# Greedy

Algorithm Analysis

- Basic Idea
  - Change Making
- Review - Kruskal's algorithm
- Union Find
- Review – Prim's algorithm
- Priority Queue (heap)
- Heapsort

# Basic Idea

Greedy

# Greedy algorithm solves problem by

performing a sequence of

locally optimal,  irrevocable steps.

# Change Making

change of X cents using the least number of coins

U.S. coin denominations    (1,5,10,25)

# X = 37 by Greedy

| iteration | Remaining change | 25 | 10 | 5 | 1 |
|---|---|---|---|---|---|
| 0 | 37 | | | | |
| 1 | 12 | 1 | 0 | 0 | 0 |
| 2 | 2 | 1 | 1 | 0 | 0 |
| 3 | 0 | 1 | 1 | 0 | 2 |

Correct? Yes but only for U.S. denominations.

Think about {1,7,10} and X==14

*Fractional Knapsack*

*W[] = {10, 20, 30}*
*V[] = {60, 100, 120}*
*Knapsack Capacity, W = 50;*

***Output:***
*Maximum possible value = 240*
*by taking items of weight 10 and 20 kg and 2/3 fraction*

# Kruskal's Algorithm

Find a weighted graph’s minimal spanning tree (MST).

# Minimal spanning tree

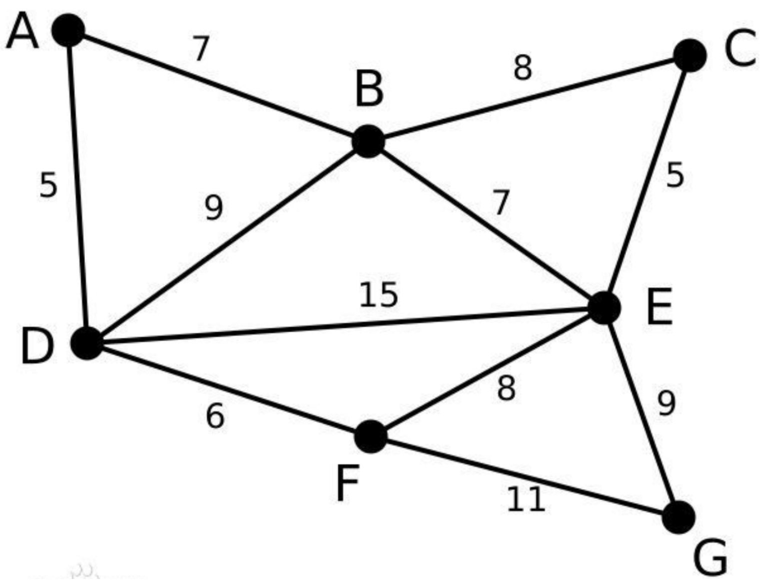
**Tree - connected acyclic graph**

**Spanning tree - a subgraph of a given graph which**

1) **contains all vertices and**
2) **forms a tree.**

**Minimal spanning tree - a spanning tree of minimal weight.**

# Example



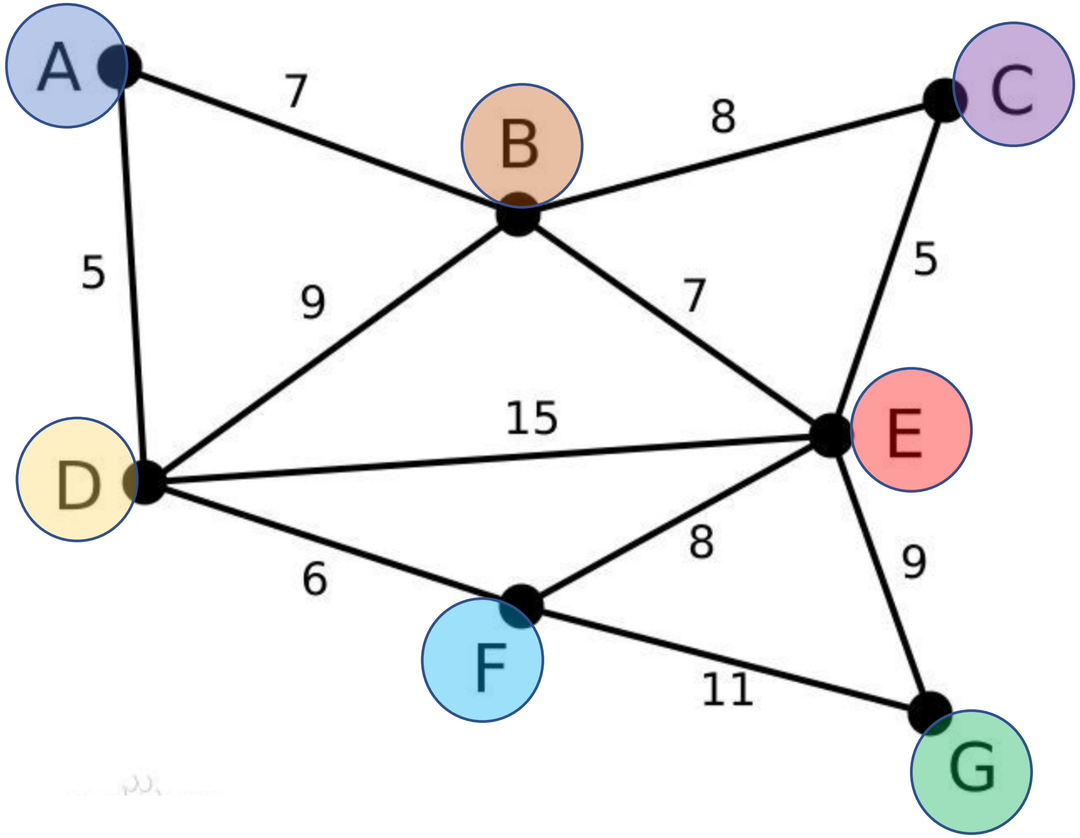

V: set of vertices
E: set of edges

The number of edges in a tree == |V|-1

Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

# Example-coloring

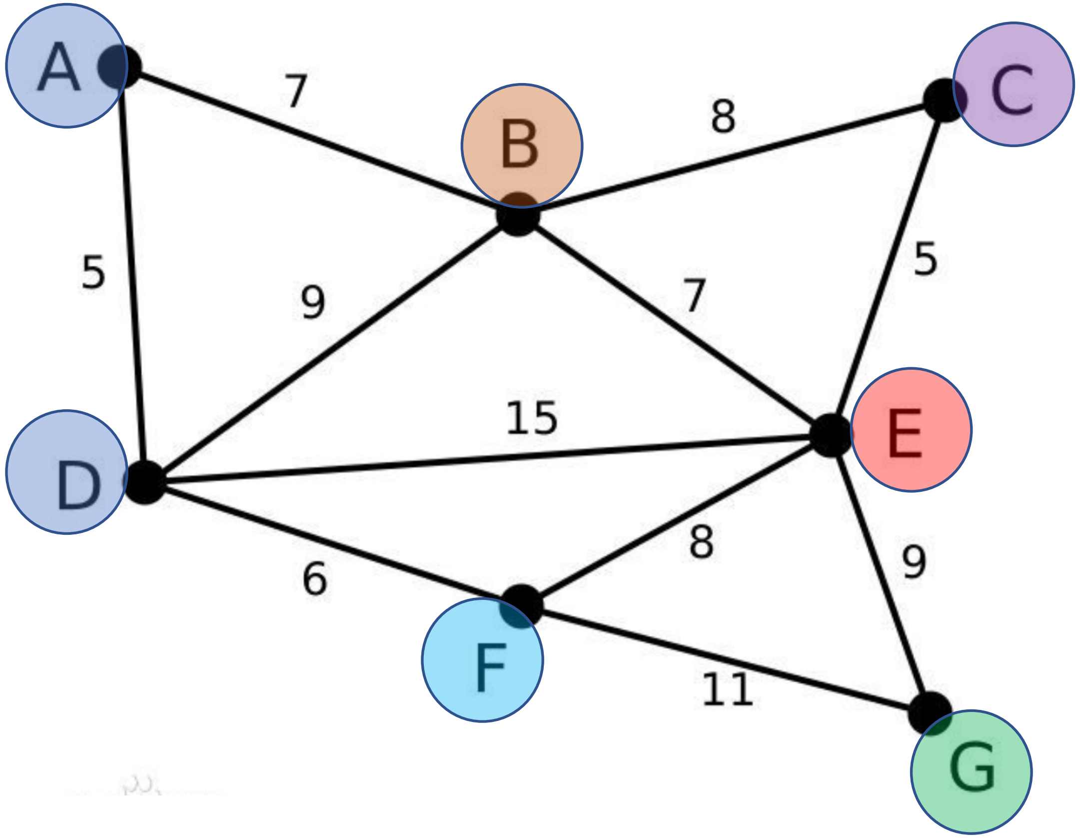



Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|

# Example-{AD}

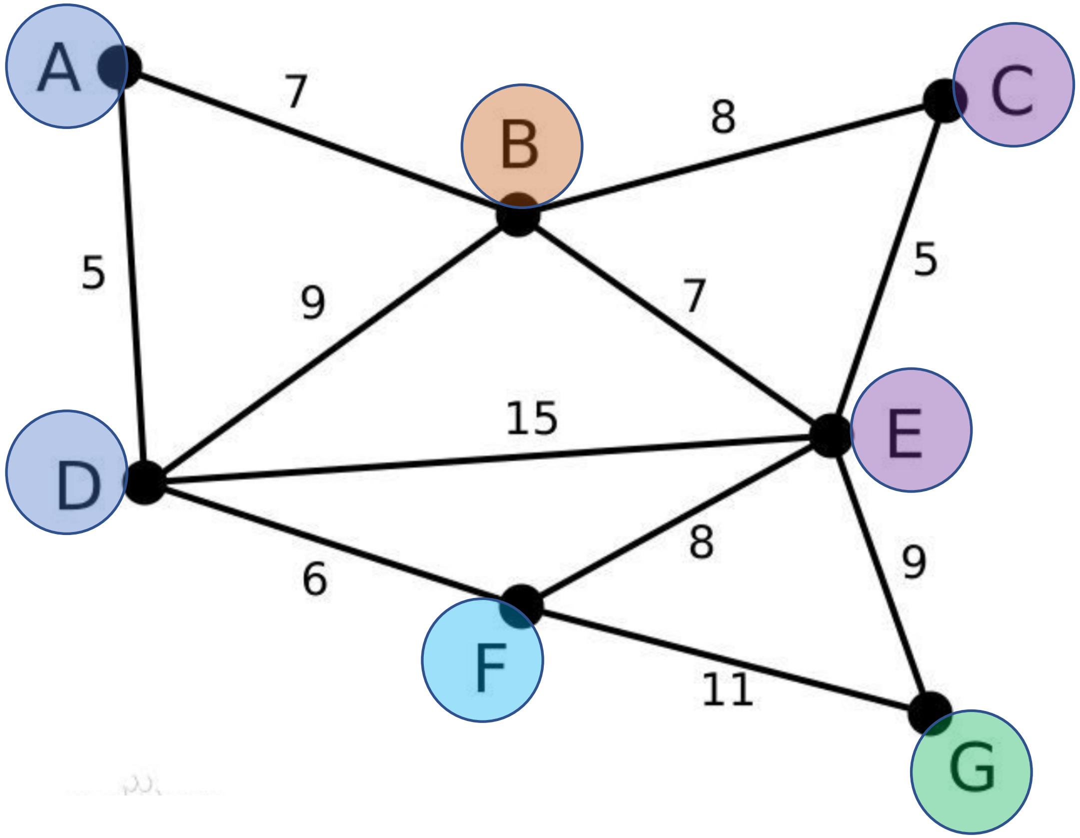



Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|

# Example-{AD CE}

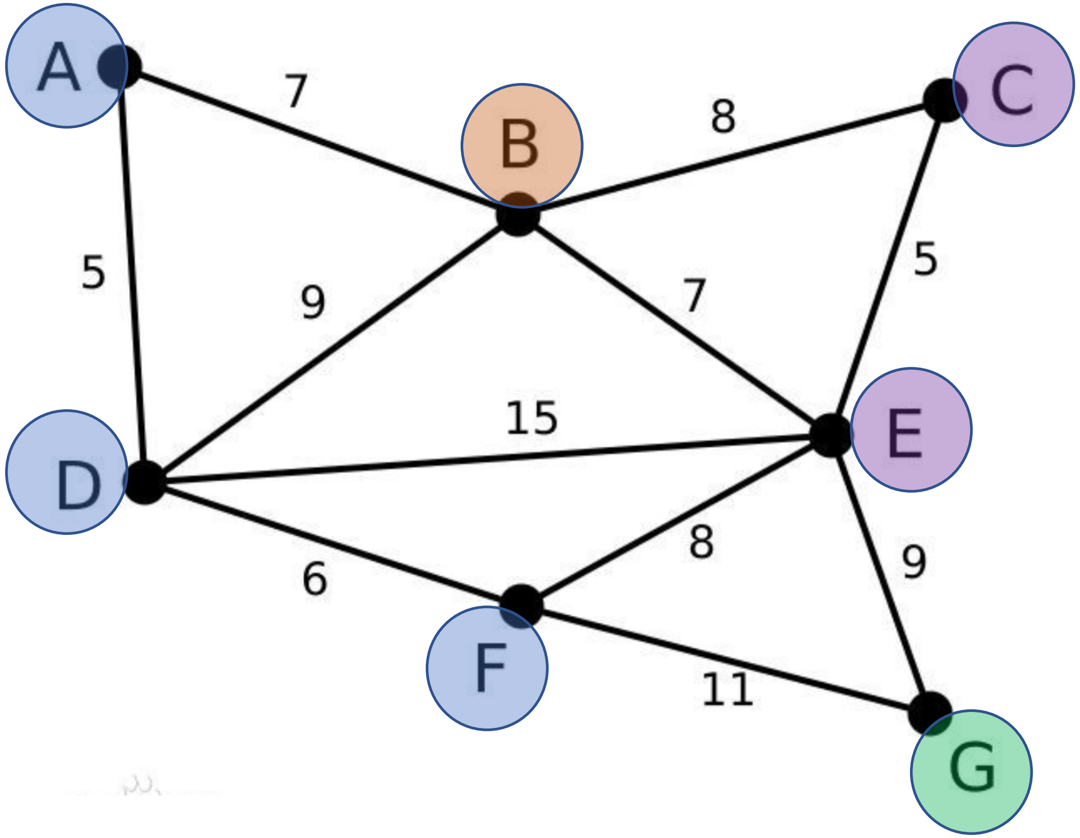



Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|

# Example-{AD CE DF}

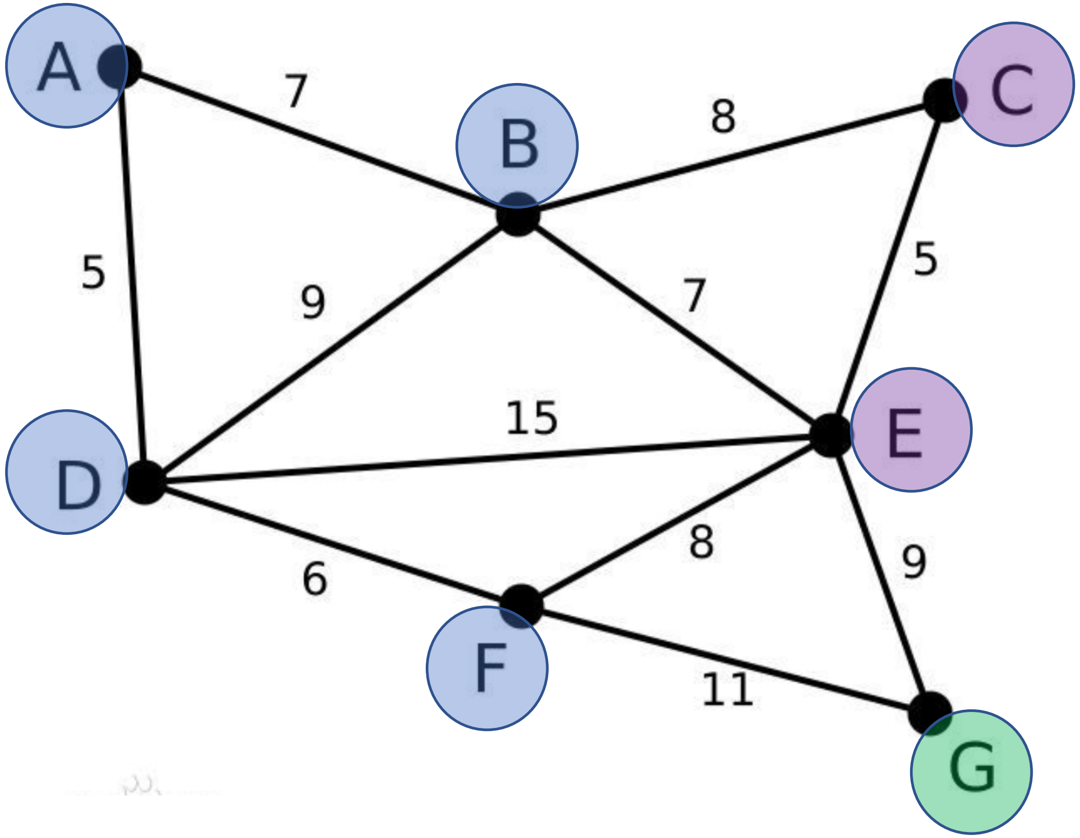



Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|

# Example-{AD CE DF AB}

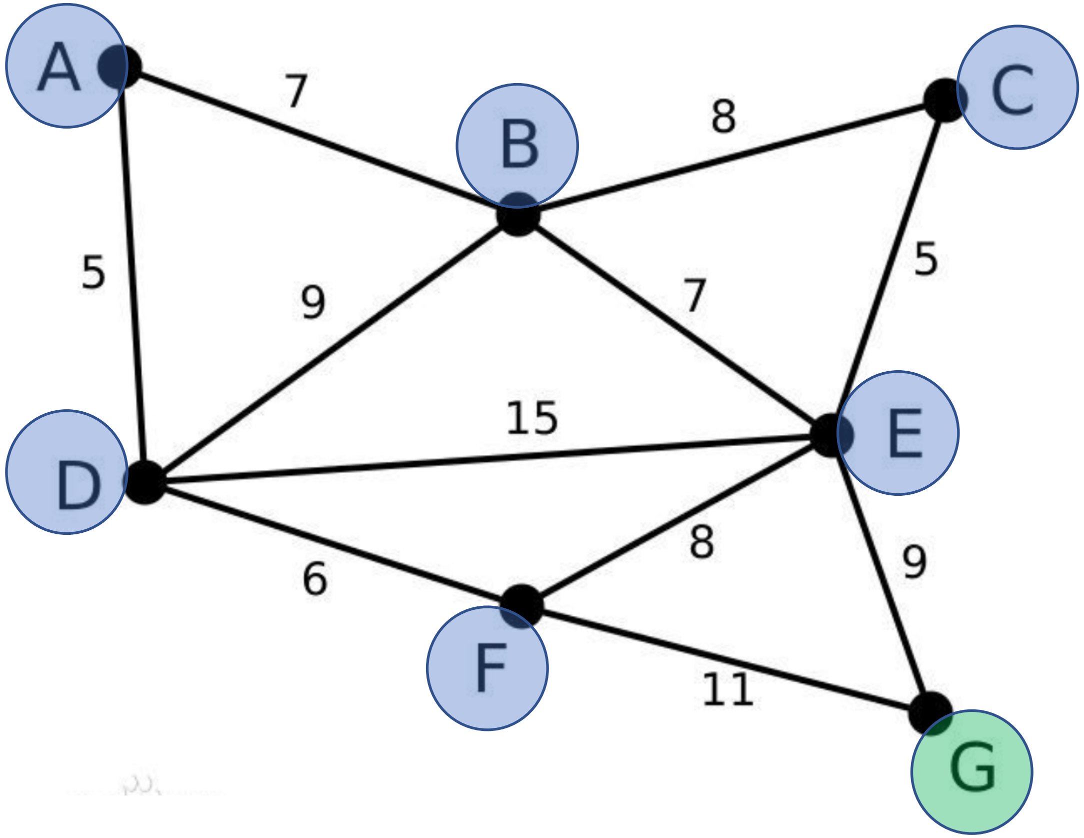



Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|

# Example-{AD CE DF AB BE}



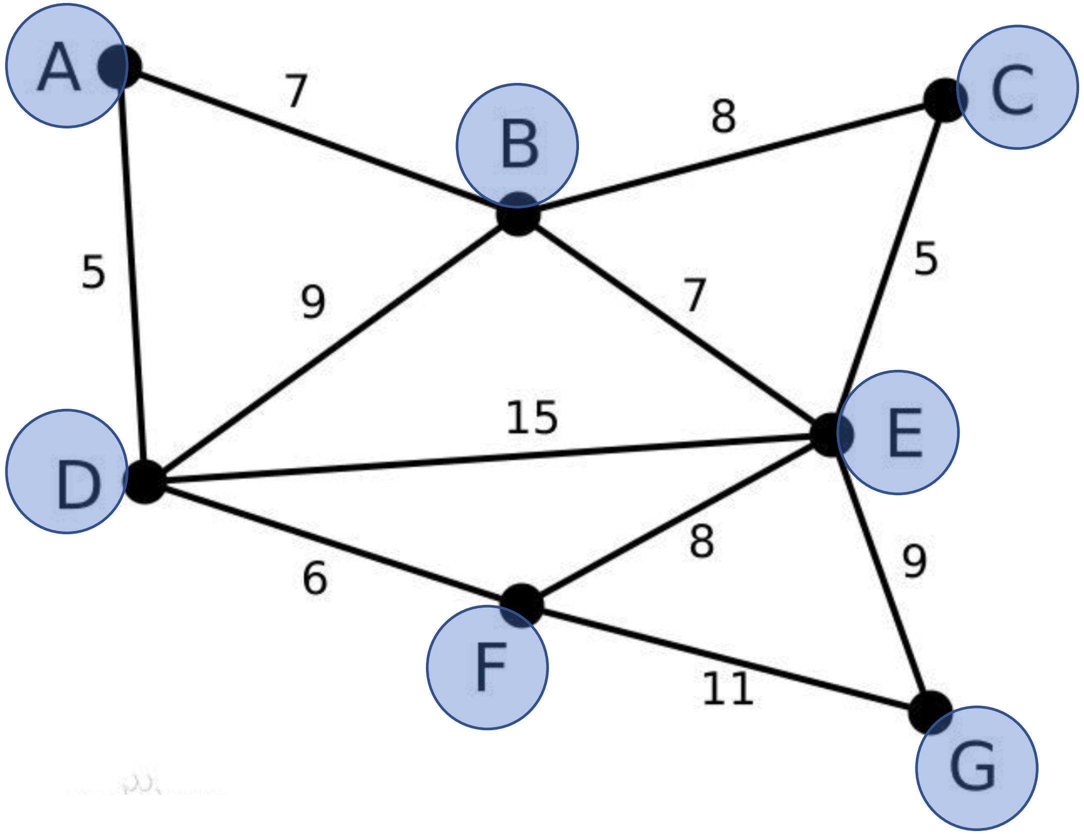

Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

| | | | | | | |
|---|---|---|---|---|---|---|
| A | B | C | D | E | F | G |

# Example-{AD CE DF AB BE EG}

Sorting (weight)



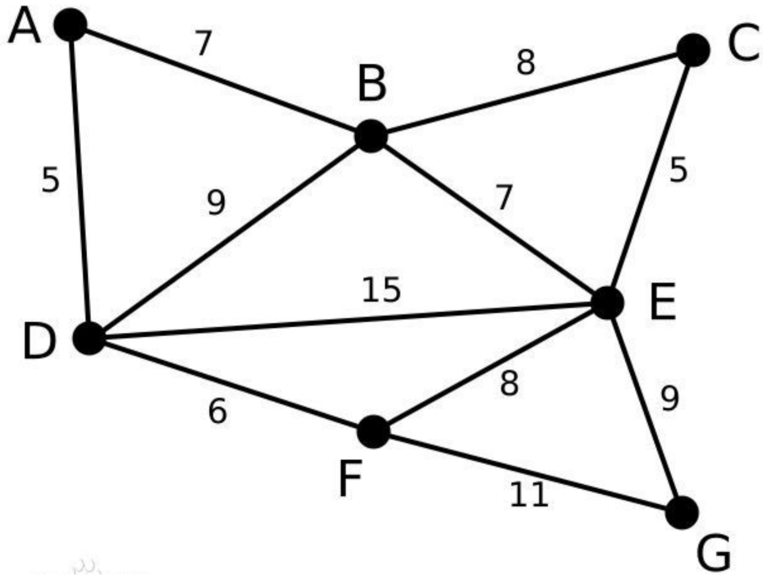

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

Sorting:
O(|E|log|E|)

Coloring:
O((|V|-1)*V)

| A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|

# Example – Disjoint Subsets




idea: keep same color vertices in a singly-linked list

Sorting (weight)

| Weight | Edge |
|---|---|
| 5 | AD |
| 5 | CE |
| 6 | DF |
| 7 | AB |
| 7 | BE |
| 8 | BC |
| 8 | EF |
| 9 | EG |
| 9 | DB |
| 11 | FG |
| 15 | DE |

# Disjoint Subsets

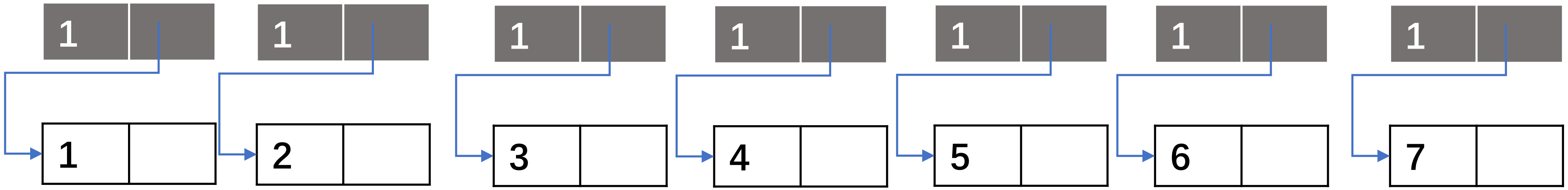
Union and Find algorithm

# Initialization

Array of representative

| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|
| A-1 | B-2 | C-3 | D-4 | E-5 | F-6 | G-7 |

7 singly- linked lists




O(|V|)

# Find- {AD 14}

P: Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |

$P(1) = 1$

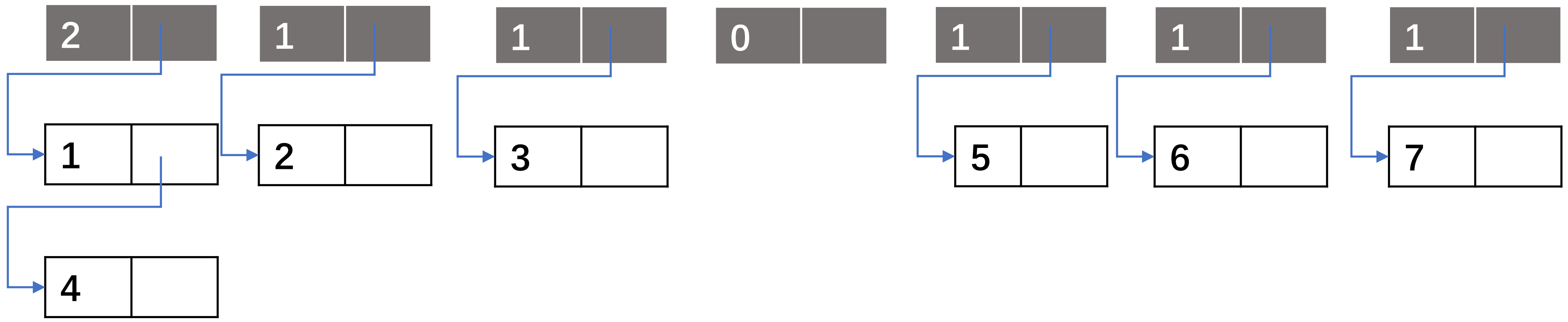
$P(4) = 4$

# Union- {AD 14}

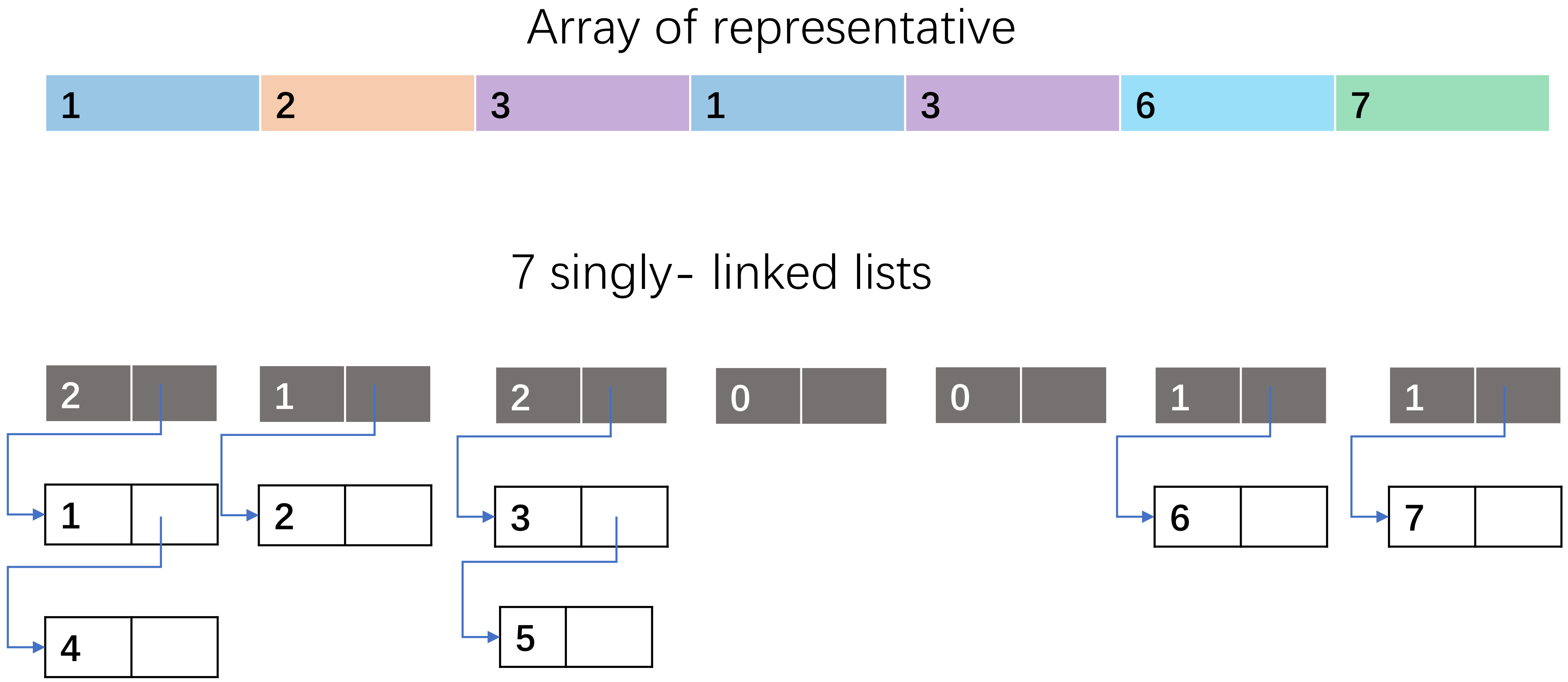
Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 1 | 5 | 6 | 7 |

7 singly- linked lists



# Find & Union- {AD 14, CE 35}

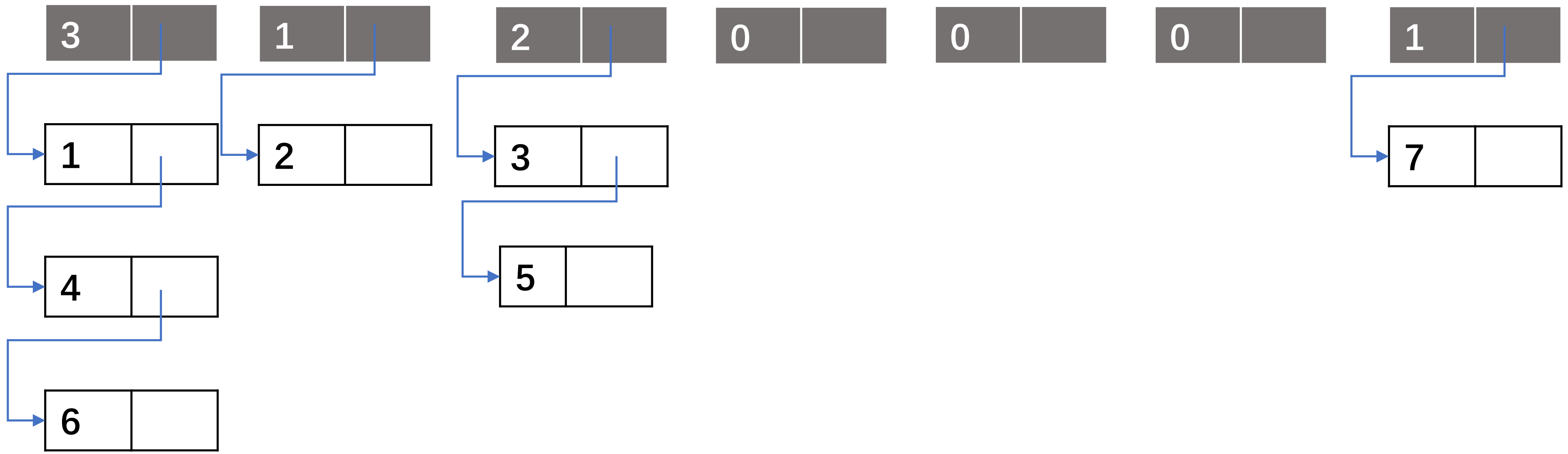


# Find & Union- {AD 14, CE 35, DF 46}

Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 2 | 3 | 1 | 3 | 1 | 7 |

7 singly- linked lists

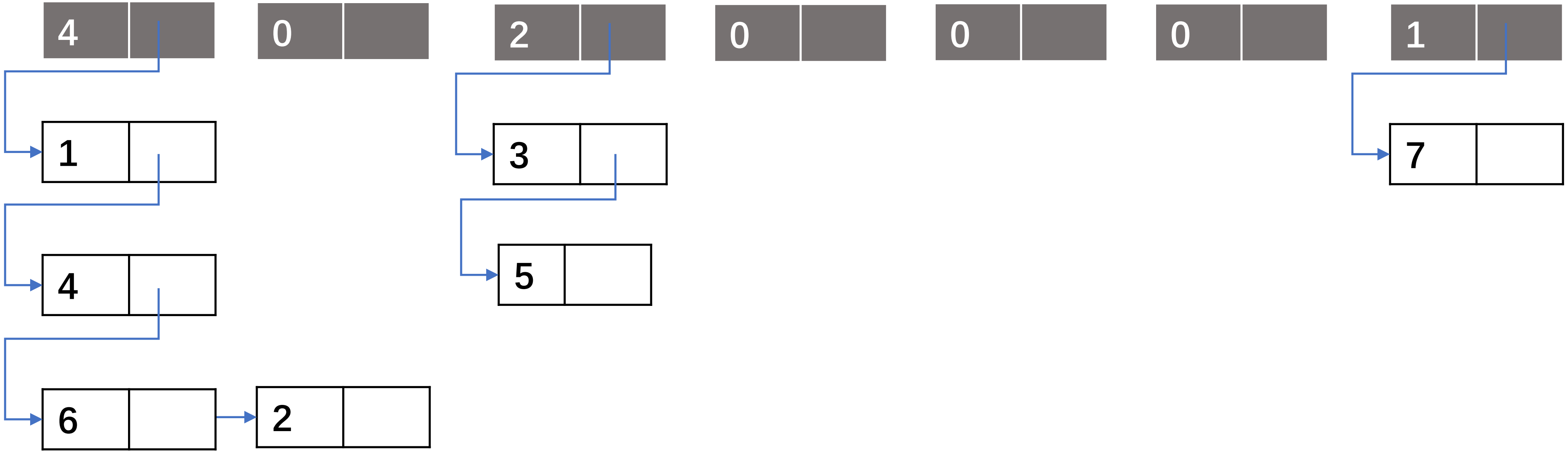


# Find & Union- {AD 14, CE 35, DF 46, AB 12}

Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 3 | 1 | 3 | 1 | 7 |

7 singly- linked lists

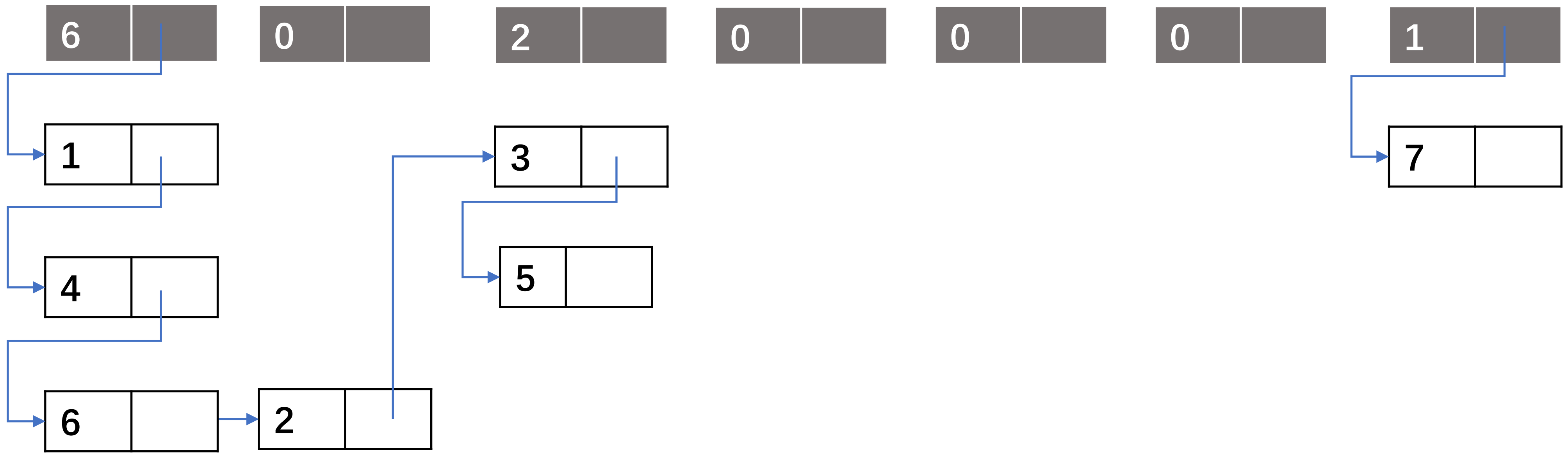


# Find & Union- {AD 14, CE 35, DF 46, AB 12, BE 25}

Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 3 | 1 | 7 |

7 singly- linked lists

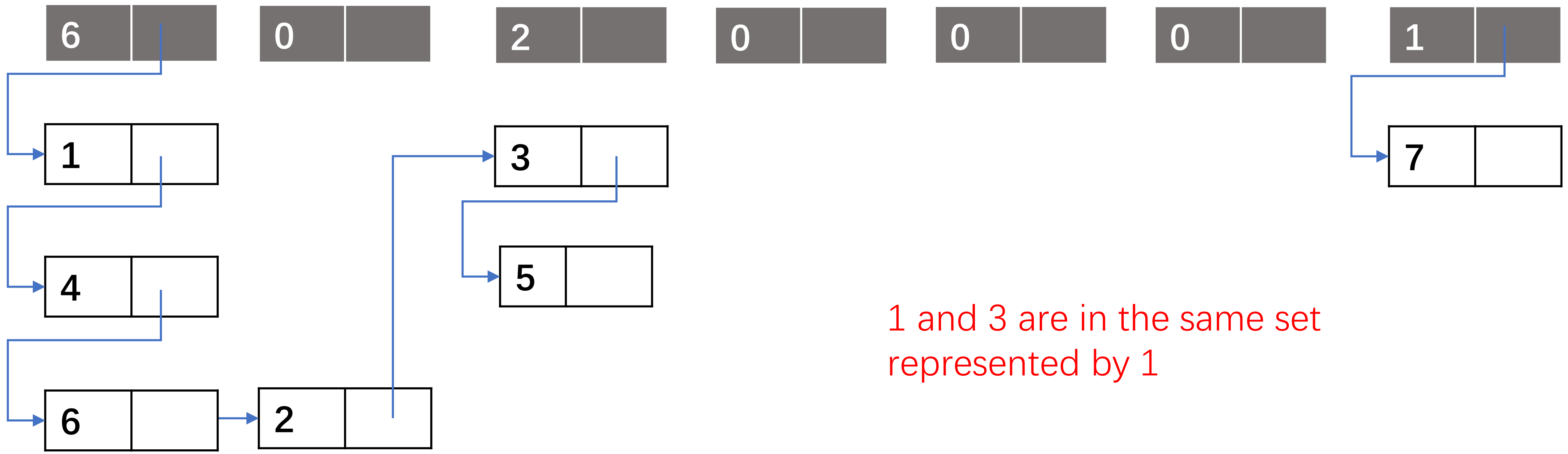


# Find - {AD 14, CE 35, DF 46, AB 12, BE 25, BC 23}

Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 7 |

7 singly- linked lists



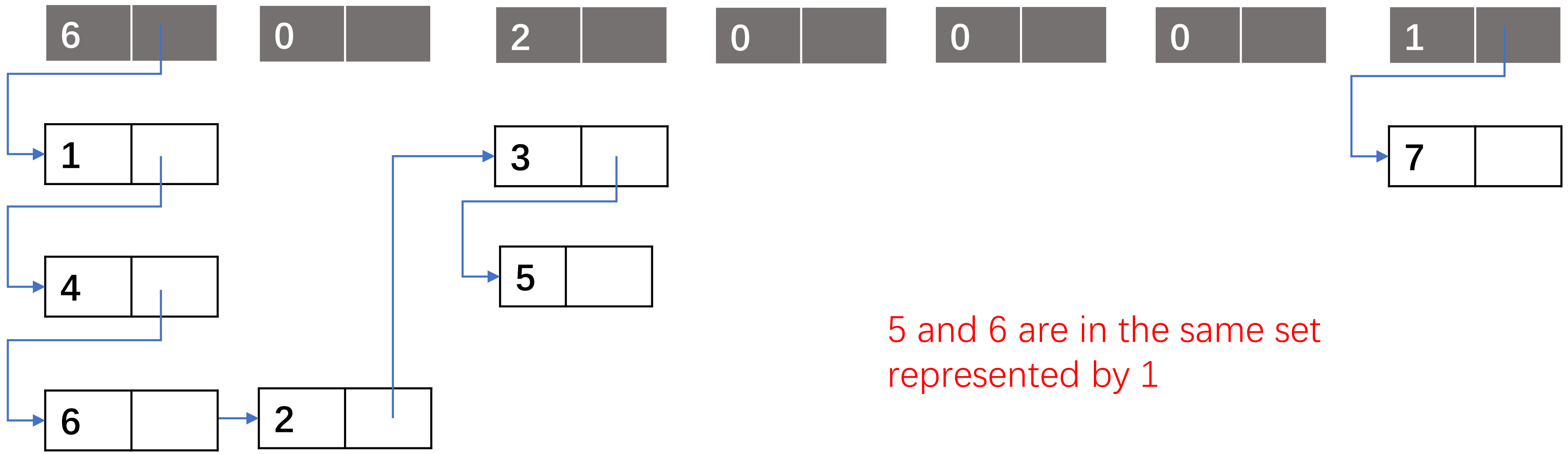

1 and 3 are in the same set represented by 1

# Find - {AD 14, CE 35, DF 46, AB 12, BE 25, EF 56}

Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 7 |

7 singly- linked lists



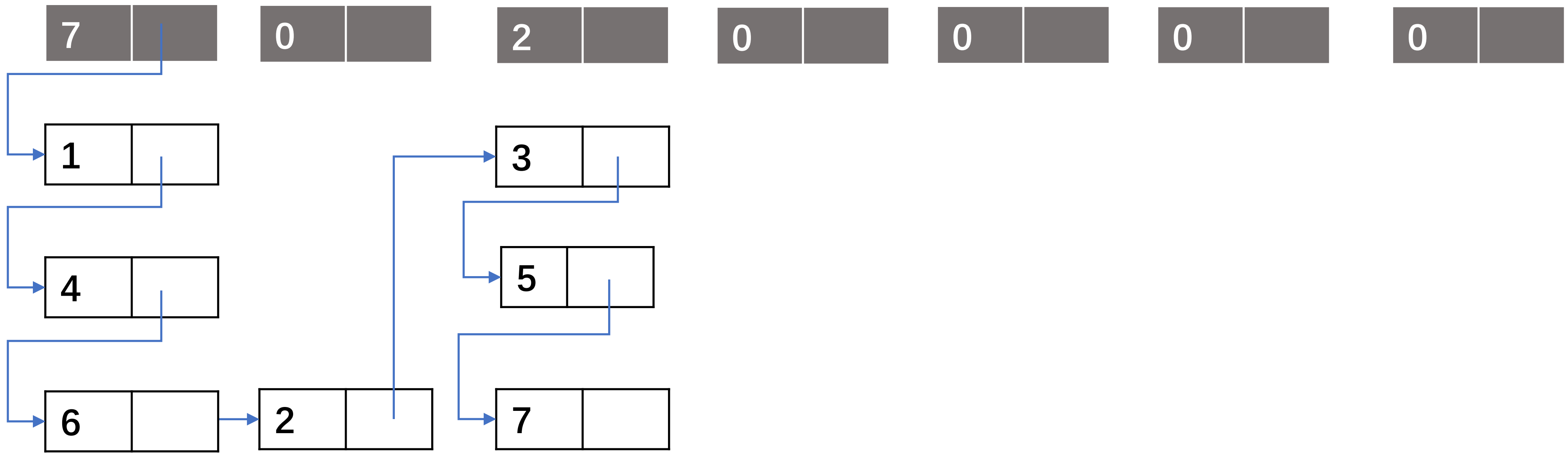

5 and 6 are in the same set represented by 1

# Find&Union - {AD 14, CE 35, DF 46, AB 12, BE 25, EG 57}

Array of representative

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 |

7 singly- linked lists



# Reconsider Kruskal's algorithm

Sorting:
$O(|E|\log|E|)$

Coloring:
$O((|V|-1)*V)$

Sorting:
$O(|E|\log|E|)$

Find and Union: (at most $|E|$ iterations
$O(|E| + |V|)$

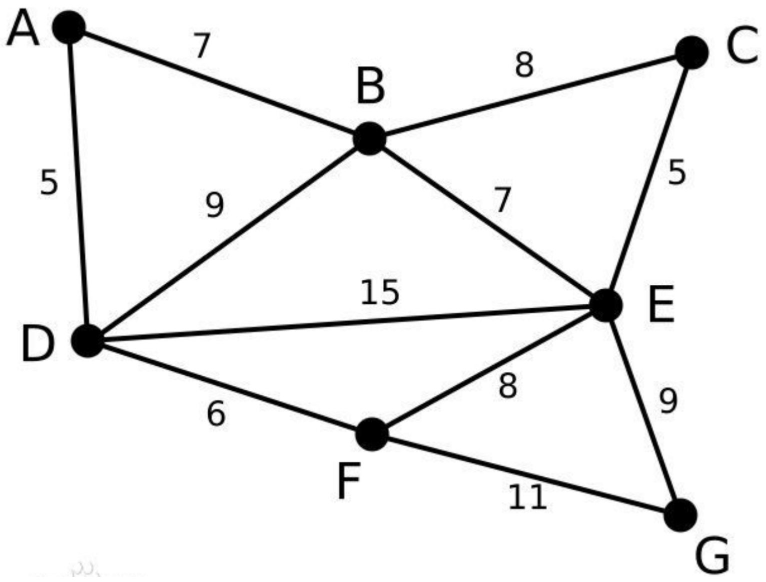
$E>V-2$ (connected graph)

$O(|E|\log|E|)$

# Prim’s algorithm

# Example - Ini

Randomly select a node, say, **D**

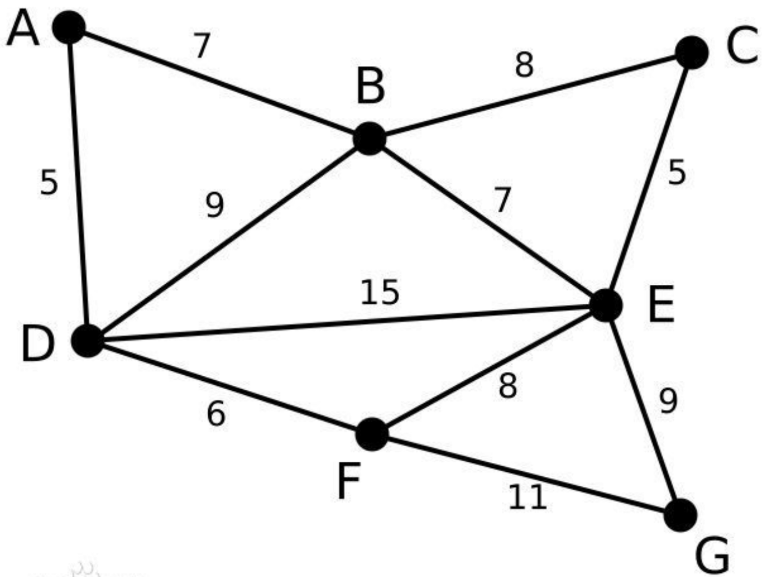



MST= {}

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| cost | 5 | 9 | ∞ | 15 | 6 | ∞ |

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| mst | D | D | D | D | D | D |

# Example - Update

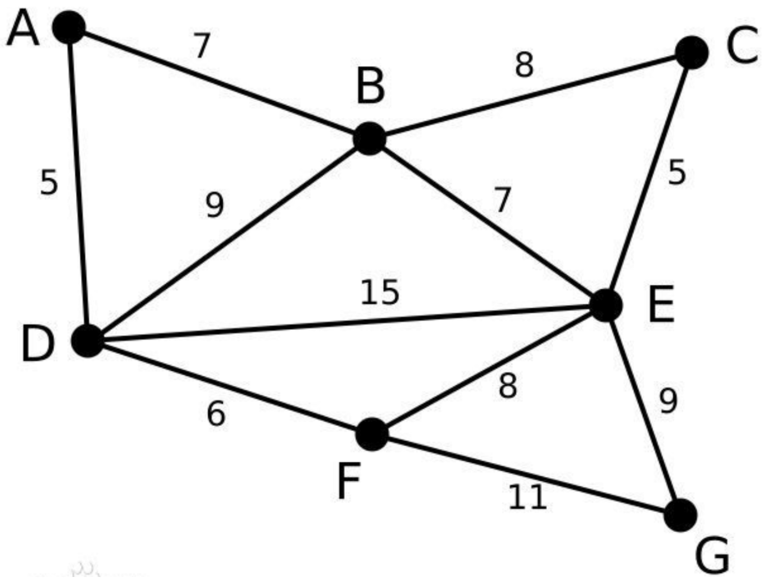



MST=
{AD}

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| cost | | 7 | ∞ | 15 | 6 | ∞ |

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| mst | 0 | A | D | D | D | D |

# Example - Update




MST=
{AD DF}

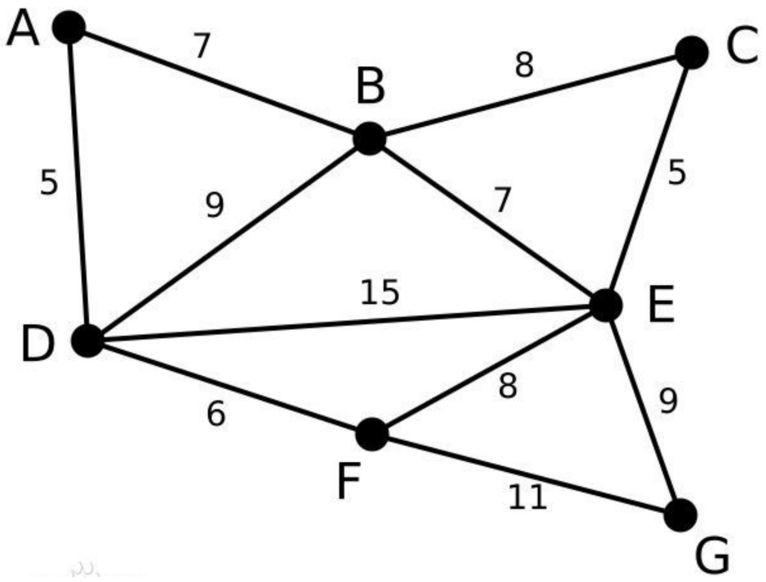
cost

| A | B | C | E | F | G |
|---|---|---|---|---|---|
| | 7 | ∞ | 8 | | 11 |

mst

| A | B | C | E | F | G |
|---|---|---|---|---|---|
| 0 | A | D | F | 0 | F |

# Example - Update

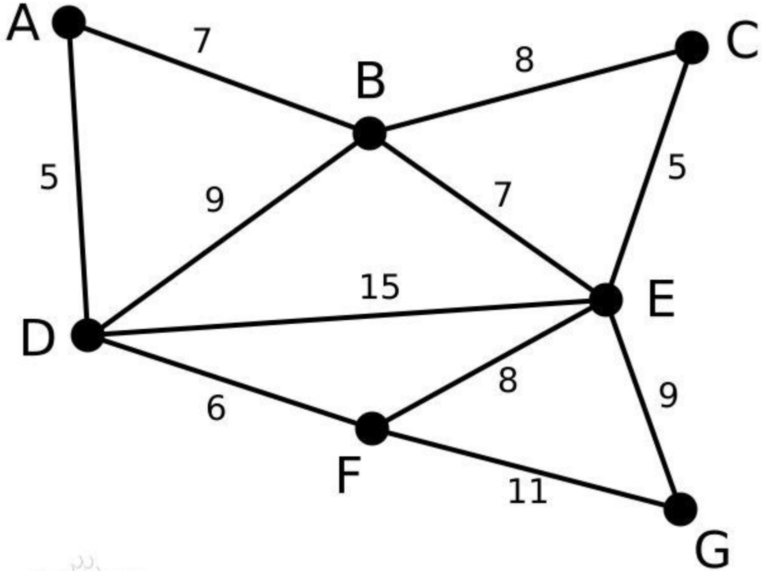



MST=
{AD DF
AB}

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| cost | | | 8 | 7 | | 11 |

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| mst | 0 | 0 | B | B | 0 | F |

# Example - Update

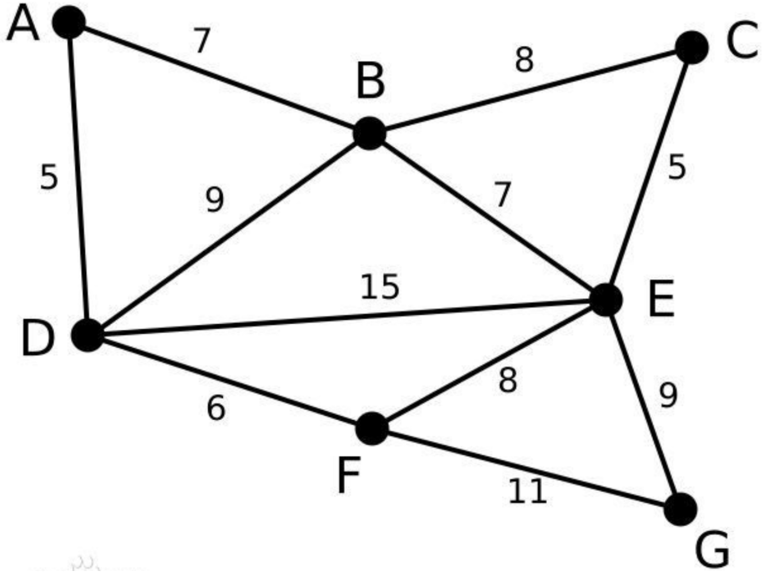



MST=
{AD DF
AB BE}

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| cost | | | 5 | | | 9 |

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| mst | 0 | 0 | E | 0 | 0 | E |

# Example - Update

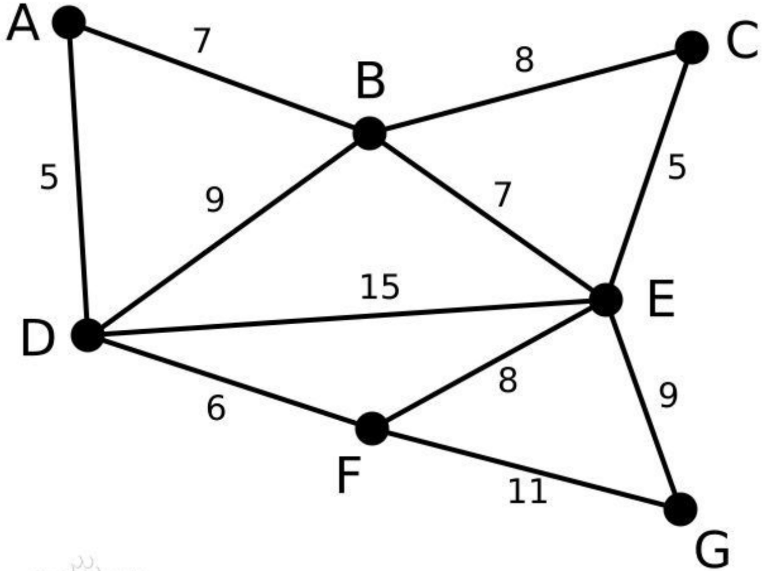



MST=
{AD DF
AB BE CE}

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| cost | | | | | | 9 |

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| mst | 0 | 0 | 0 | 0 | 0 | E |

# Example - Update




MST=
{AD DF
AB BE CE
EG}

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| cost | | | | | | |

| | A | B | C | E | F | G |
|---|---|---|---|---|---|---|
| mst | 0 | 0 | 0 | 0 | 0 | 0 |

# Priority Queue

Heap

# Min-heap

**Complete binary tree**

**the root of the heap/sub-heap is the smallest element**

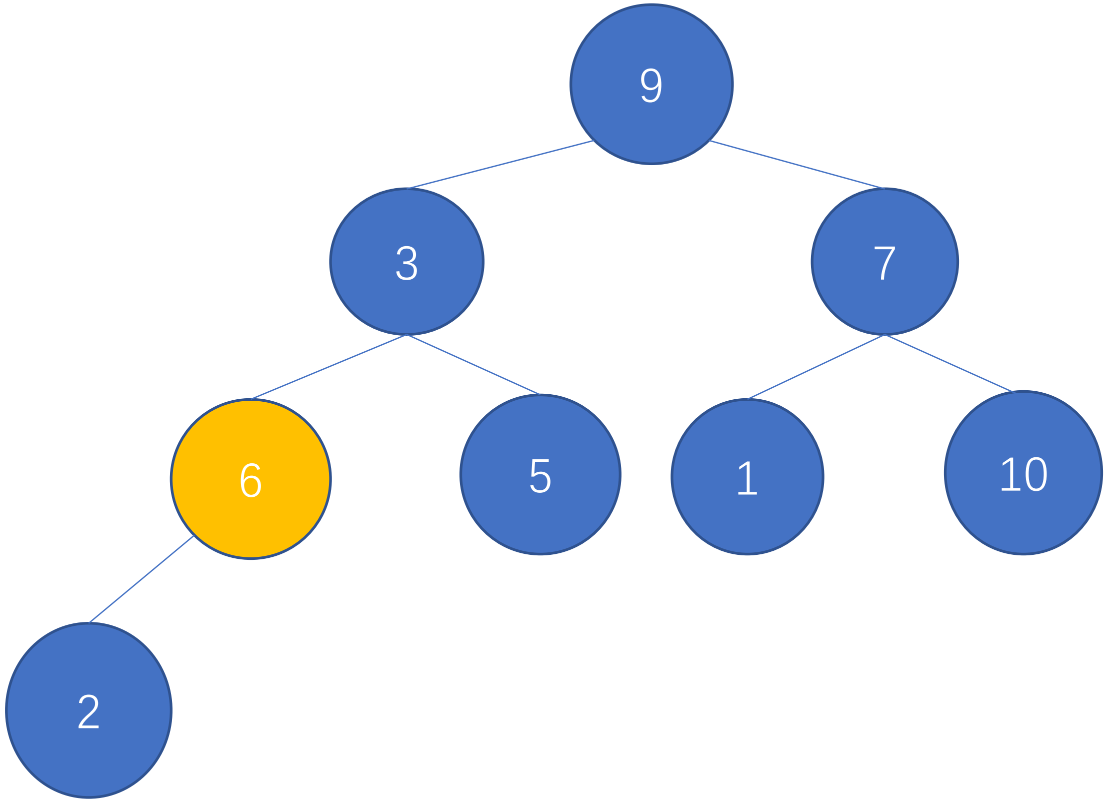
**Can be represented by an array. (index starts from 0)**

**left(i)  = 2i+1**
**right(i) = 2i+2**
**parent(i)= floor( (i-1)/2 )**

# Min-heap: Build

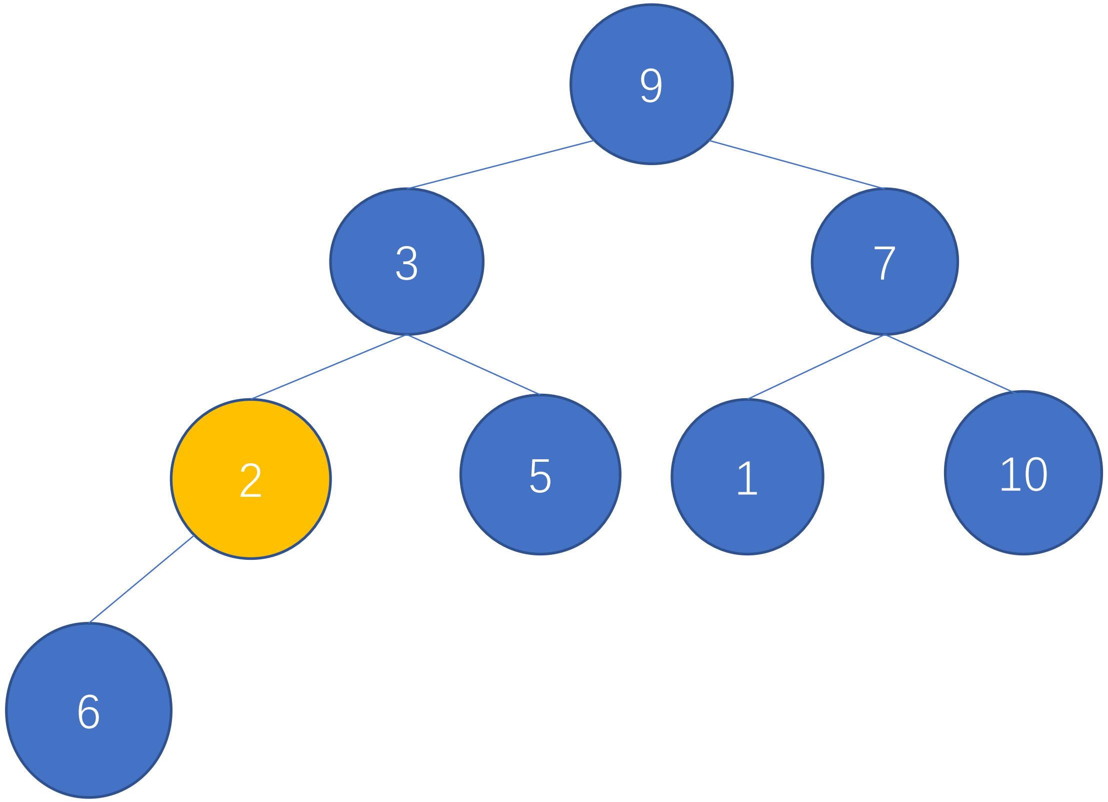
Start with the last non-leaf node

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 9 | 3 | 7 | 6 | 5 | 1 | 10 | 2 |



# Min-heap: Build

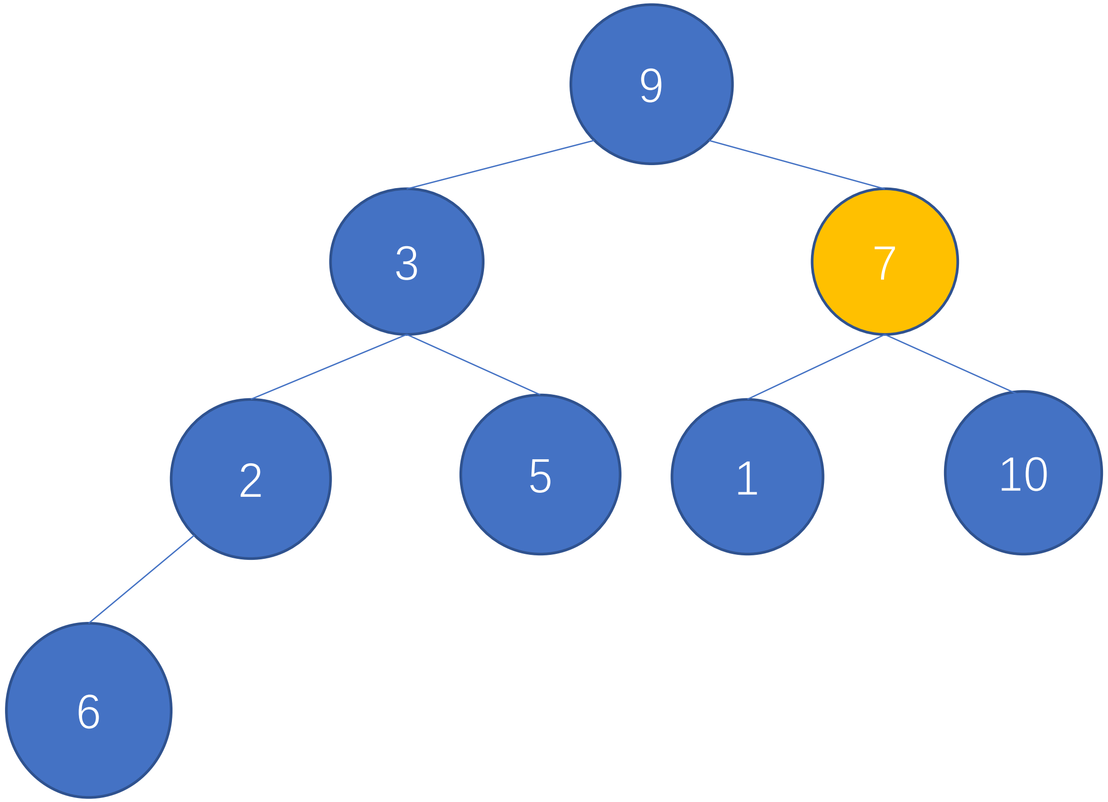
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 9 | 3 | 7 | 2 | 5 | 1 | 10 | 6 |




2<6 swapping

# Min-heap: Build

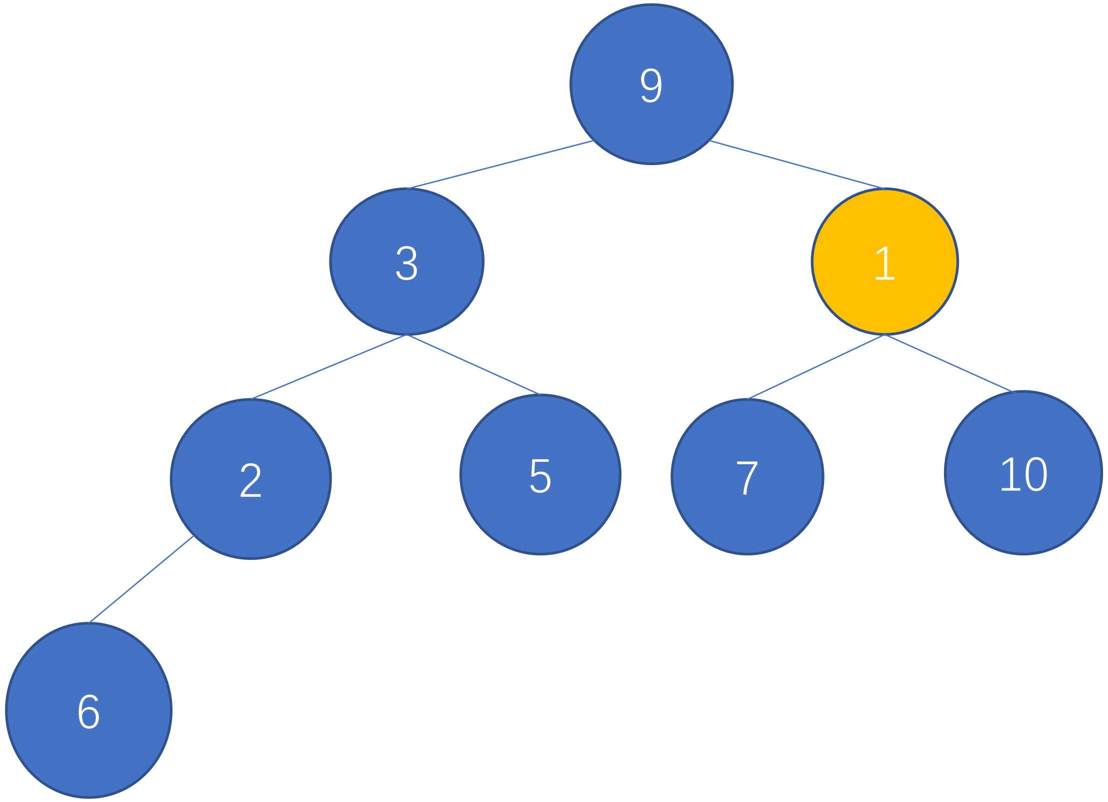
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 9 | 3 | 7 | 2 | 5 | 1 | 10 | 6 |




Looking at the second last non-leaf node

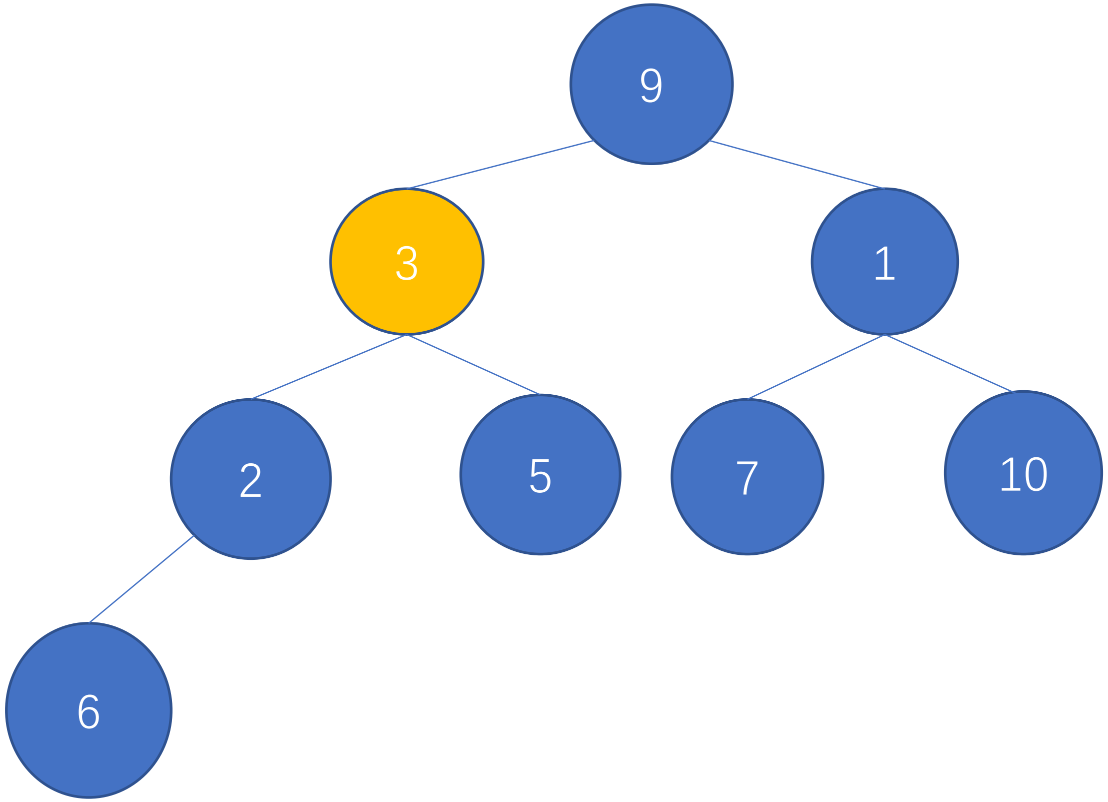
# Min-heap: Build

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 9 | 3 | 1 | 2 | 5 | 7 | 10 | 6 |



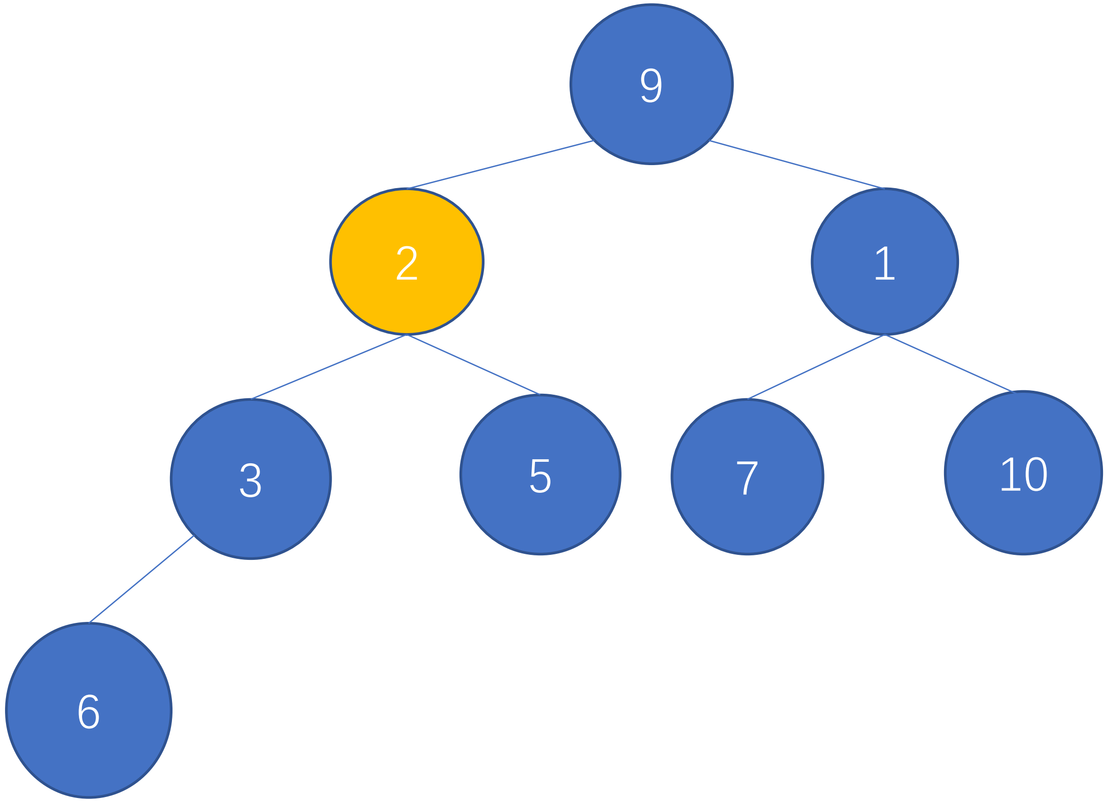
# Min-heap: Build

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 9 | 3 | 1 | 2 | 5 | 7 | 10 | 6 |



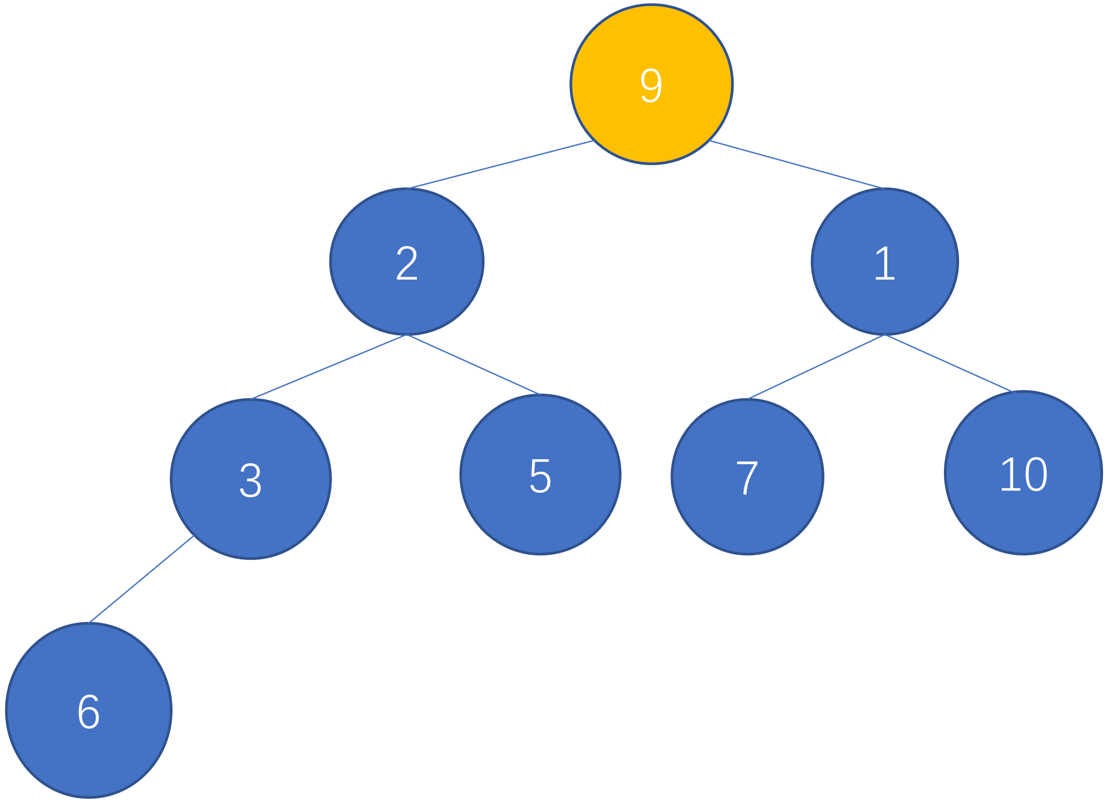
# Min-heap: Build

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 9 | 2 | 1 | 3 | 5 | 7 | 10 | 6 |



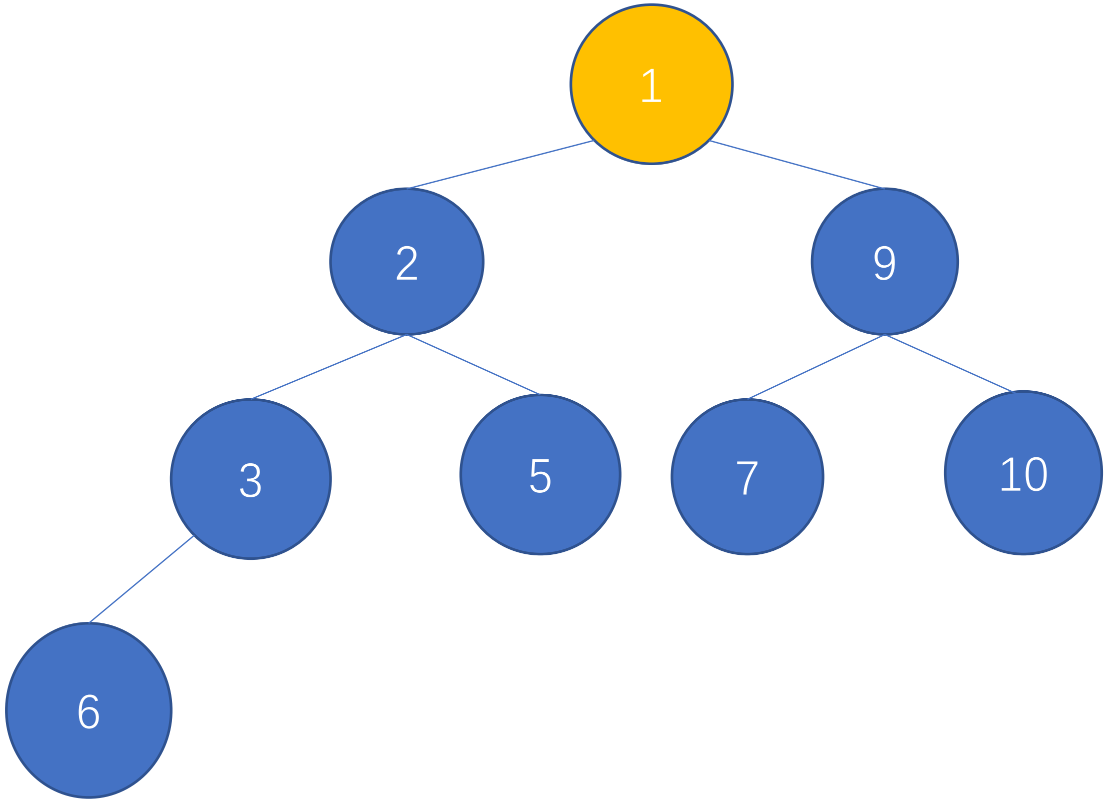
# Min-heap: Build

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 9 | 2 | 1 | 3 | 7 | 1 | 10 | 6 |



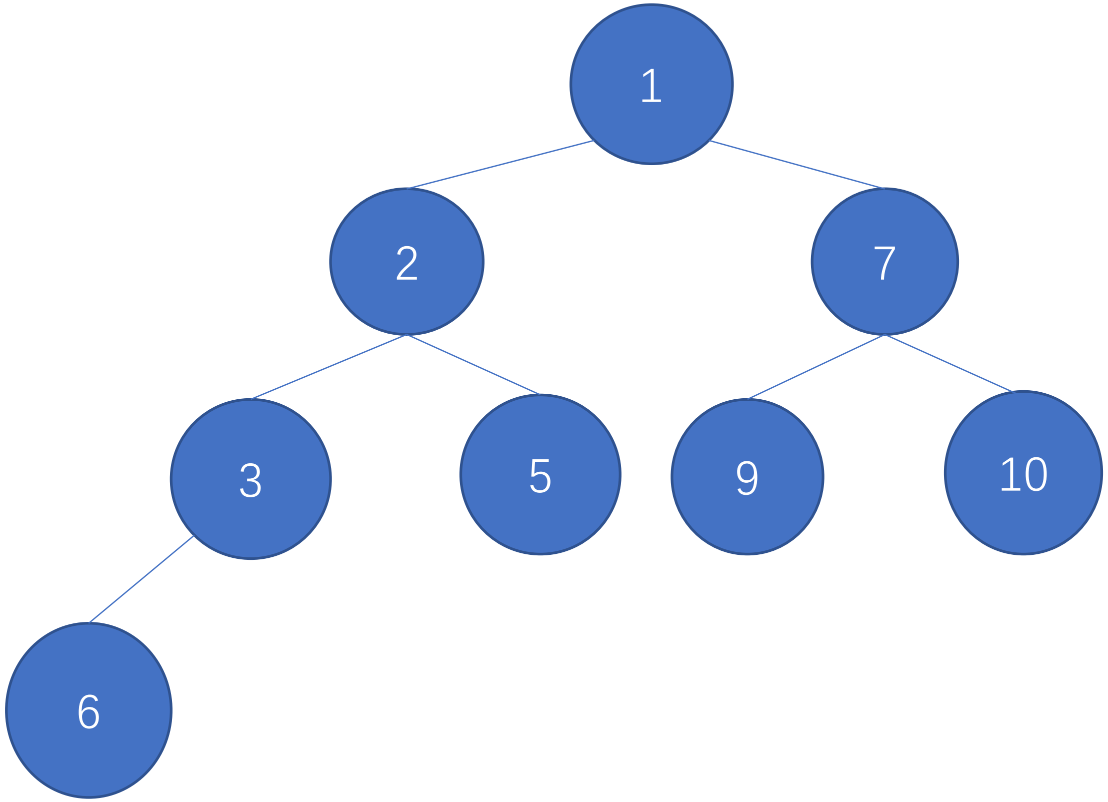
# Min-heap: Build

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 9 | 3 | 5 | 7 | 10 | 6 |



# Min-heap: Build

| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|
| 1 | 2 | 7 | 3 | 5 | 9 | 10 | 6 |




O(n)

# Min-heap-Build - O(n)

size: n

Height : k = floor(logn)+1

Level k: the number of nodes $2^{(k-1)}$, each non-leaf node needs to compare 0 time

Level k-1: #nodes = $2^{(k-2)}$, compare 1 time

…

Level 1: #nodes = $2^0$, compare k-1 time

$1*2^{(k-2)}+2*2^{(k-3)} +3*2^{(k-4)} +\cdots+(k-1)*2^0 <=n \rightarrow O(n)$ (see next slides)

Or

$T(n) = 2T(n/2) + \log n \rightarrow = O(n)$

# Min-heap-Build - O(n)

$1*2^{(k-2)}+2*2^{(k-3)}+3*2^{(k-4)}+\cdots+(k-1)*2^{0}$ K $=log_2 n+1$

$$S = 1\times\frac{n}{2^1}+2\times\frac{n}{2^2}+3\times\frac{n}{2^3}+\cdots+(k-1)\times\frac{n}{2^{k-1}}$$

$$S = (1\times\frac{1}{2^1}+2\times\frac{1}{2^2}+3\times\frac{1}{2^3}+\cdots+(k-1)\times\frac{1}{2^{k-1}})\, n$$

$$S/2 = (1\times\frac{1}{2^2}+2\times\frac{1}{2^3}+3\times\frac{1}{2^4}+\cdots+(k-2)\times\frac{1}{2^{k-1}}+(k-1)\times\frac{1}{2^k})\, n$$

$$S - S/2 = (\frac{1}{2^1}+\frac{1}{2^2}+\frac{1}{2^3}+\cdots+\frac{1}{2^{k-1}} - (k-1)\times\frac{1}{2^k})\, n = \text{n-logn-1}$$

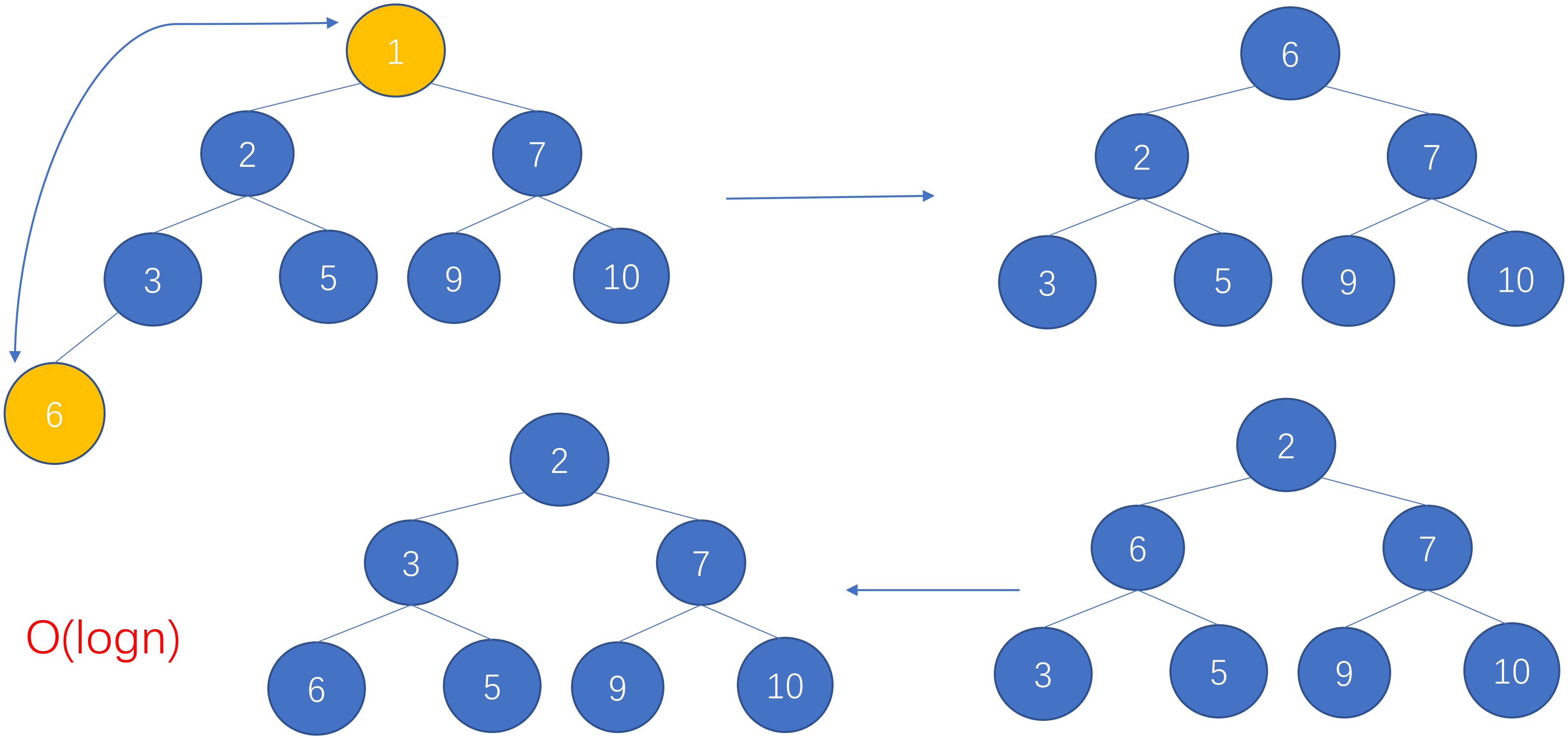
$S<=n$

# Min-heap- minimum

**Return the first element**

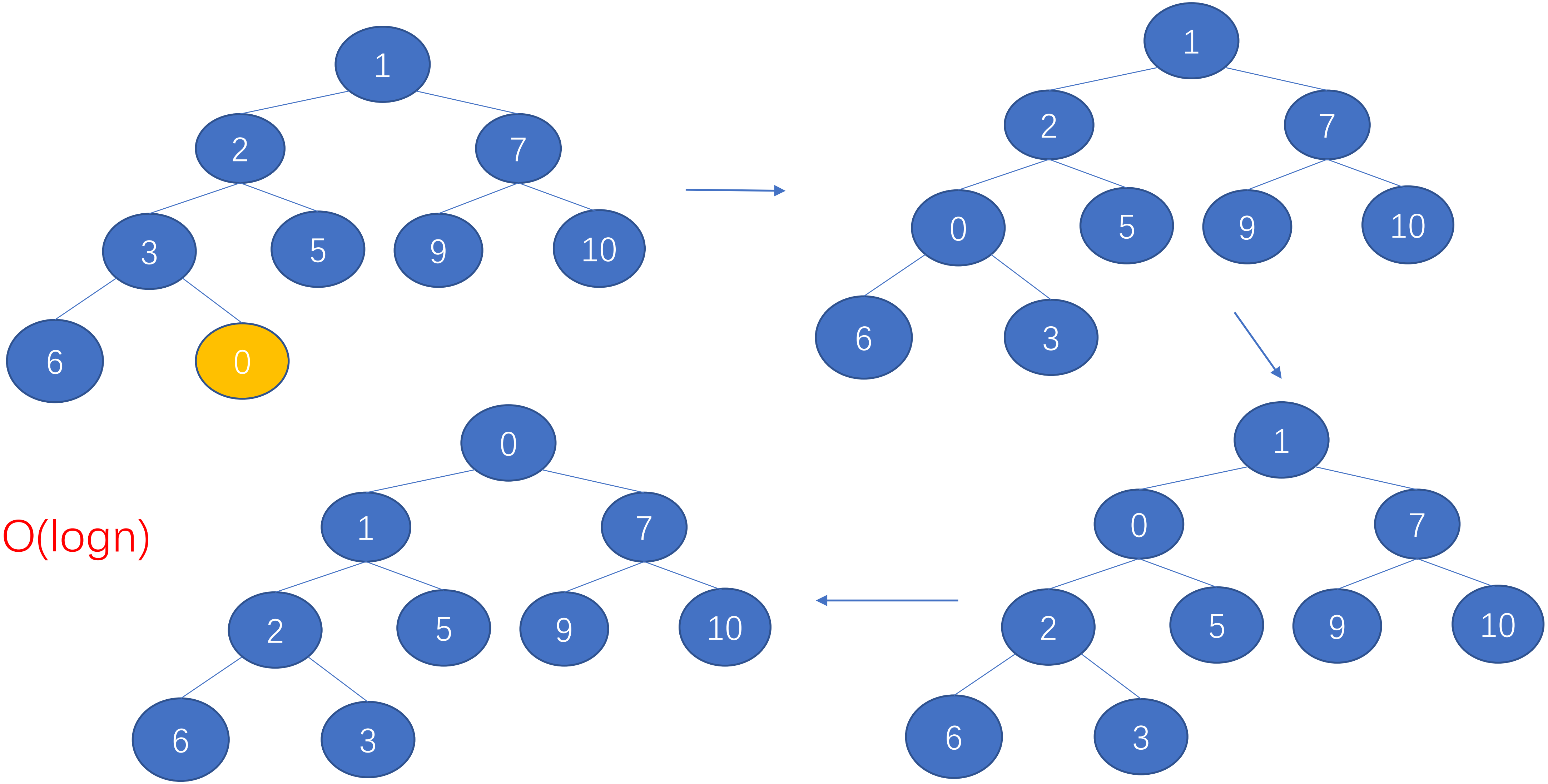
**O(1)**

# Min-heap- extract minimum




O(logn)

# Min-heap- insert




O(logn)

# Reconsider Prim's algorithm

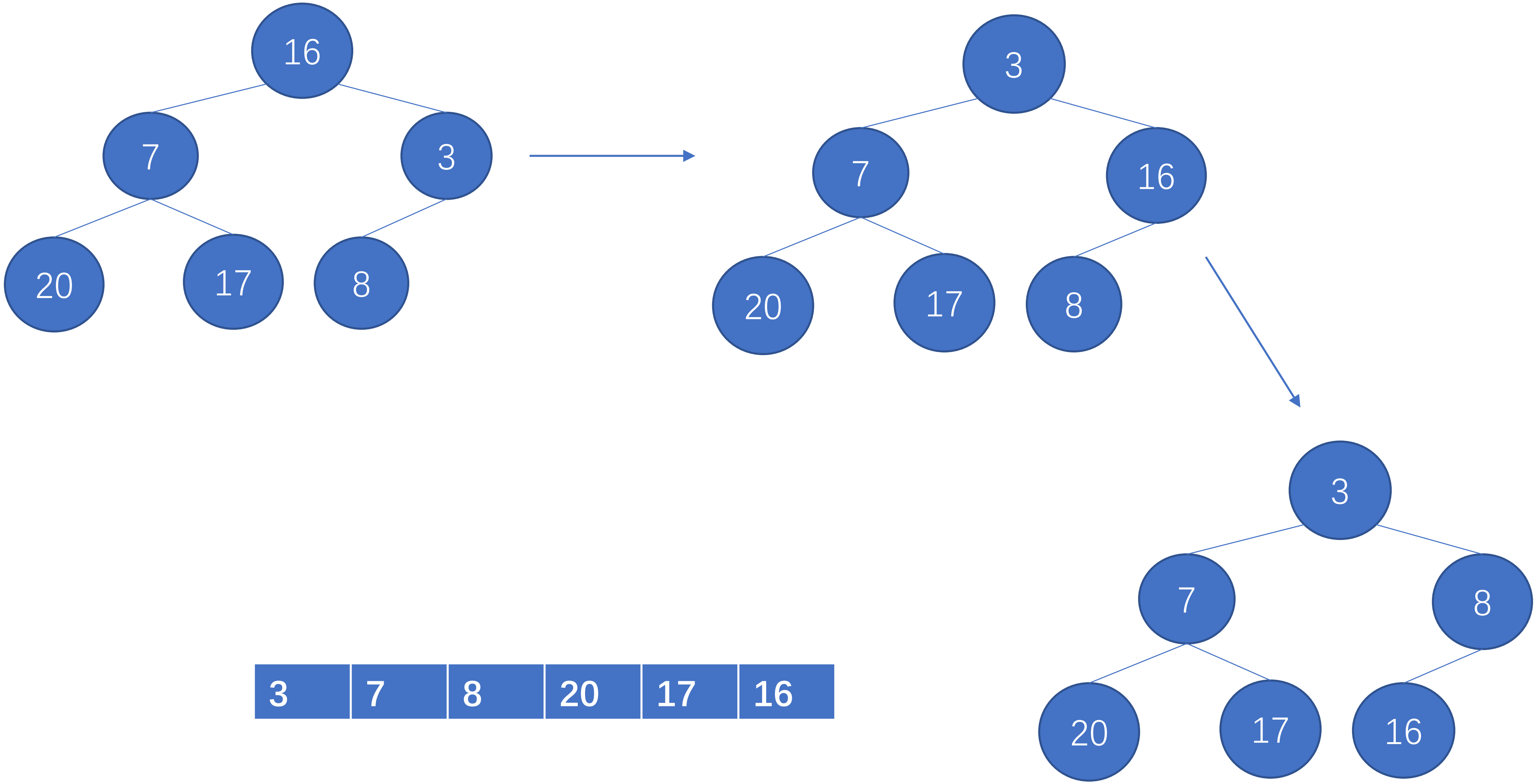
| | | |
|---|---|---|
| *cost* (min-heap) | | |
| | extract-minimum | log\|V\| |
| | insert | log\|V\| |

To maintain *cost*, each edge in the adjacency list is visited |E|

O(|E|log|V|)

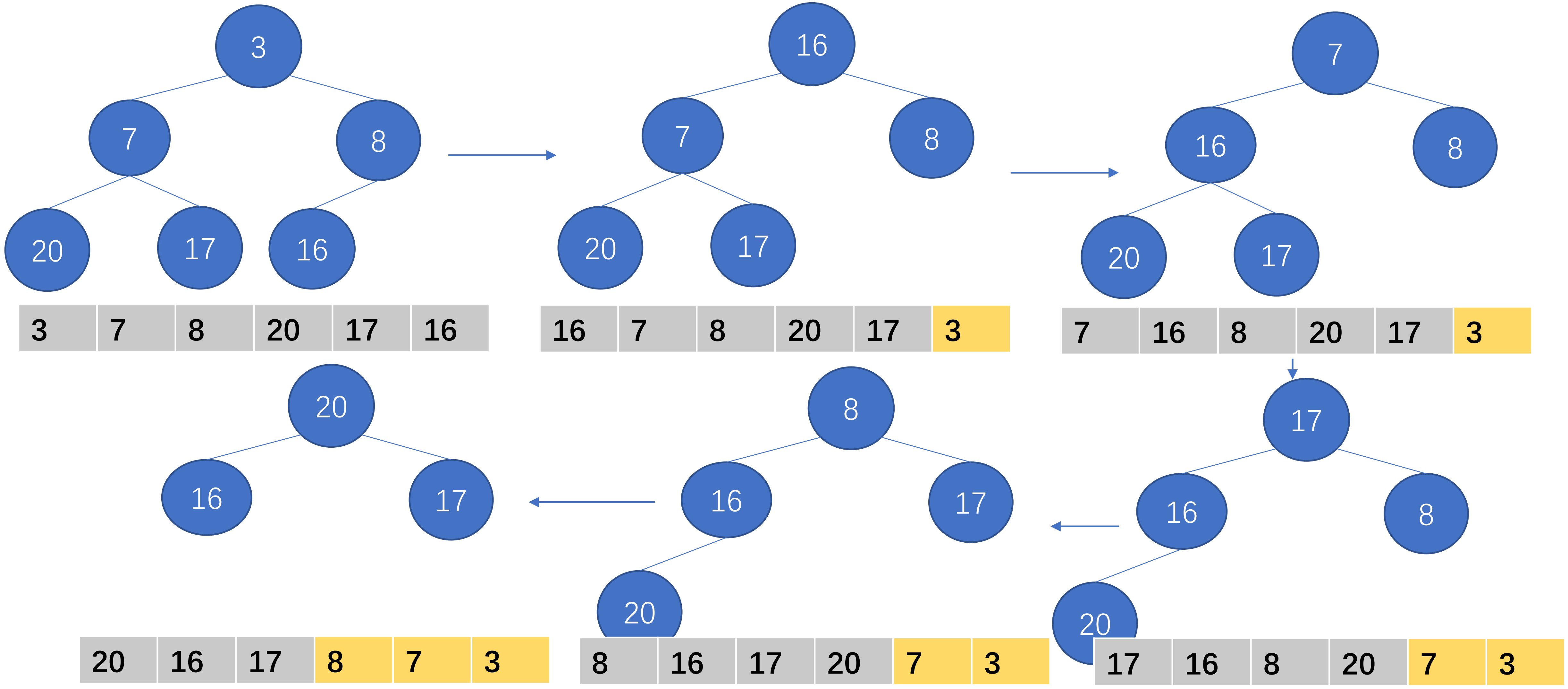
# Heapsort

# Heapsort - Buildup




| | | | | | |
|---|---|---|---|---|---|
| 3 | 7 | 8 | 20 | 17 | 16 |

# Heapsort – extract-minimum



# Heapsort – extract-minimum