



# Introduction to Scientific Computation

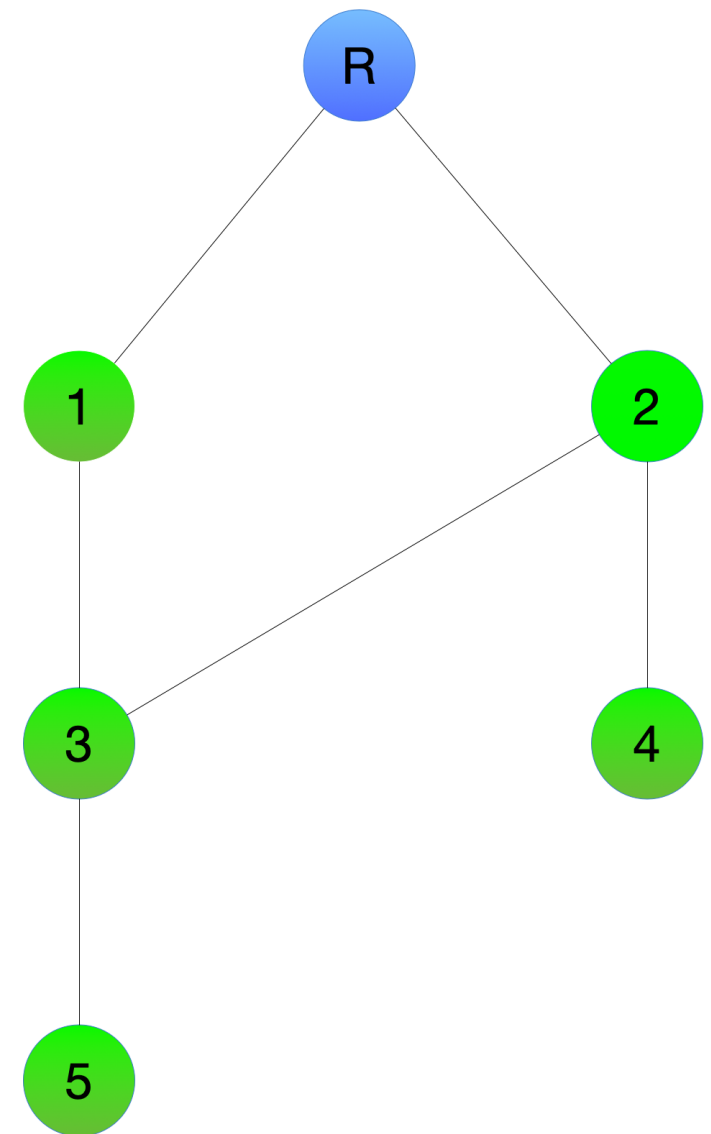
## Lecture 6

### Fall 2018

Graphs algorithms

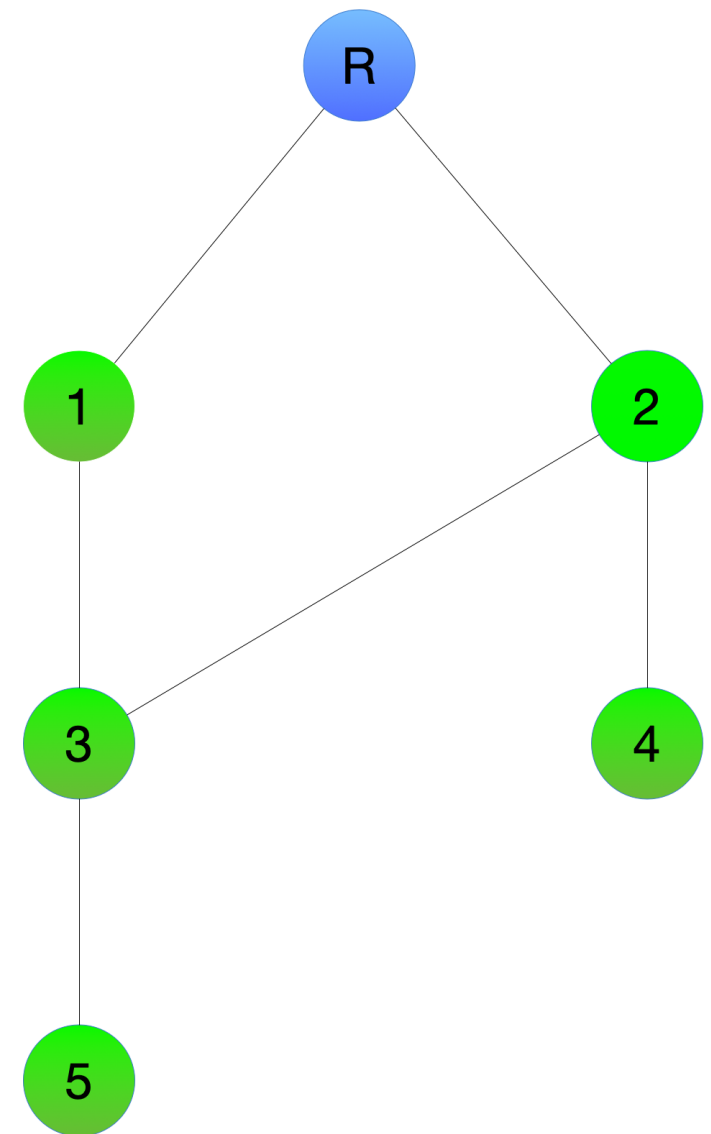
## Graph traversals

- Breadth-First Search (BFS)
- Depth-First Search (DFS)



## Breadth-First Search (BFS)

**Main Idea:** mark each vertex when we **first** visit it & monitor completely unexplored items.



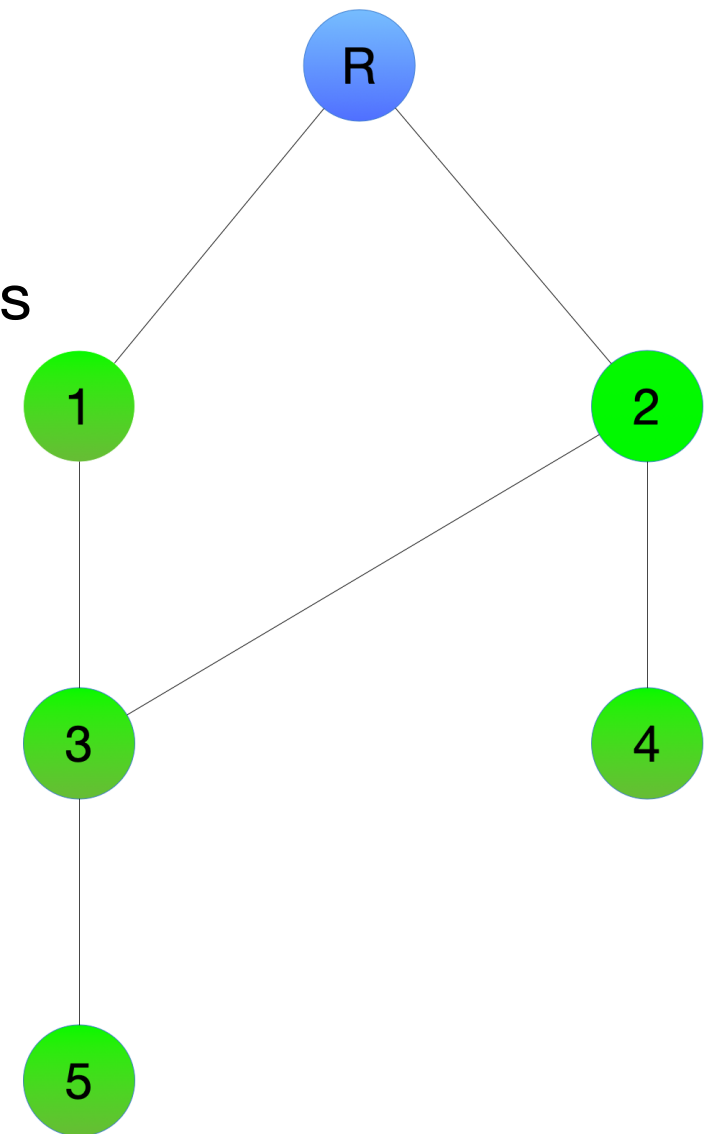
## Breadth-First Search (BFS)

**Main Idea:** mark each vertex when we **first** visit it & monitor completely unexplored items.

Each vertex can exist in one of three states:

- Undiscovered - the node has never been visited
- Discovered - the node has been found, but not all its edges are visited
- Processed - all adjacent edges of the node are visited

Container to be used - queue.





## Breadth-First Search (BFS)

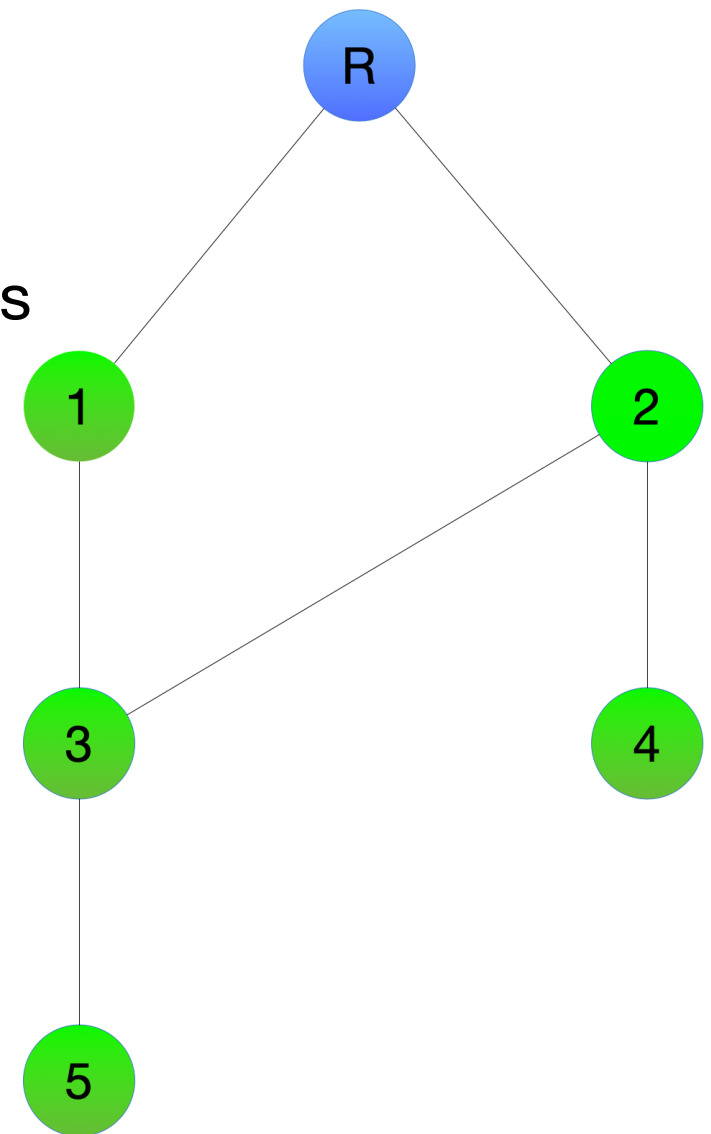
**Main Idea:** mark each vertex when we **first** visit it & monitor completely unexplored items.

Each vertex can exist in one of three states:

- Undiscovered - the node has never been visited
- Discovered - the node has been found, but not all its edges are visited
- Processed - all adjacent edges of the node are visited

Container to be used - queue.

**Quiz:** What is the complexity of BFS ?



## Breadth-First Search (BFS)

**Main Idea:** mark each vertex when we **first** visit it & monitor completely unexplored items.

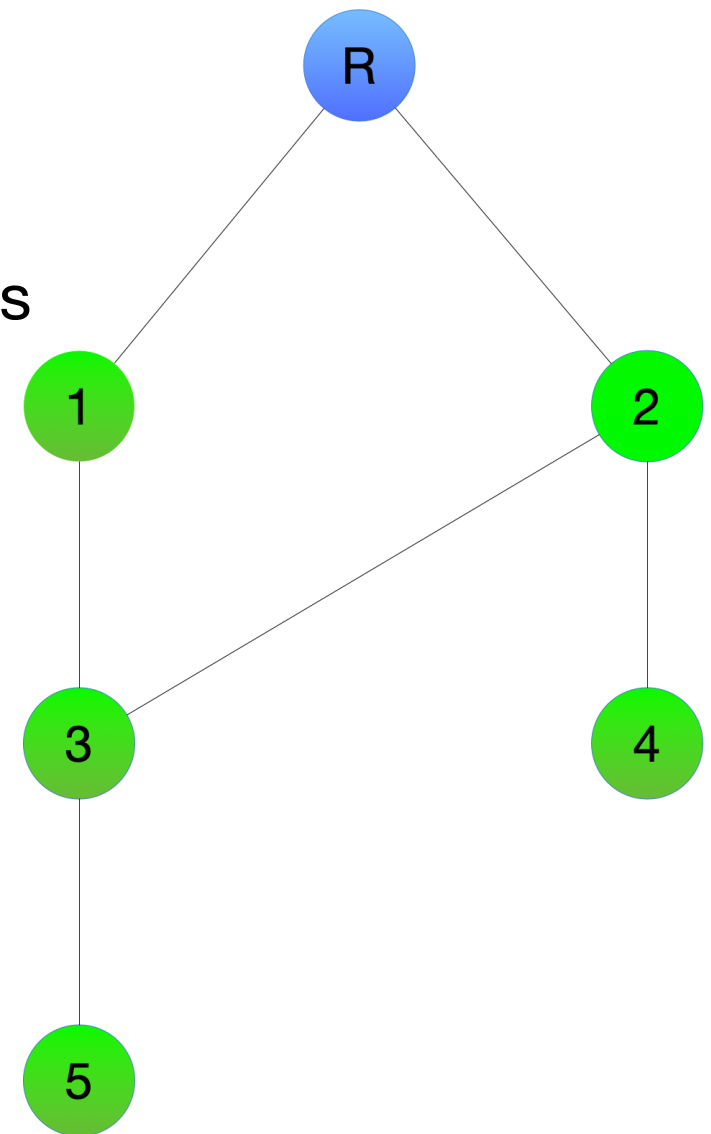
Each vertex can exist in one of three states:

- Undiscovered - the node has never been visited
- Discovered - the node has been found, but not all its edges are visited
- Processed - all adjacent edges of the node are visited

Container to be used - queue.

**Quiz:** What is the complexity of BFS ?

**Answer:**  $O(V + E)$



## Depth-First Search (DFS)

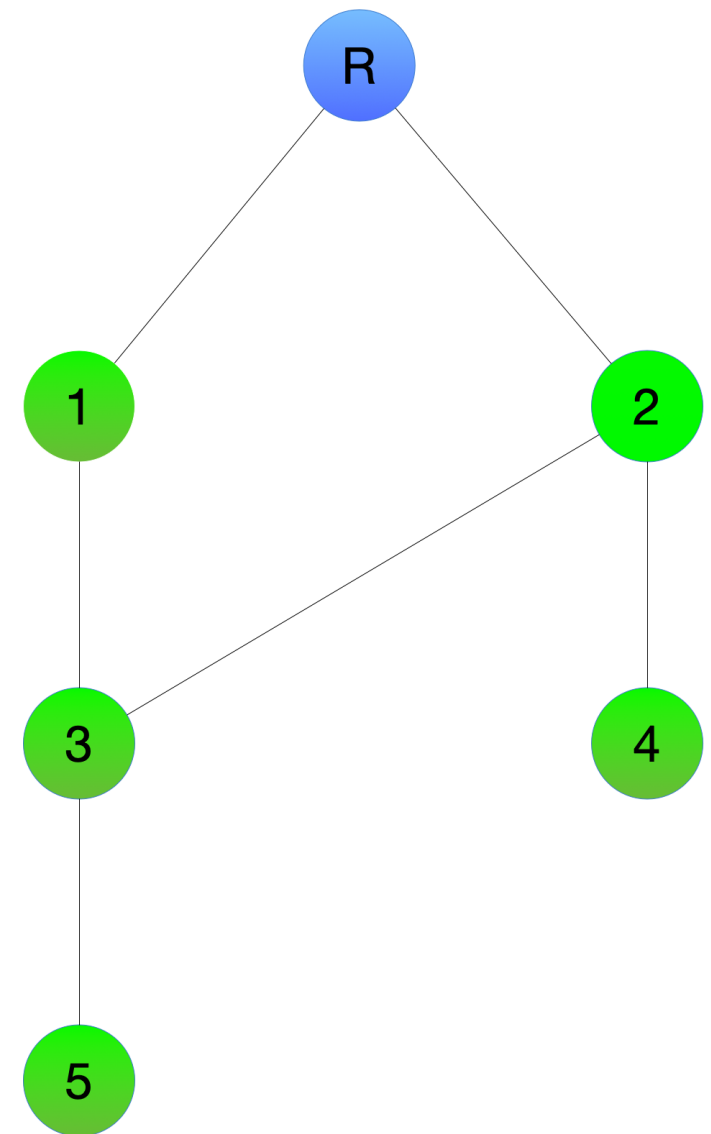
**Main Idea:** start DFS from every next node until there are nodes which are not visited.

Each vertex can exist in one of two states:

- Undiscovered - the node has never been visited
- Discovered - the node has been visited

Container to be used - stack (recursion).

**Quiz:** What is the complexity of DFS ?



## Depth-First Search (DFS)

**Main Idea:** start DFS from every next node until there are nodes which are not visited.

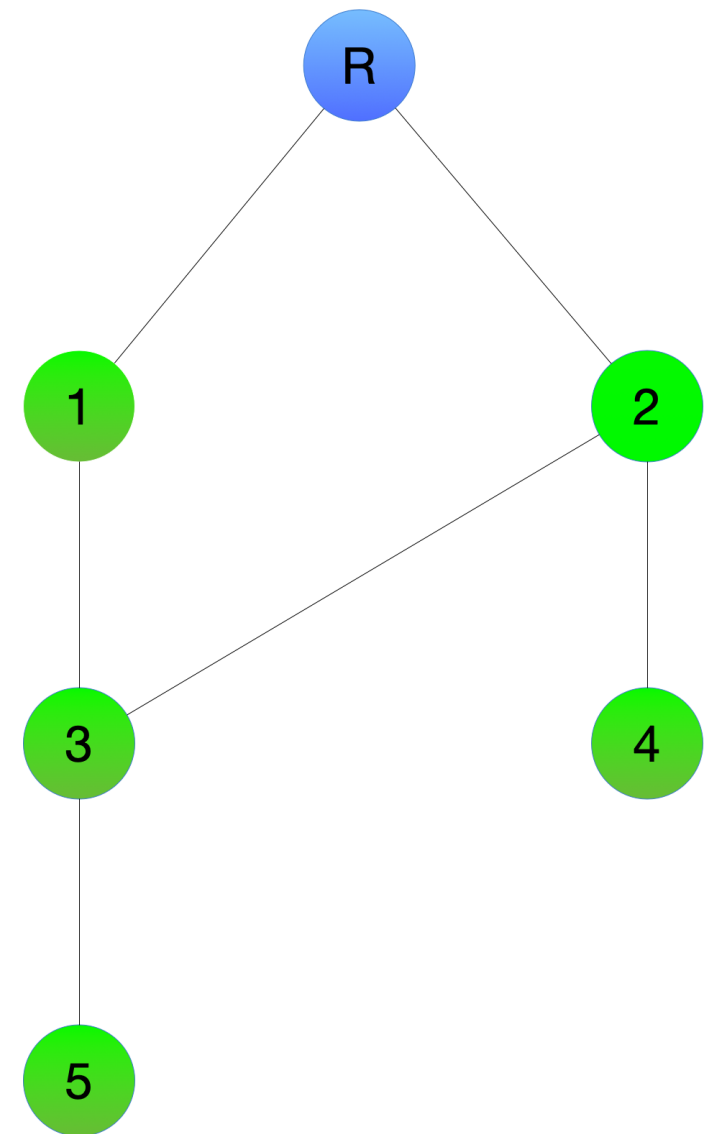
Each vertex can exist in one of two states:

- Undiscovered - the node has never been visited
- Discovered - the node has been visited

Container to be used - stack (recursion).

**Quiz:** What is the complexity of DFS ?

**Answer:**  $O(V + E)$

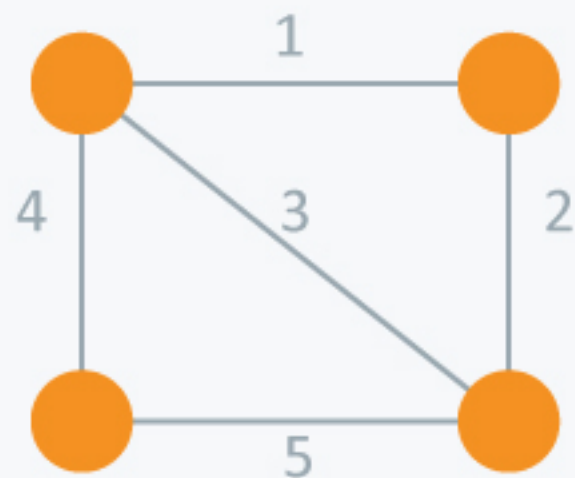




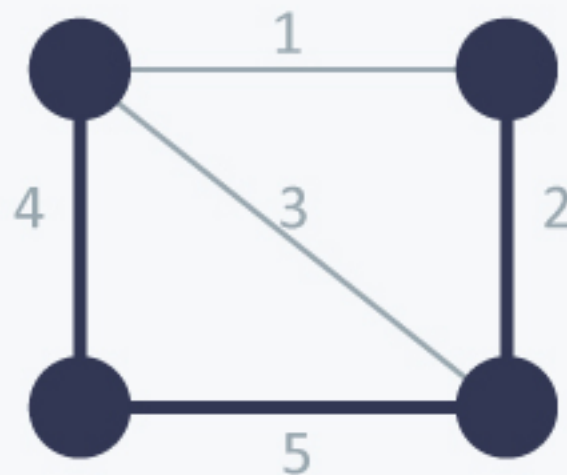
## Minimum Spanning Tree

**Spanning tree:** for graph  $G = (V, E)$  is a subset of edges from  $E$ , forming a tree connecting all vertices of  $V$ .

For edge-weighted graph we are usually interested in minimum value spanning tree

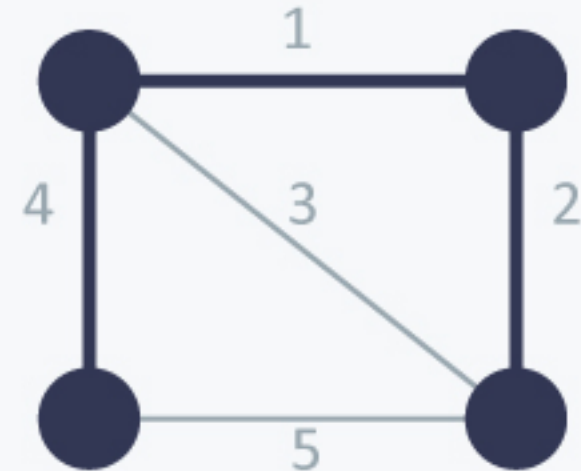


Undirected  
Graph



Spanning  
Tree

Cost = 11(=4+5+2)



Minimum Spanning  
Tree

Cost = 7(=4+1+2)

## Minimum Spanning Tree

**Spanning tree:** for graph  $G = (V, E)$  is a subset of edges from  $E$ , forming a tree connecting all vertices of  $V$ .

For edge-weighted graph we are usually interested in minimum value spanning tree

## Minimum Spanning Tree

**Spanning tree:** for graph  $G = (V, E)$  is a subset of edges from  $E$ , forming a tree connecting all vertices of  $V$ .

For edge-weighted graph we are usually interested in minimum value spanning tree

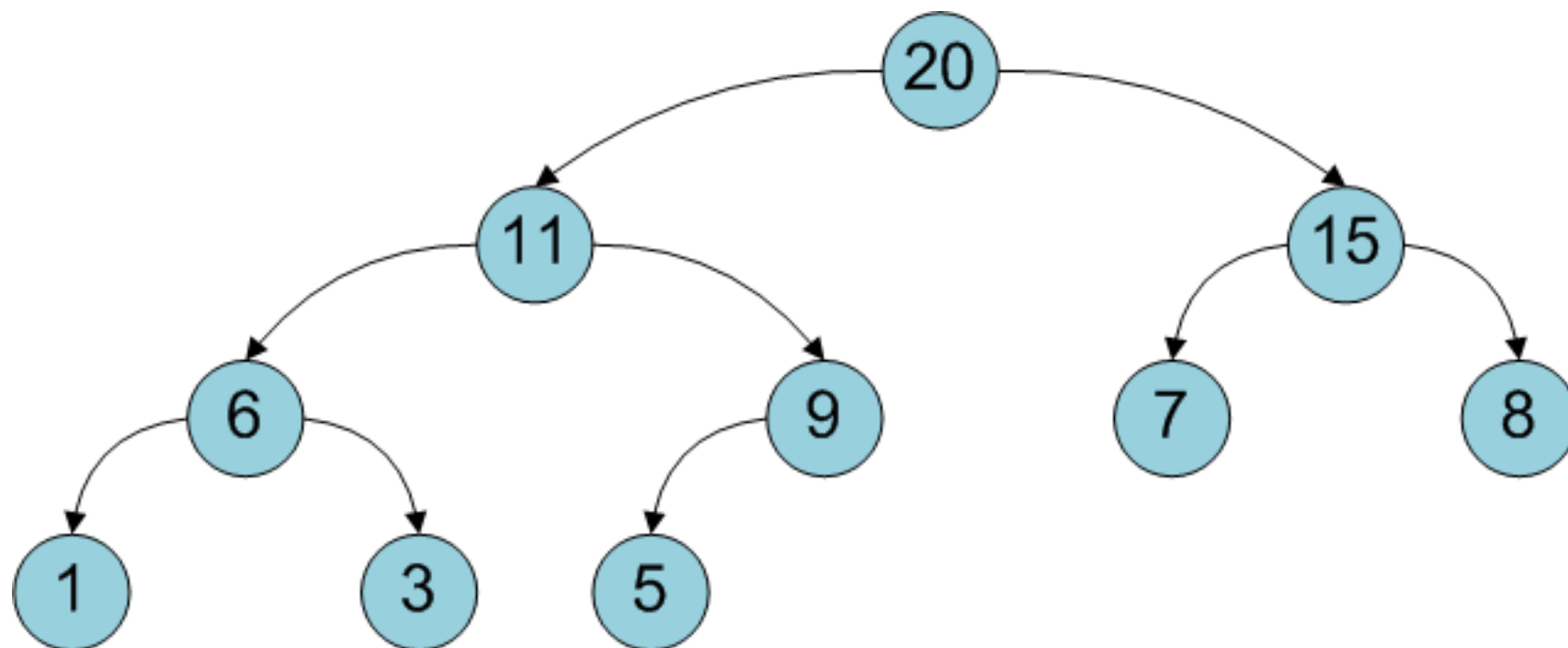
Two main algorithms:

- Prim
  - Kruskal
- 
- Minimum spanning tree minimises total length (weight) over all possible spanning trees.
  - There could be more than one minimum spanning tree in the graph.

Algorithms are greedy

## Heap

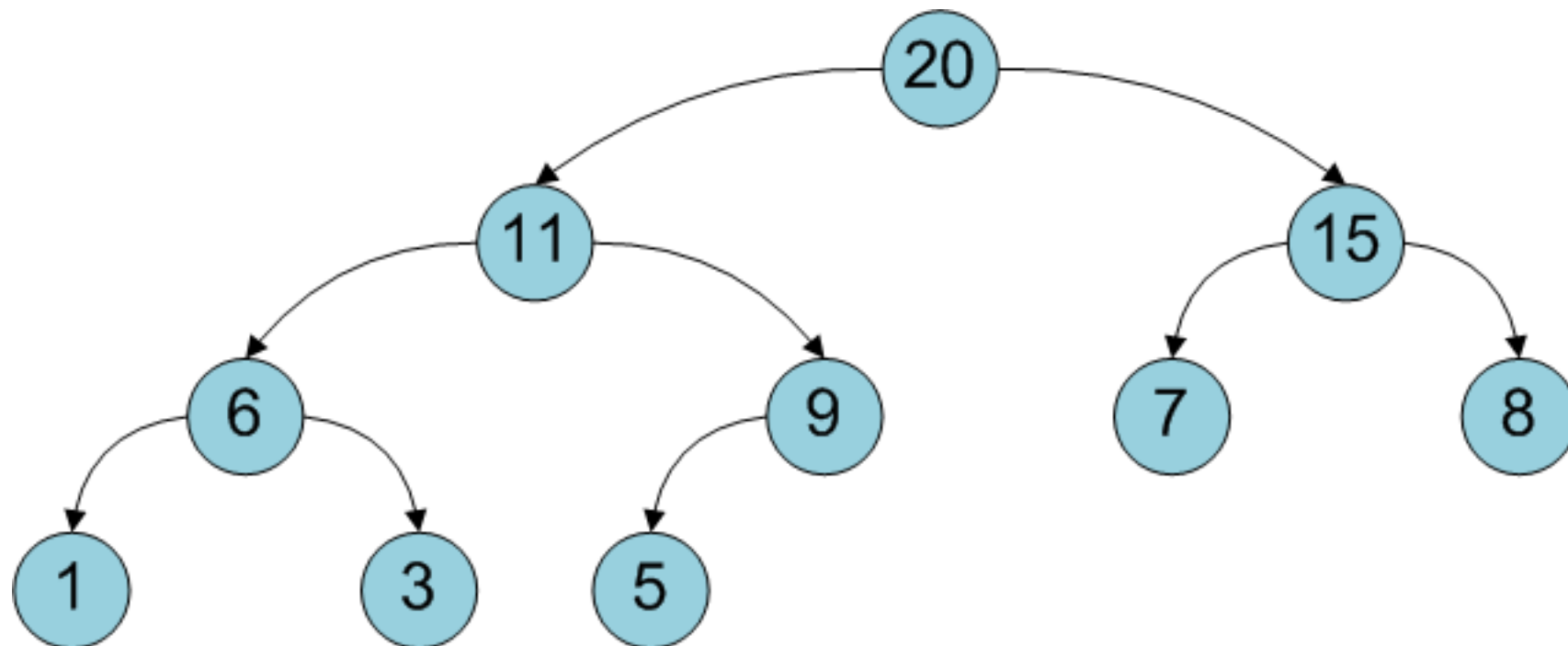
**Def:** is a binary tree with the property value of each vertex is less than the value of its ancestors



## Heap

**Def:** is a binary tree with the property value of each vertex is greater than the value of its ancestors

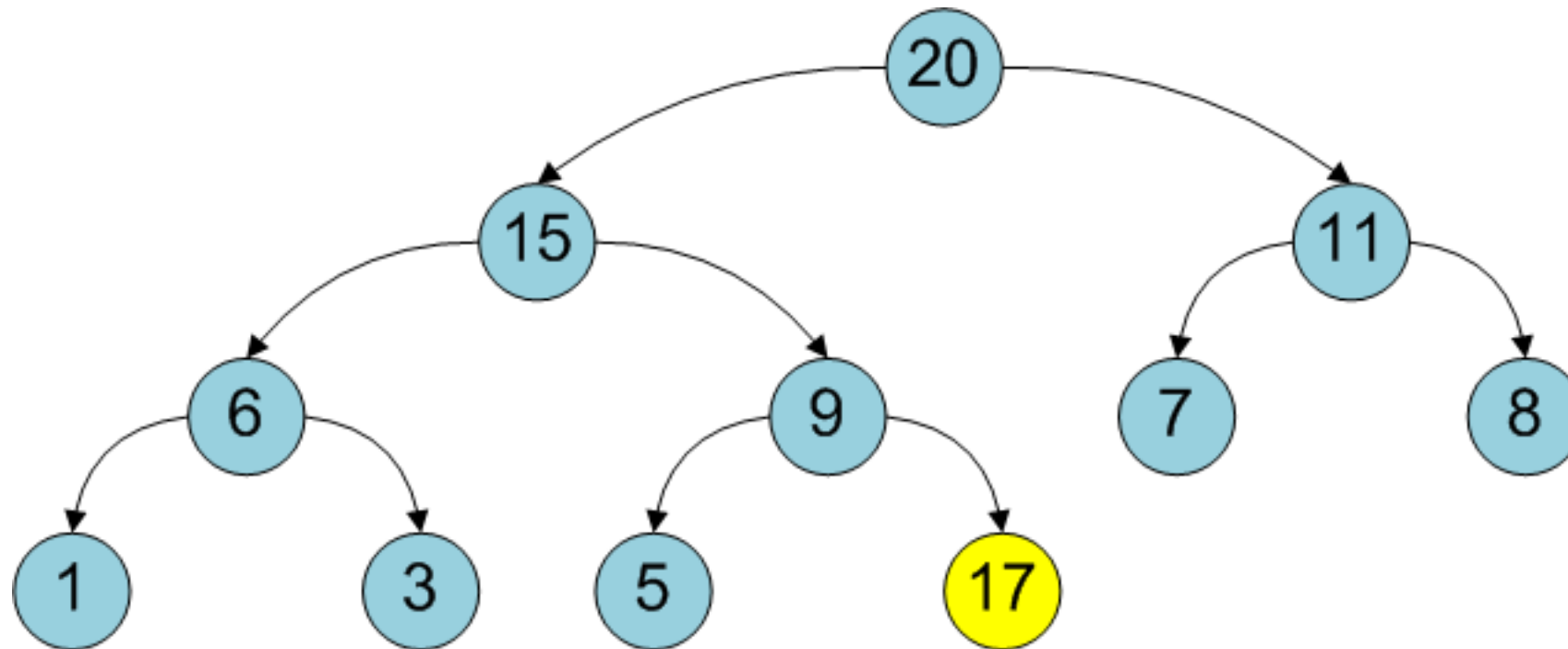
One can store entire data structure in an array. Root element is in index 0, left ancestor of vertex  $i$  would have an index  $2 * i + 1$ , while the right one would have an index  $2 * i + 2$



## Heap

### Push

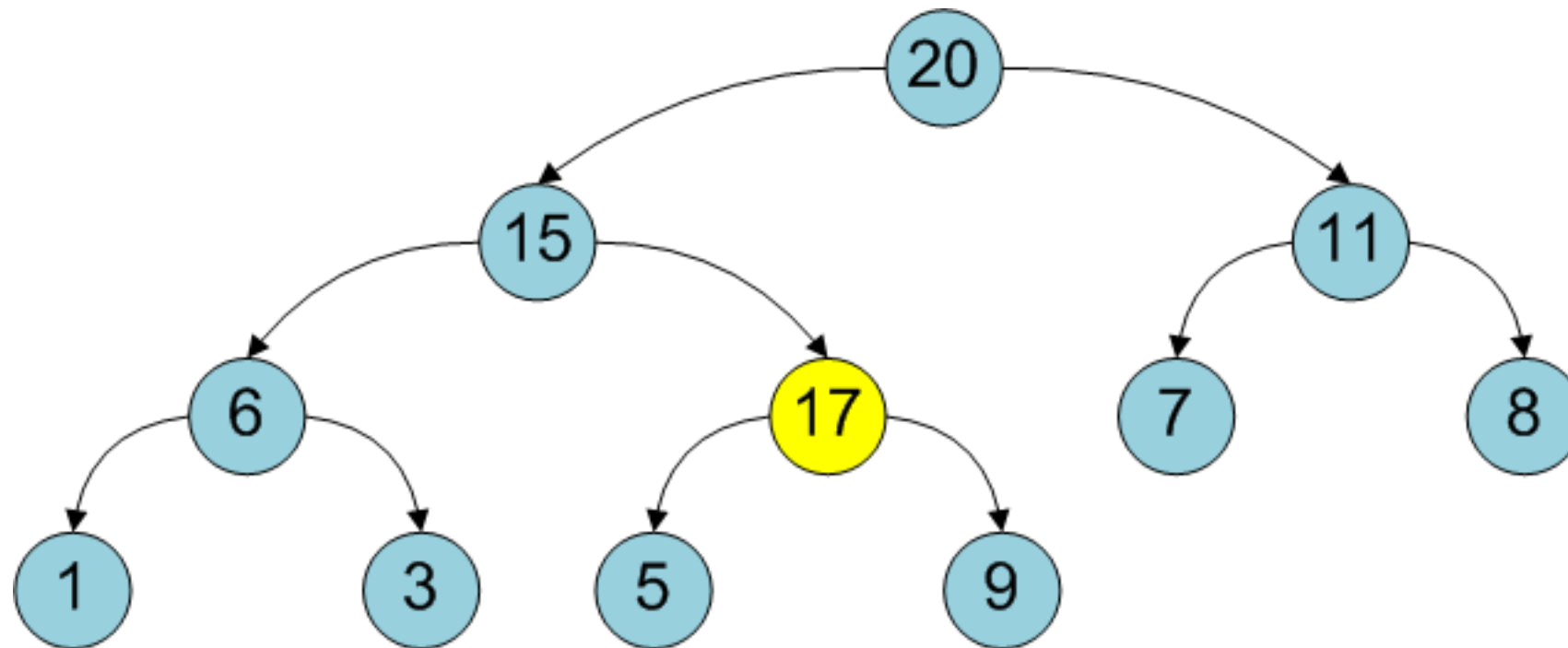
Add element to the end of the array. Then fix the property of the heap.





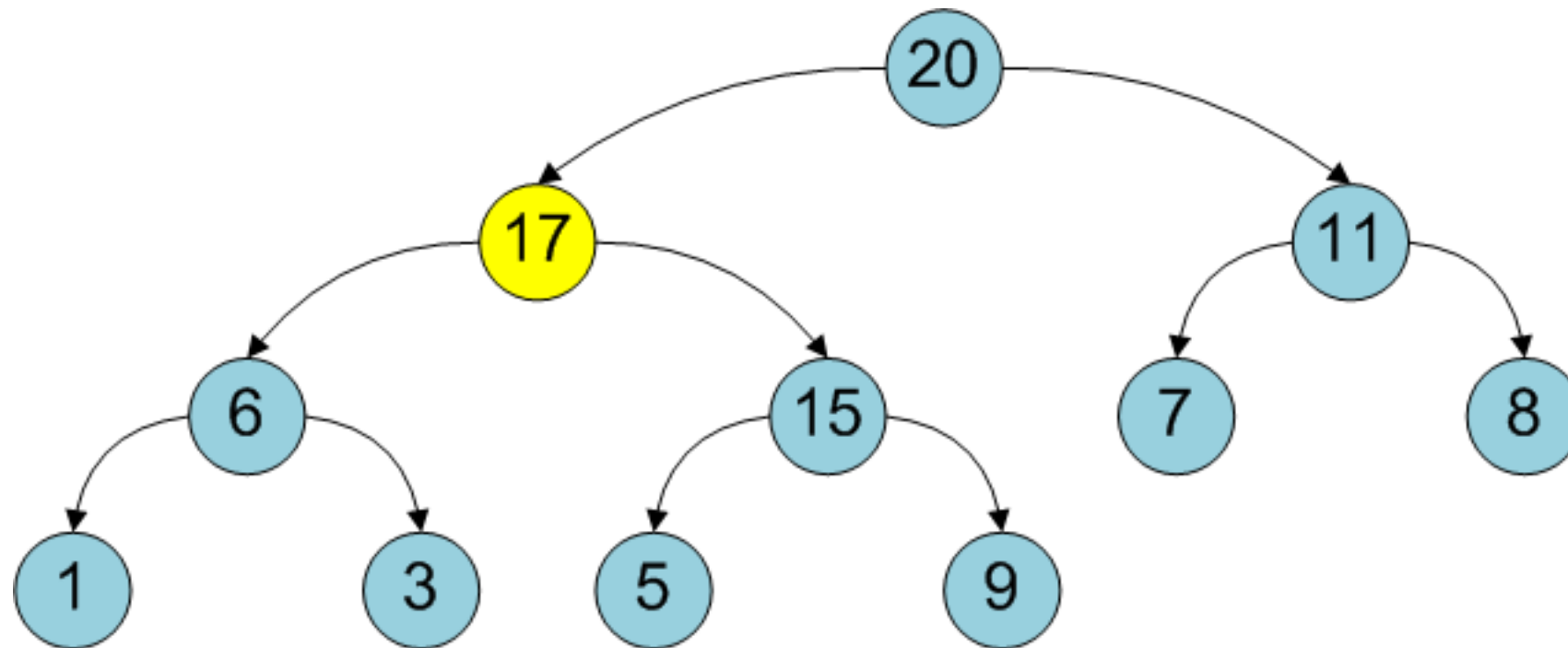
**Heap**

**Push**



Heap

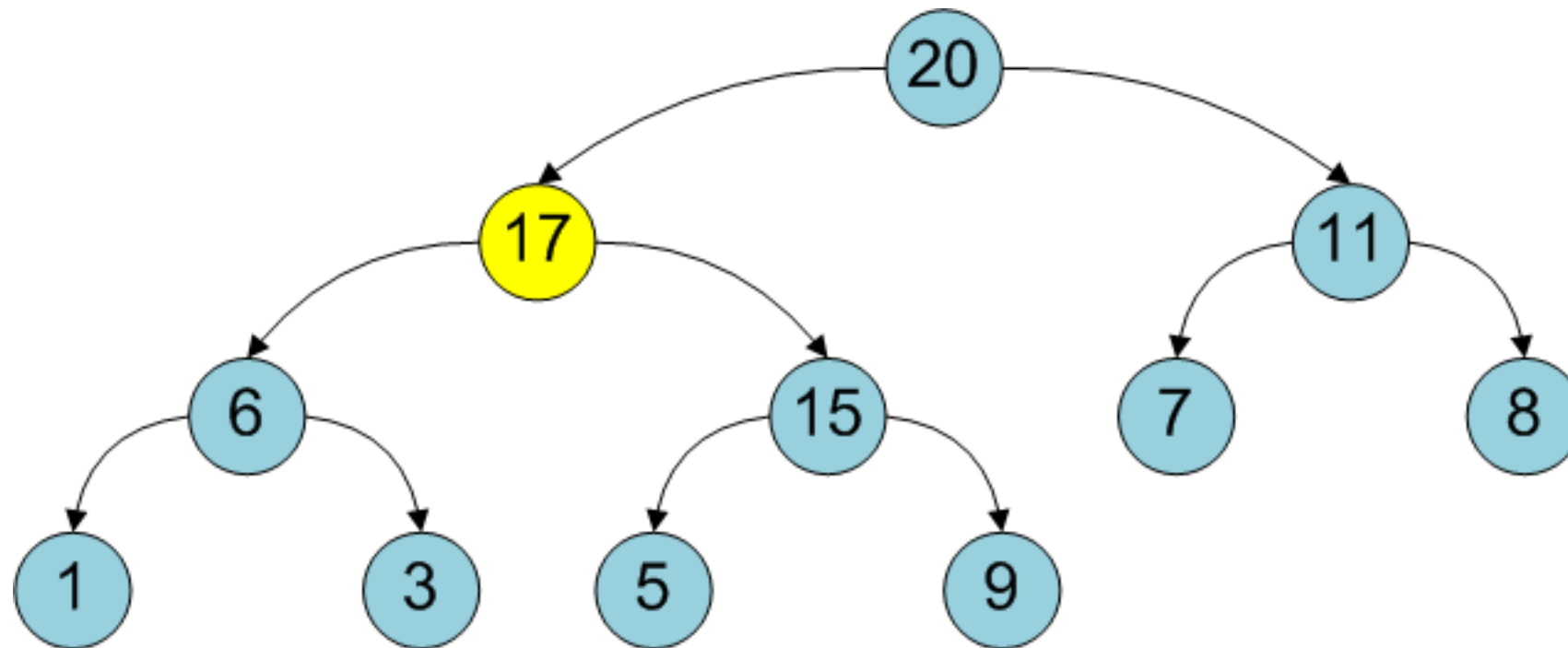
Push



**Quiz:** What is the complexity of adding an element in the heap ?

Heap

Push

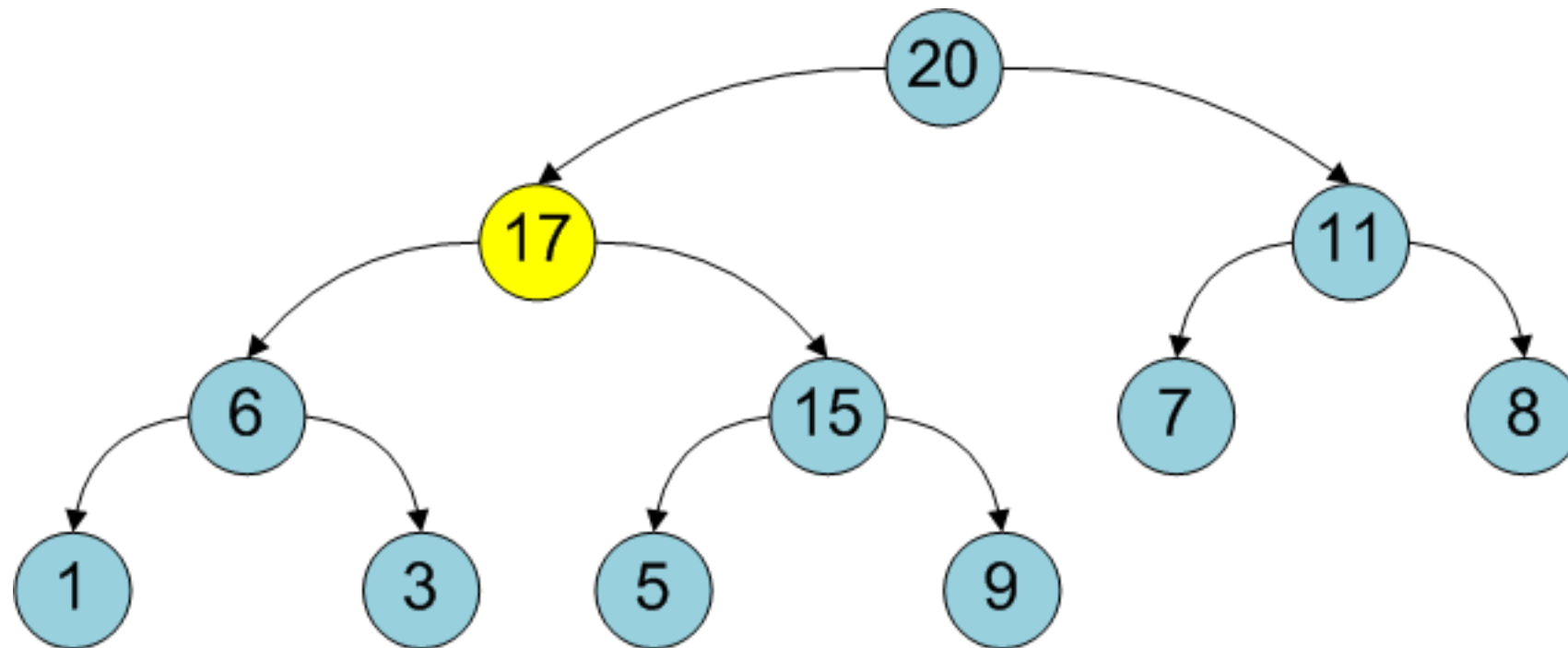


**Quiz:** What is the complexity of adding an element in the heap ?

**Answer:**  $O(\log(N))$

Heap

Push

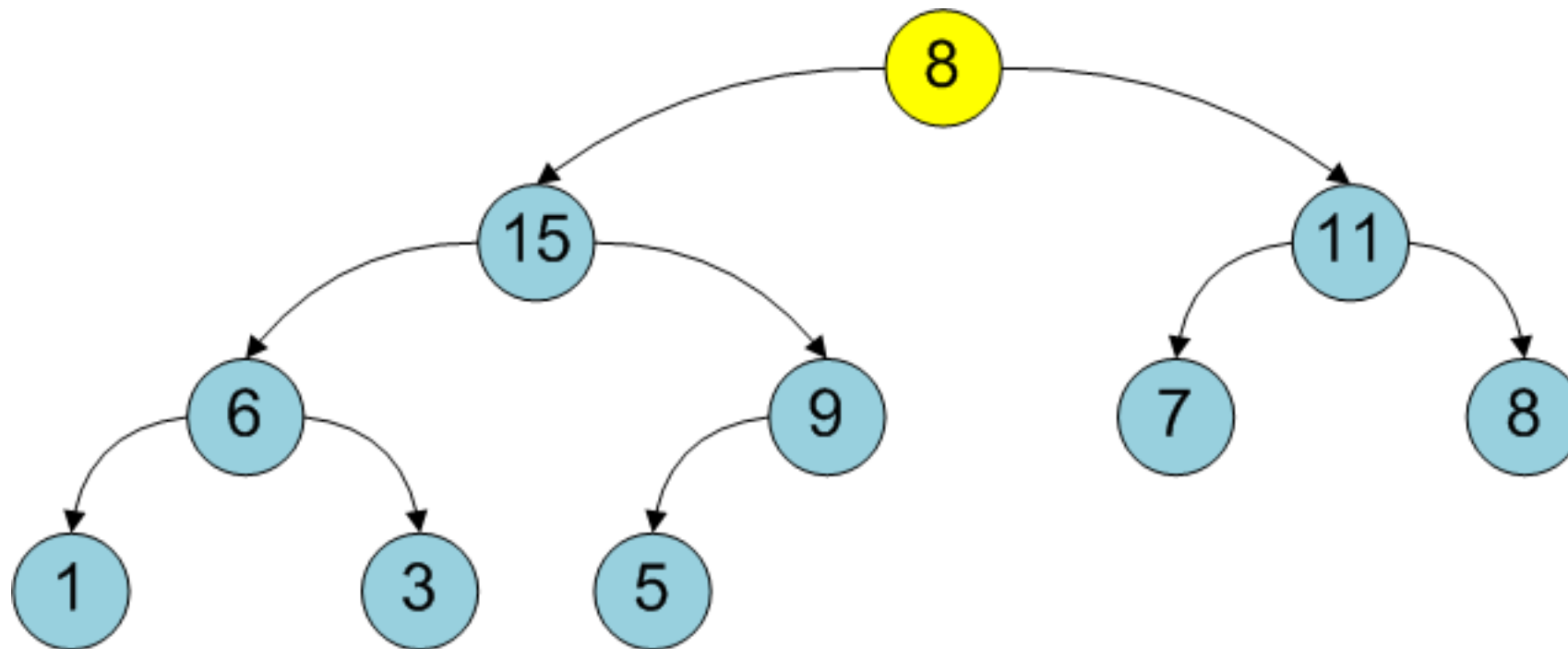


**Quiz:** What is the complexity of adding an element in the heap ?

**Answer:**  $O(\log(N))$

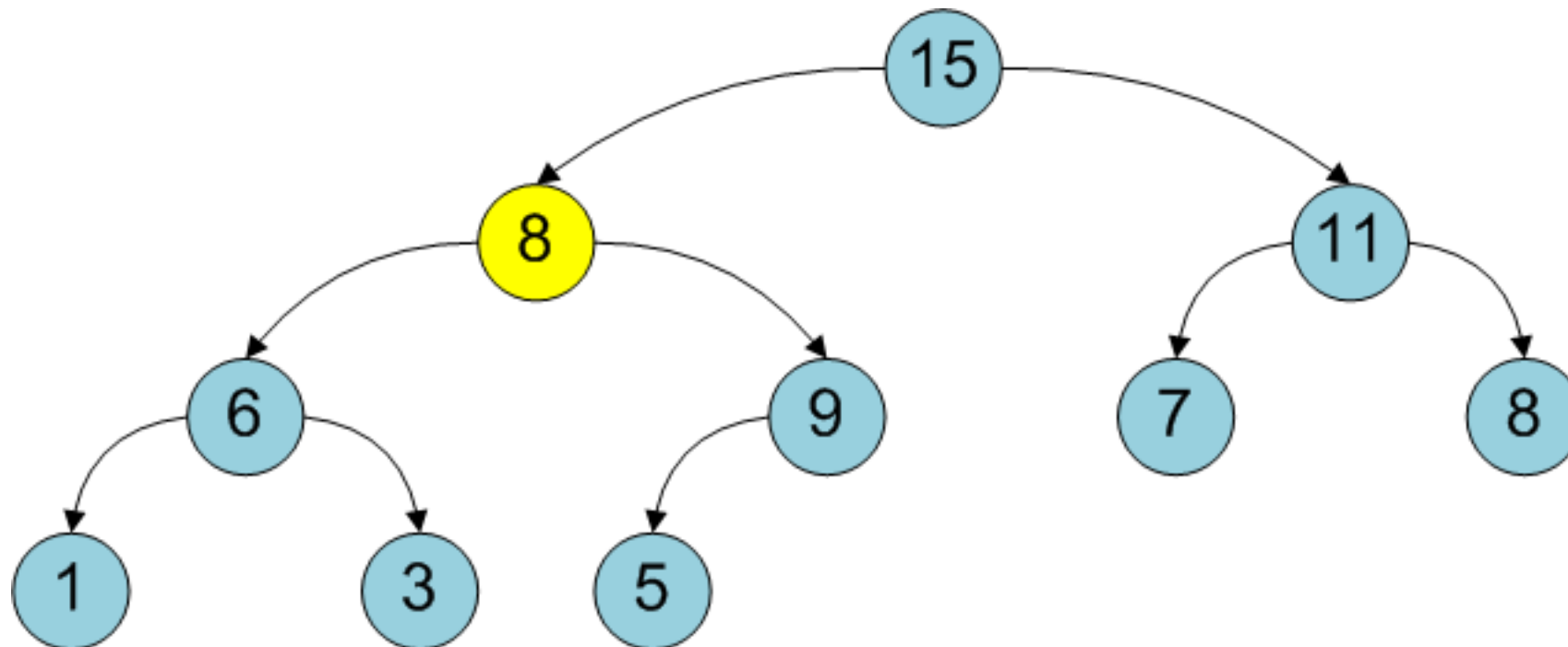
**Heap**

**Heapify**



**Heap**

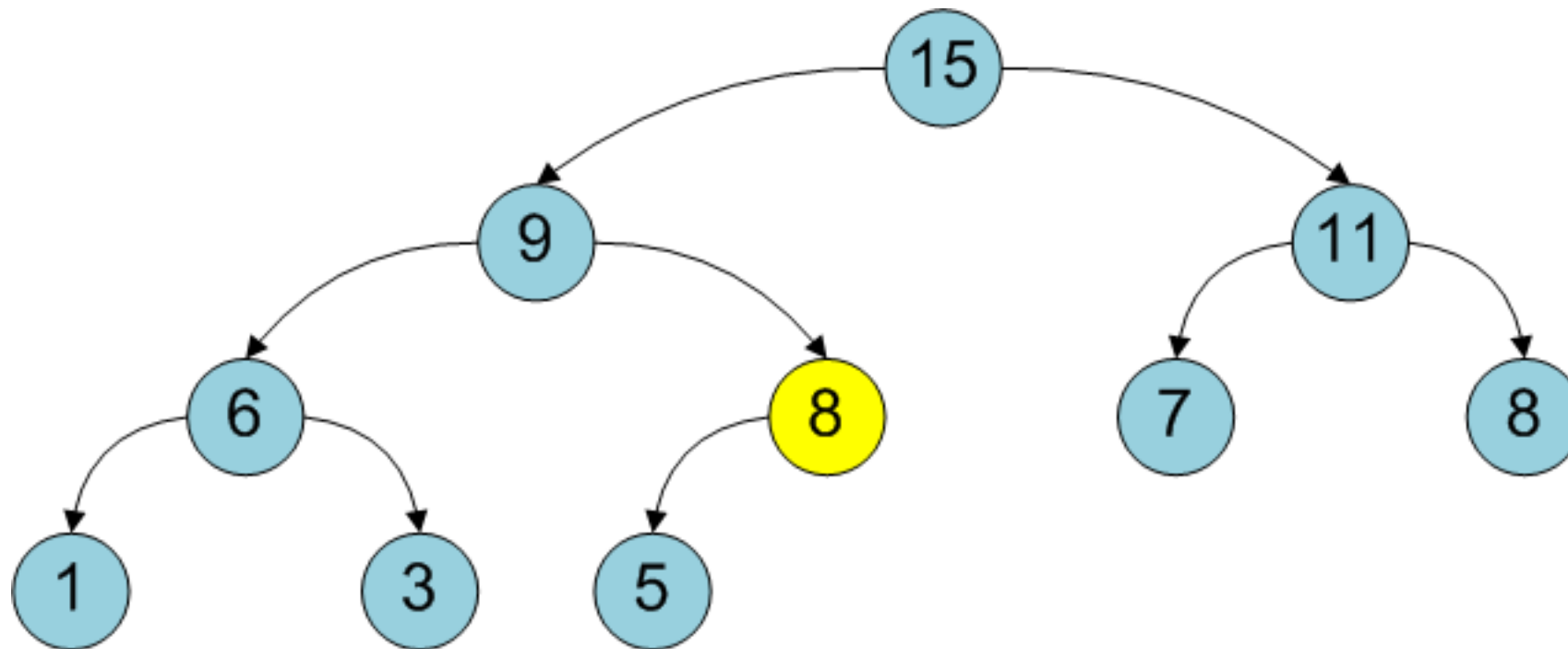
**Heapify**





**Heap**

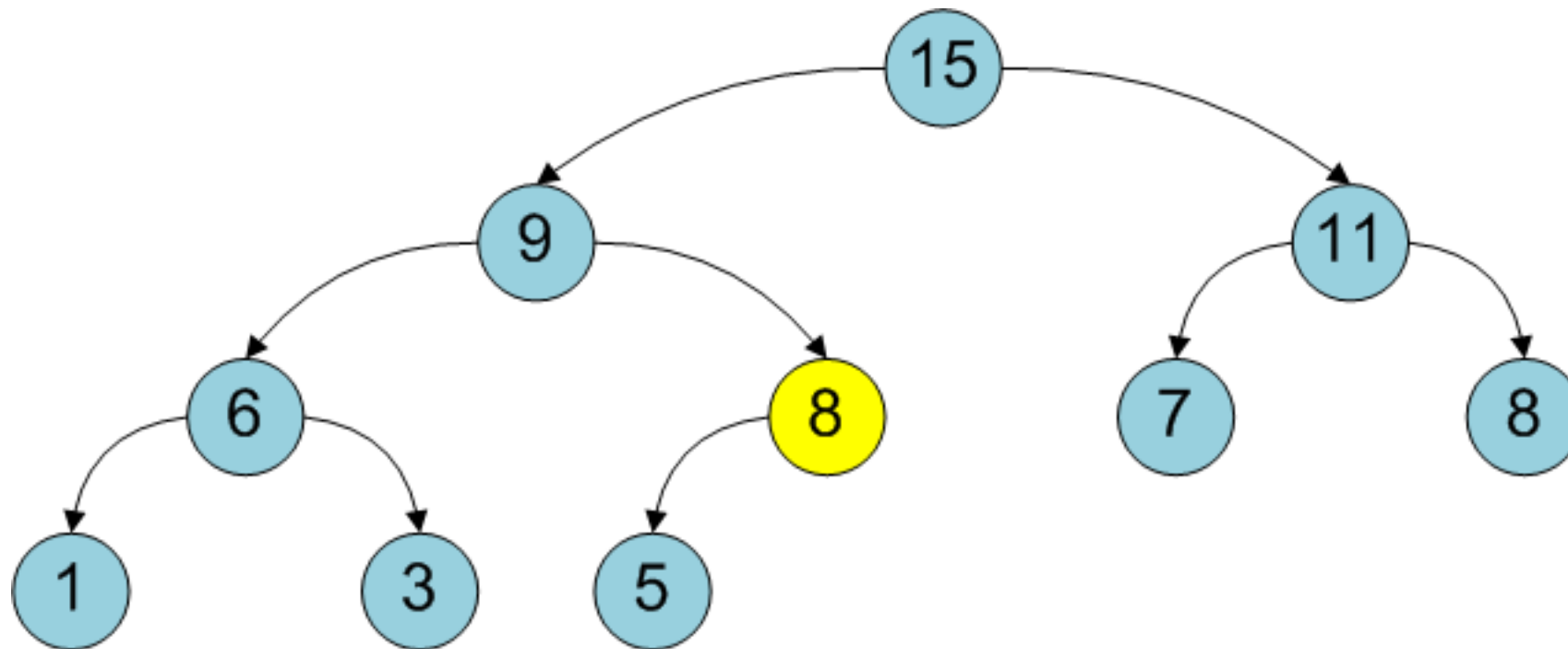
**Heapify**



**Quiz:** What is the complexity of heaping operation ?

Heap

Heapify



**Quiz:** What is the complexity of heaping operation ?

**Answer:**  $O(\log(N))$

## Heap

## Pop

1. Take root out
2. Put the last heap element on the place of the root
3. Heapify

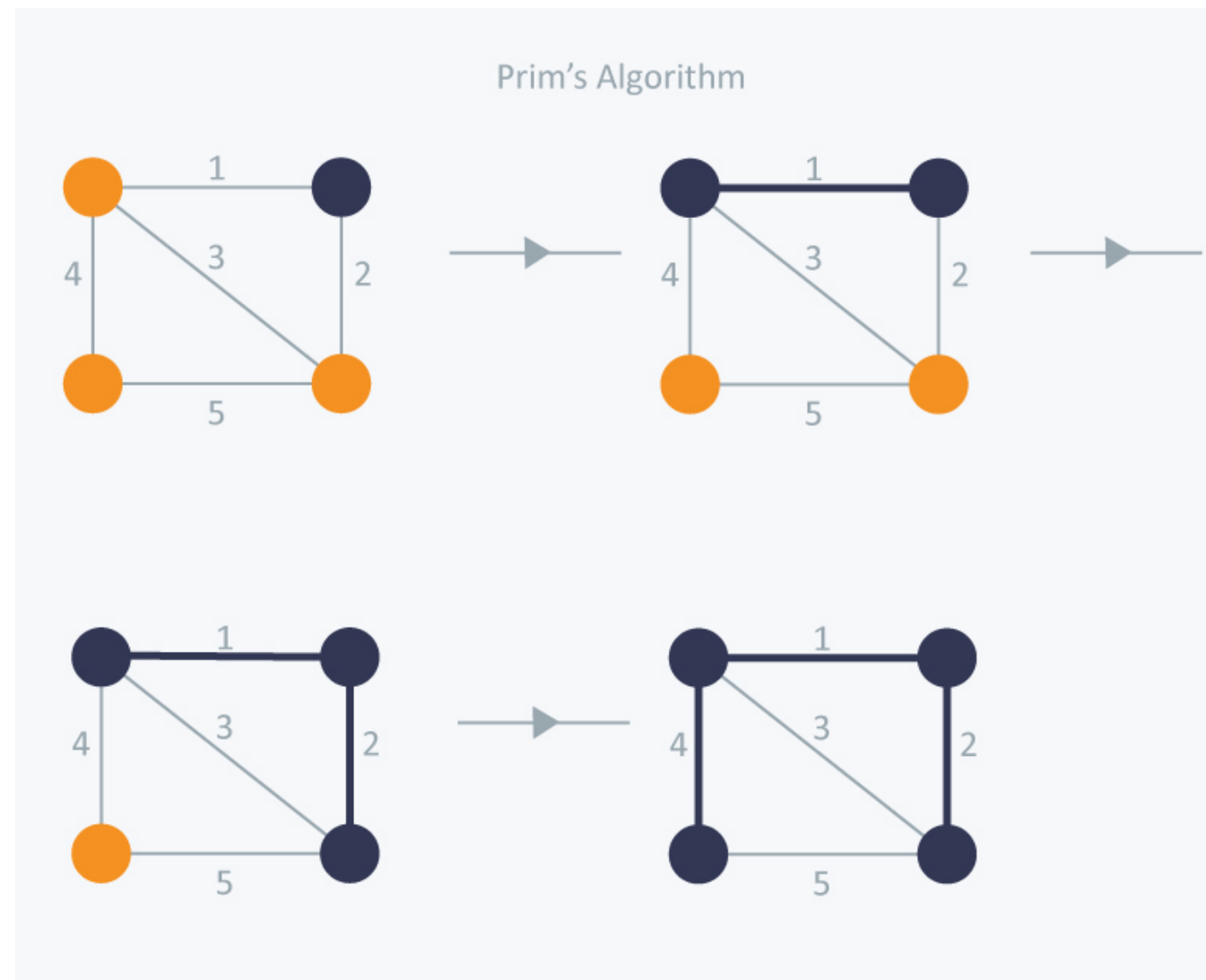
## Heap

### Applications

- Heap sort
- Priority queue

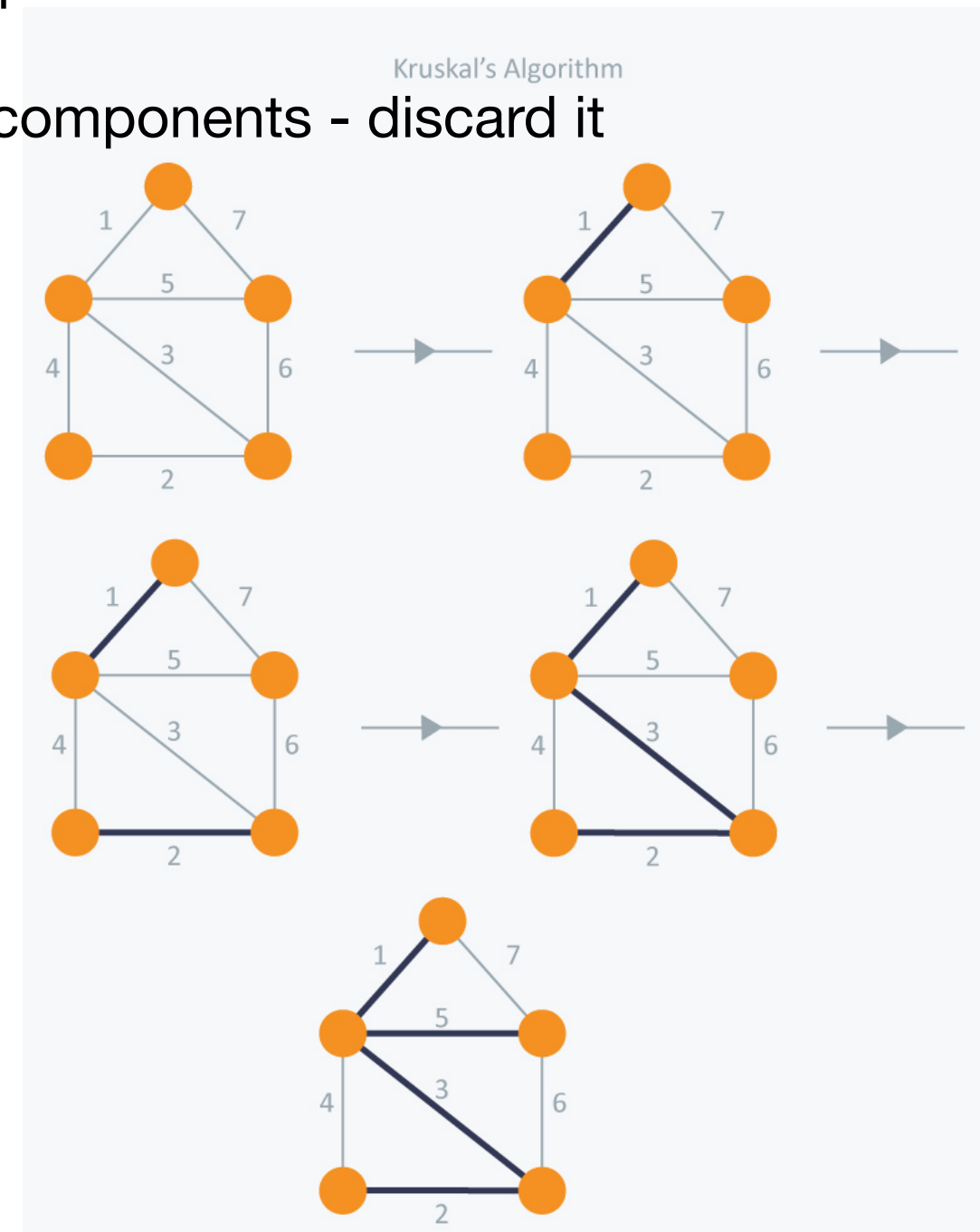
## Prim algorithm

1. Select random node to start from
2. While non-tree nodes remaining:
  1. Select an edge of minimum weight between a tree and non-tree vertex
  2. Add selected edge and vertex to the tree



## Kruskal algorithm

1. Initialise every node to be a single connected component
2. Consider the lightest edge
  1. If adjacent nodes are in the same connected components - discard it
  2. Otherwise, add edge, merge components





## Prim vs Kruskal

- Prim  **$O(VE)$** , if we don't keep track of cheapest edge
  - Prim  **$O(VV)$** , if we do, but use simple data structure
  - Prim  **$O(E + V \log V)$** , if use priority queue
  - Kruskal  **$O(E \log E)$**
- 
- Prim **is better on dense graphs**
  - Kruskal **is better on sparse graphs**

## Shortest paths

**Path** is a sequence of edges connecting two vertices

- For unweighted graphs can be found with BFS
- Same for the graphs with equal weights

**Dijkstra algorithm** find shortest path between start and end vertices.

- $O(N^2)$
- Greedy
- On each step selects the cheapest to add edge
- Almost Prim

## Dijkstra algorithm

What is different from Prim:

Instead of considering only the weight of edge consider the length of the path from the start node