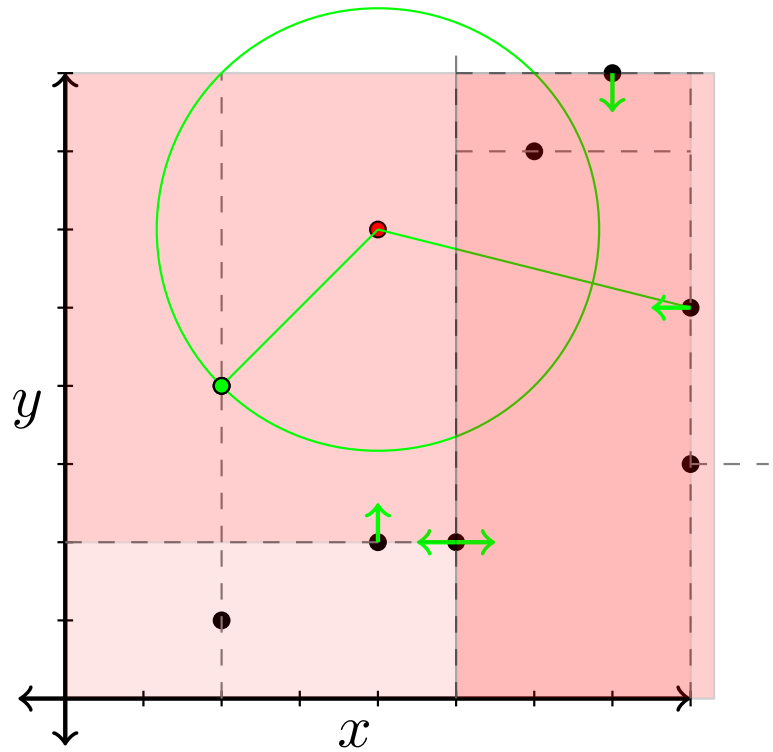


Traverse other side of tree

K-Dimensional Trees: Nearest Neighbor

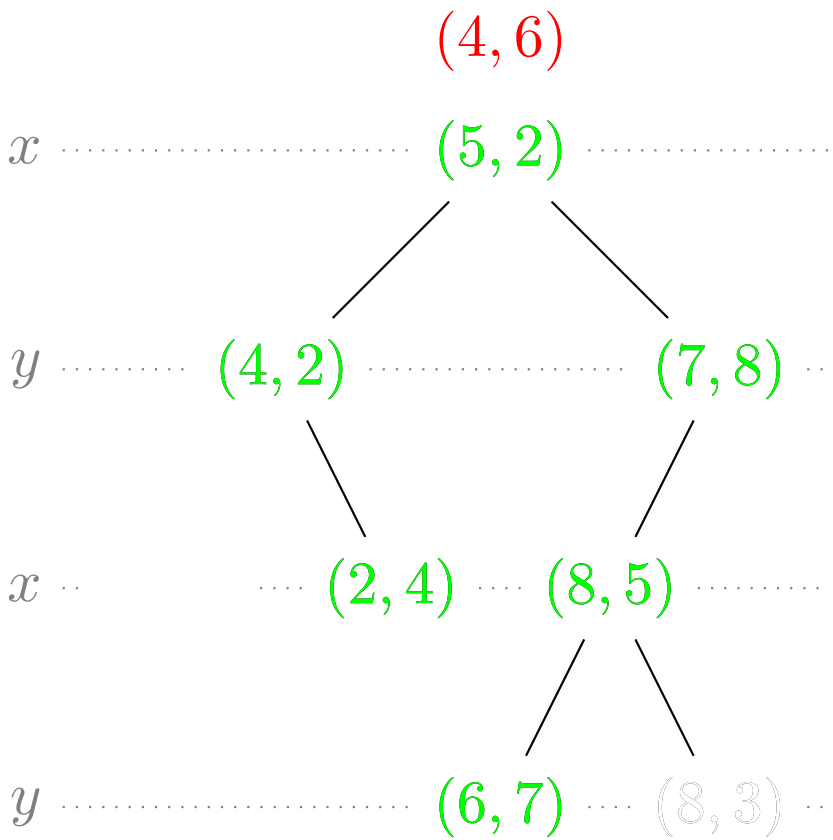
Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)

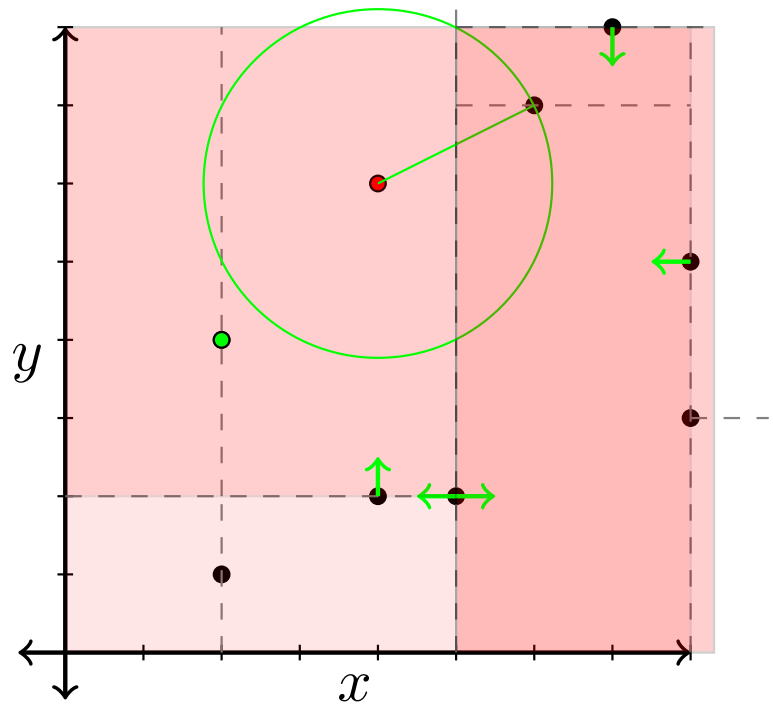




K-Dimensional Trees: Nearest Neighbor

Data: (5, 2), (7, 8), (8, 5), (4, 2), (2, 1), (2, 4), (8, 3), (6, 7)





Conclusion

- arrays are very powerful, but have limitations
- must think about what is important to our task to determine which data structure is optimal
- data structures are diverse and there are many ways to organize data to achieve our goals