Advanced Data Structures

Fibonacci Heaps

Today

- Part 1: Fibonacci Heaps Data Structure

- Agenda:
  1. A quick review on priority queues
  2. What is a Fibonacci Heap ?
  3. Amortized Analysis
  4. Basic Operations
  5. Why Fibonacci ?

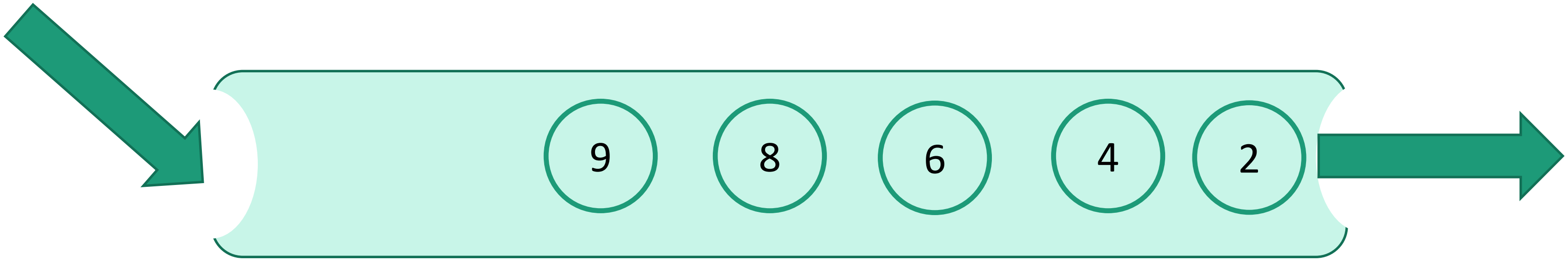
Brief review

for a discussion of priority queues!

Priority Queue

What is the problem Fibonacci heaps try to solve?

A structure like this where we only care about the smallest value at any point is called a priority queue.



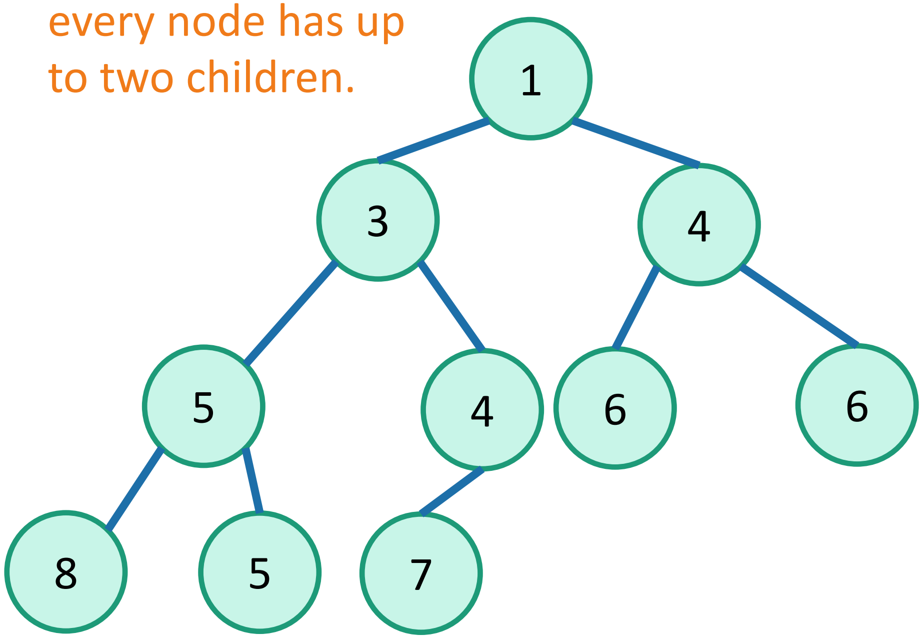
Application : Dijkstra's Algorithm

Binary Heap

What is the problem Fibonacci heaps try to solve?

A binary heap is a binary tree where each node stores exactly one element with its key.

every node has up to two children.



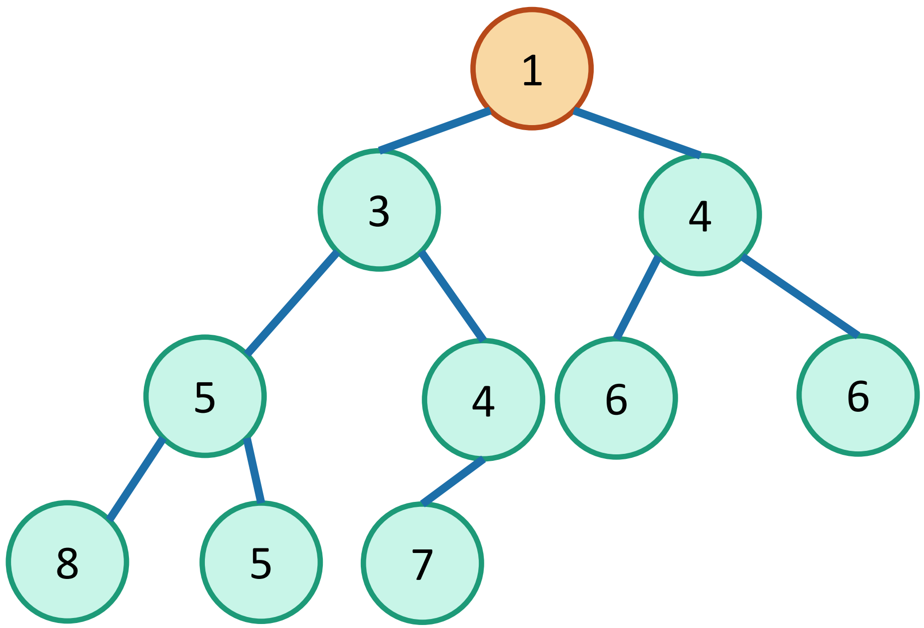
1. Every level is full
   - Last one from left to right

2. Heap property
   - No child is smaller than it’s parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

Root node : the smallest value is always the root.

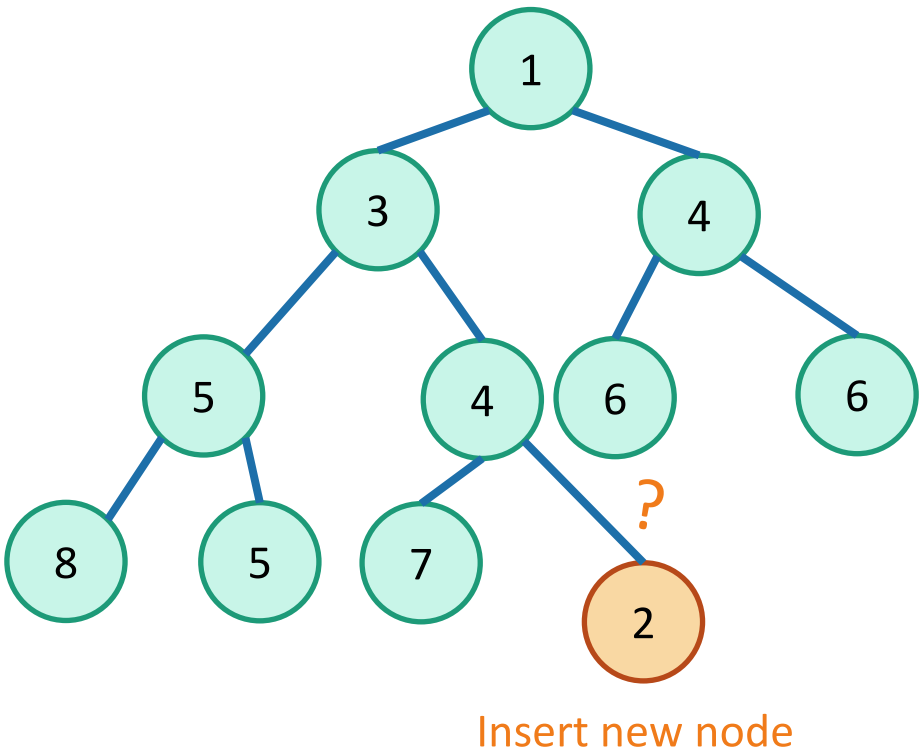


1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it’s parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

How do we insert a new element into the heap when a new message arrives?

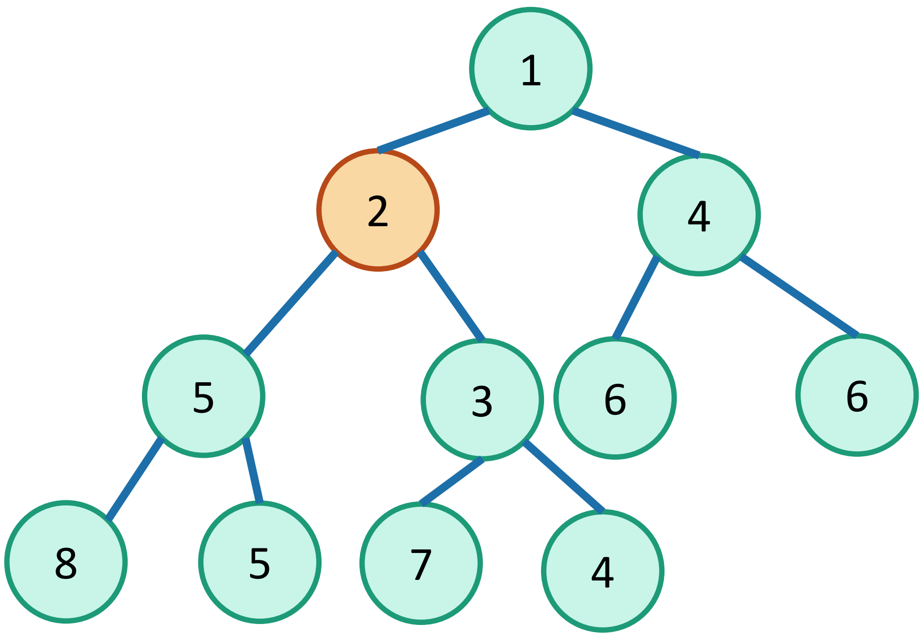


1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it’s parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

How do we insert a new element into the heap when a new message arrives?



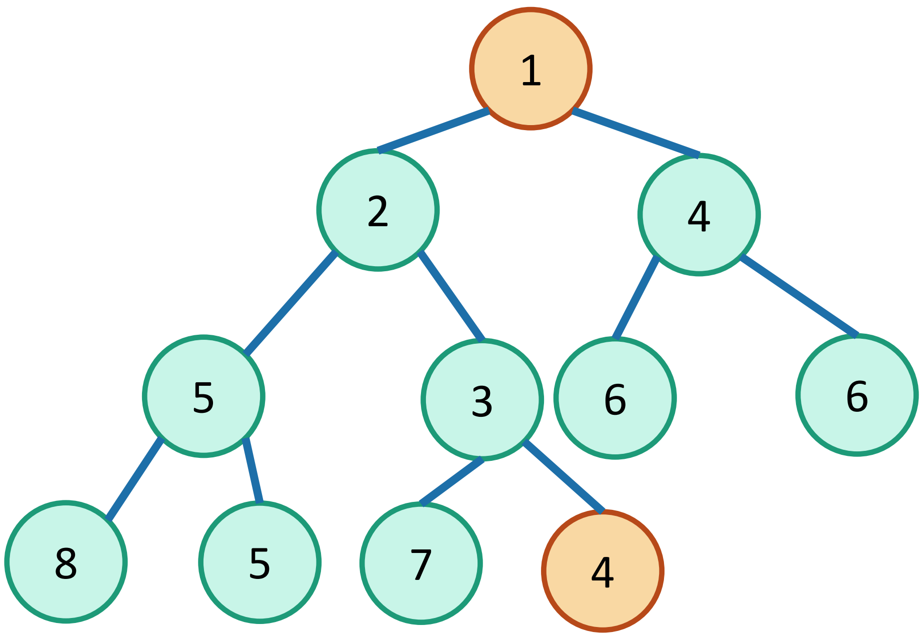
Insert at the last level from left , bubbling up the new element until property 2 is restored.

1. Every level is full
    - Last one from left to right
2. Heap property
    - No child is smaller than it's parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

What happens when we want to send the most important message?



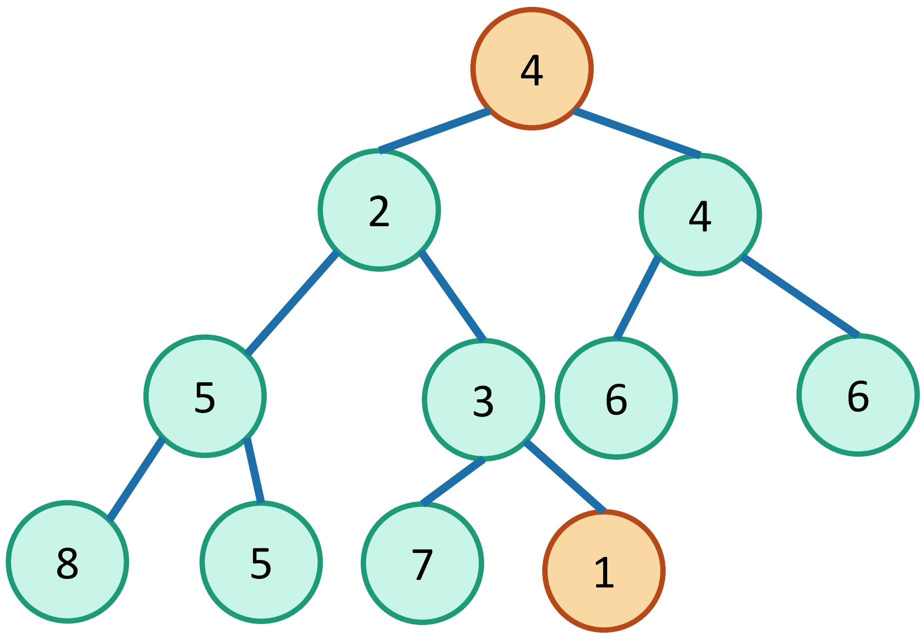
ExtractMin Operation : swap the root with the right-most element of the last level. Delete, bubble up.

1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it's parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

What happens when we want to send the most important message?



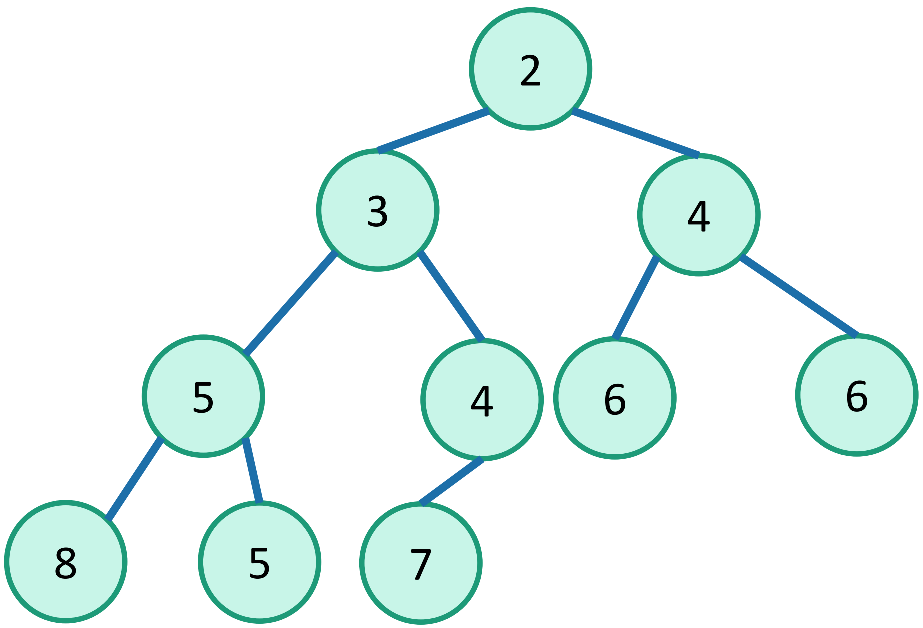
ExtractMin Operation : swap the root with the right-most element of the last level. Delete node, heapify.

1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it's parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

What happens when we want to send the most important message?



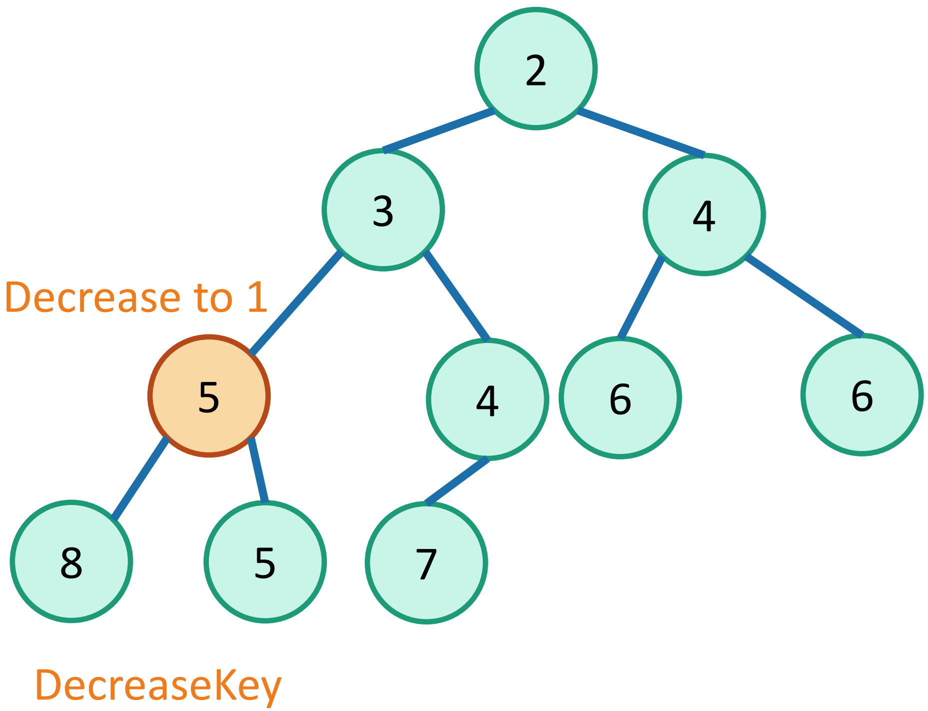
ExtractMin Operation : swap the root with the right-most element of the last level. Delete node, heapify.

1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it’s parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

What happens when we want to change priorities of our elements ?

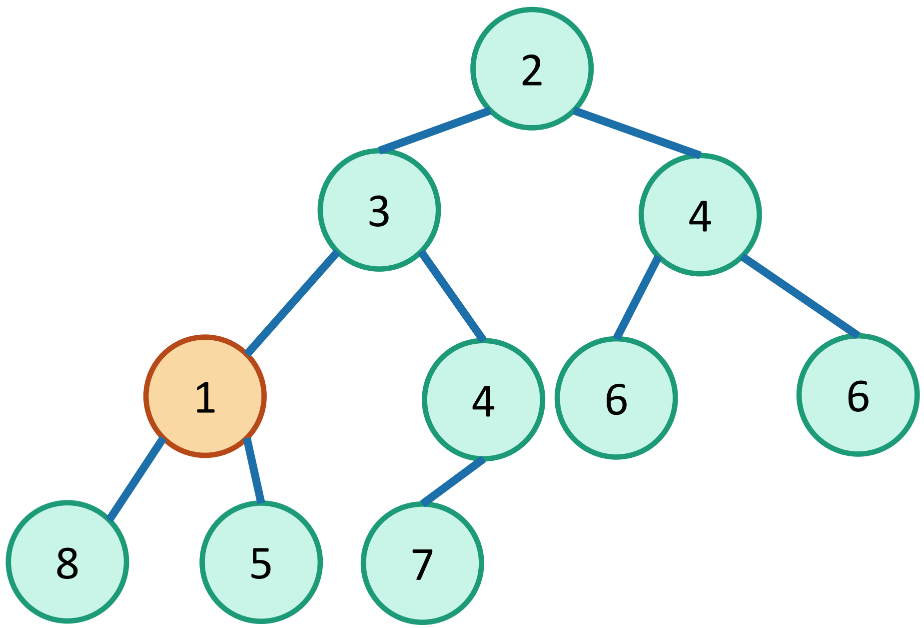


1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it’s parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

What happens when we want to change priorities of our elements ?



DecreaseKey : increase the priority, heapify

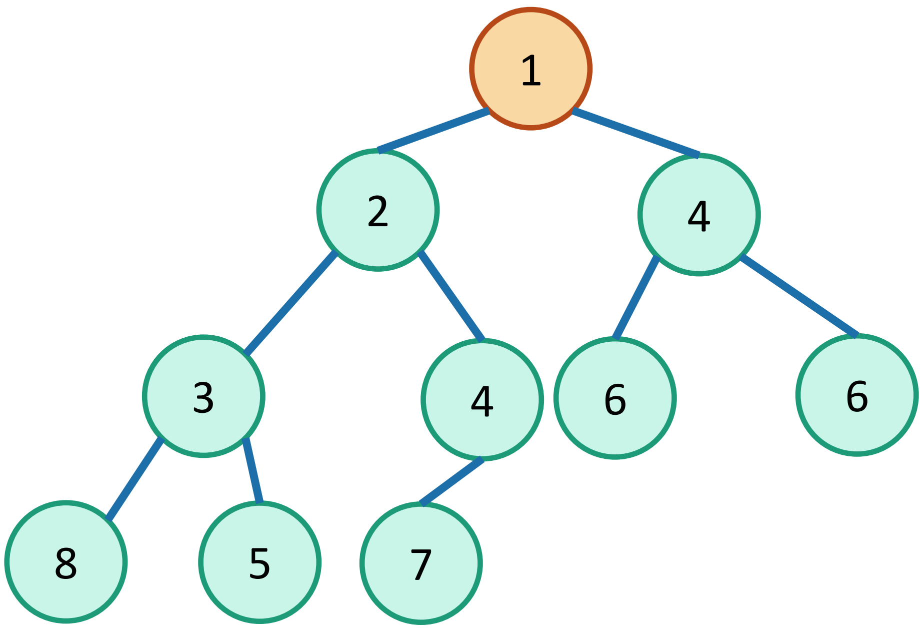
1. Every level is full
   - Last one from left to right

2. Heap property
   - No child is smaller than it's parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

What happens when we want to change priorities of our elements ?



DecreaseKey : increase the priority, heapify

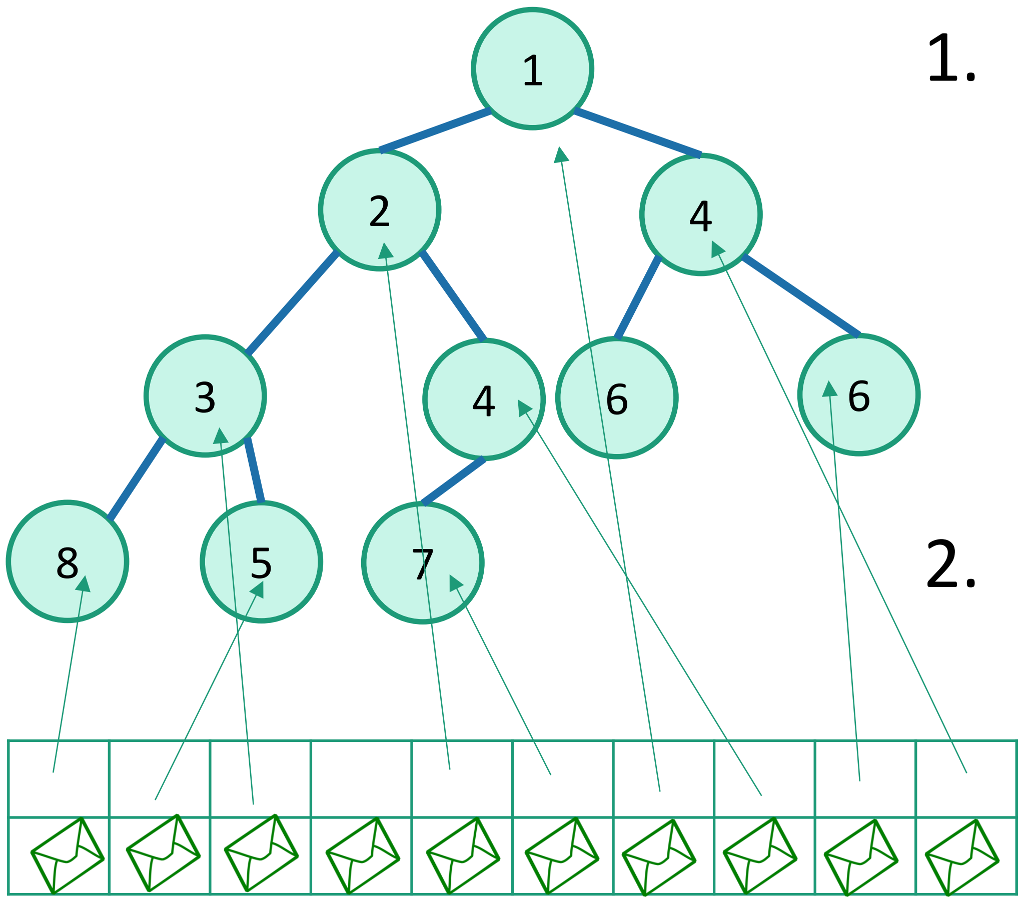
1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it's parent

Binary Heap

What is the problem Fibonacci heaps try to solve?

What happens when we want to change priorities of our elements ?

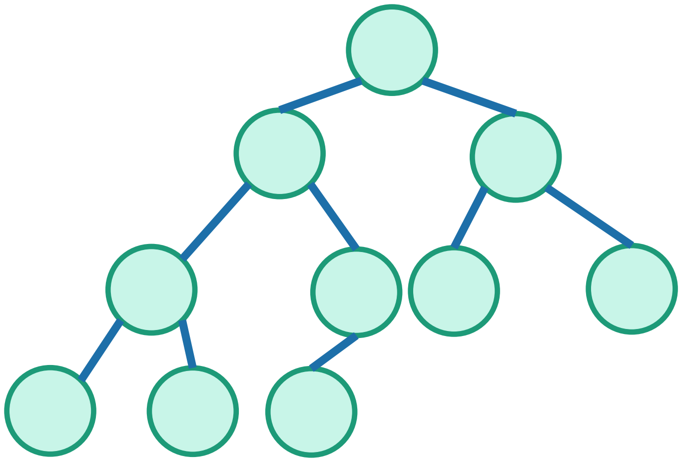
DecreaseKey : How to find the node ? Lookup table



1. Every level is full
   - Last one from left to right
2. Heap property
   - No child is smaller than it's parent

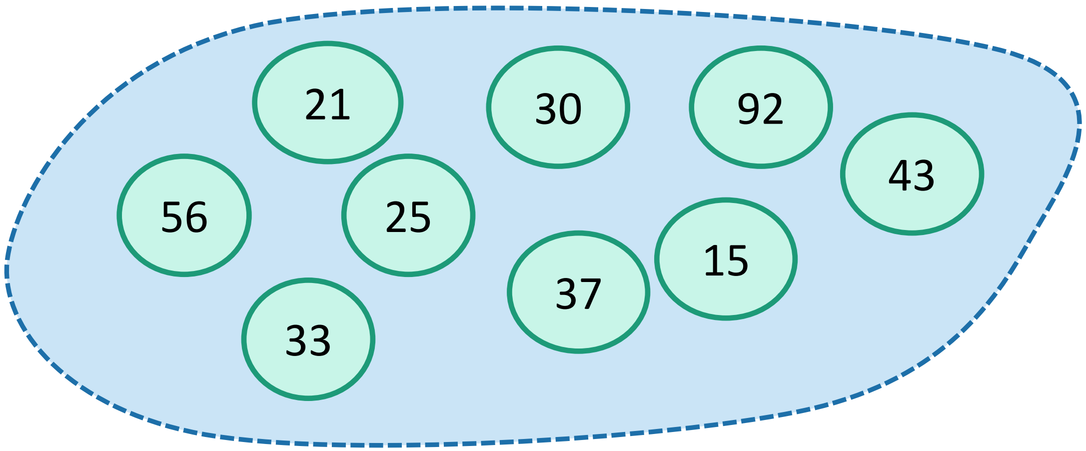
Operations: Priority Queue

1. GetMin : *O(1)*
2. Insert : *O(logn)*
3. ExtractMin : *O(logn)*
4. DecreaseKey : *O(logn)*



Operations: Priority Queue

1. GetMin : *O(1) : O(1)*
2. Insert : *O(logn) : O(1)*
3. ExtractMin : *O(logn) : O(???)*
4. DecreaseKey : *O(logn) : O(1)*



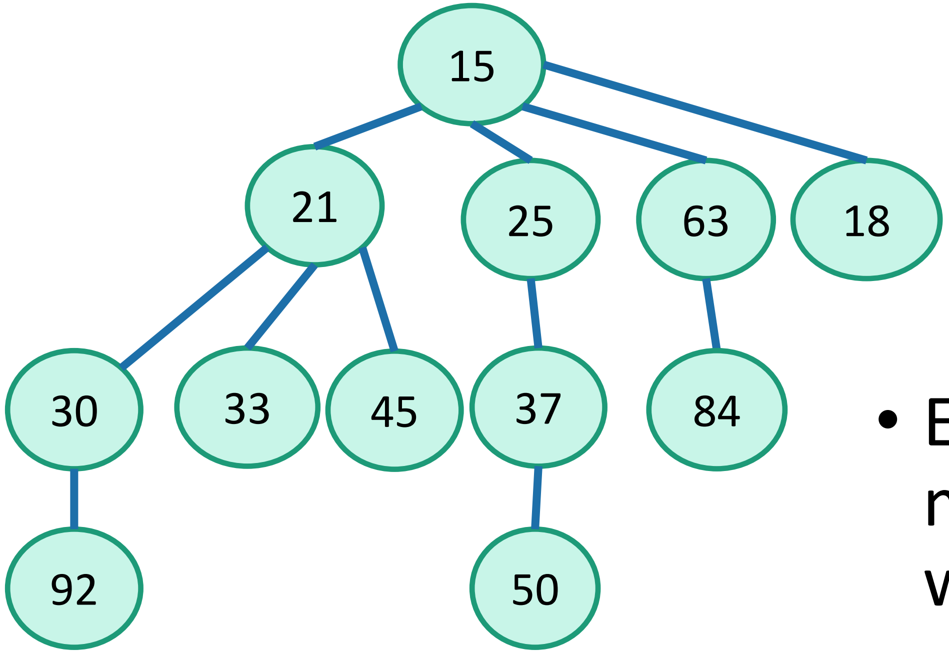
End review

Back to Fibonacci Heaps!

Fibonacci Heap

What is the problem Fibonacci heaps try to solve?

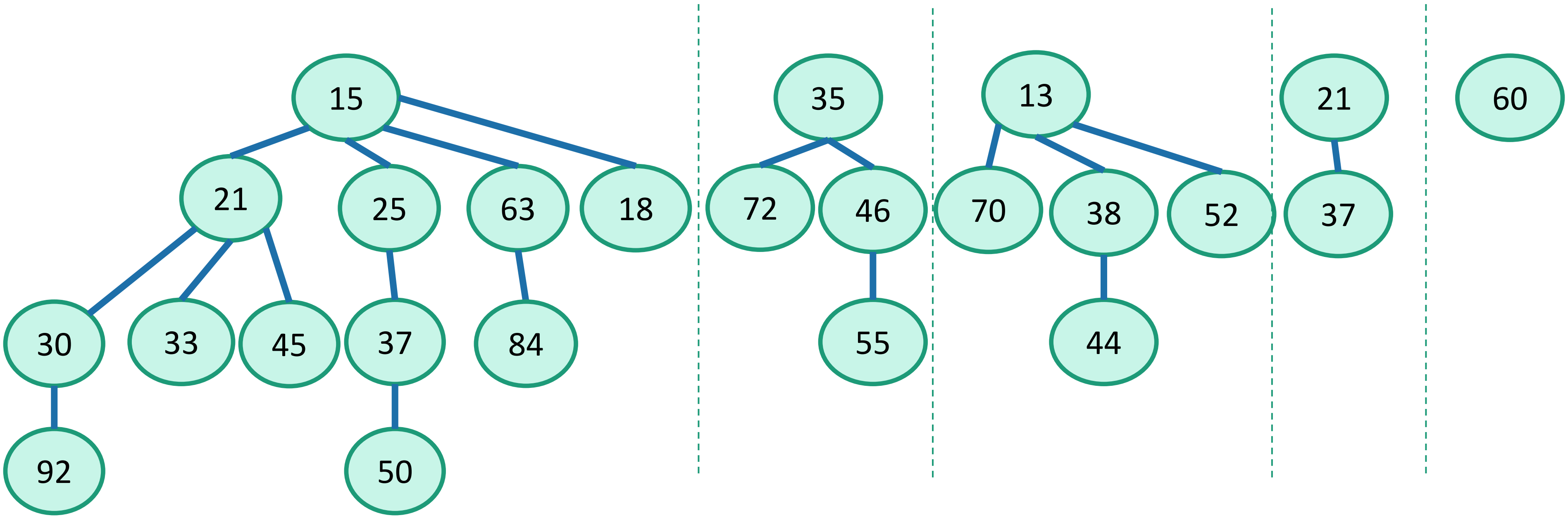
Based on binary heap but looser structure



- Every node can have as many children as it wants.

Fibonacci Heap

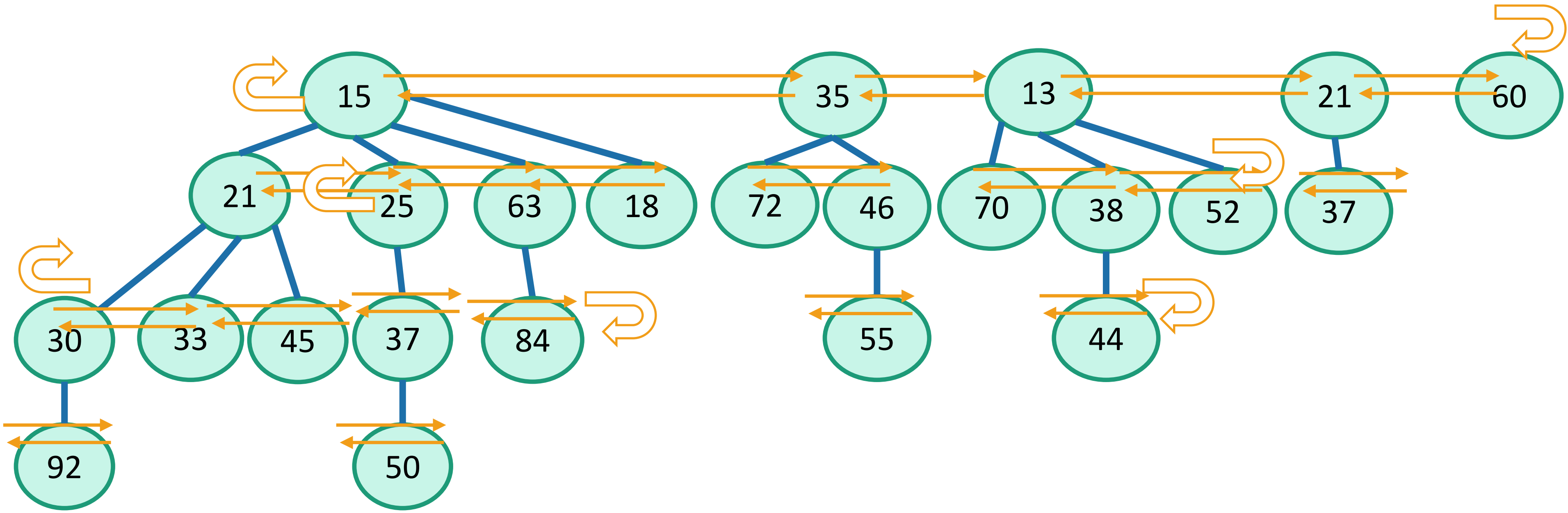
What is the problem Fibonacci heaps try to solve?



- Allow the heap to contain multiple trees

Fibonacci Heap

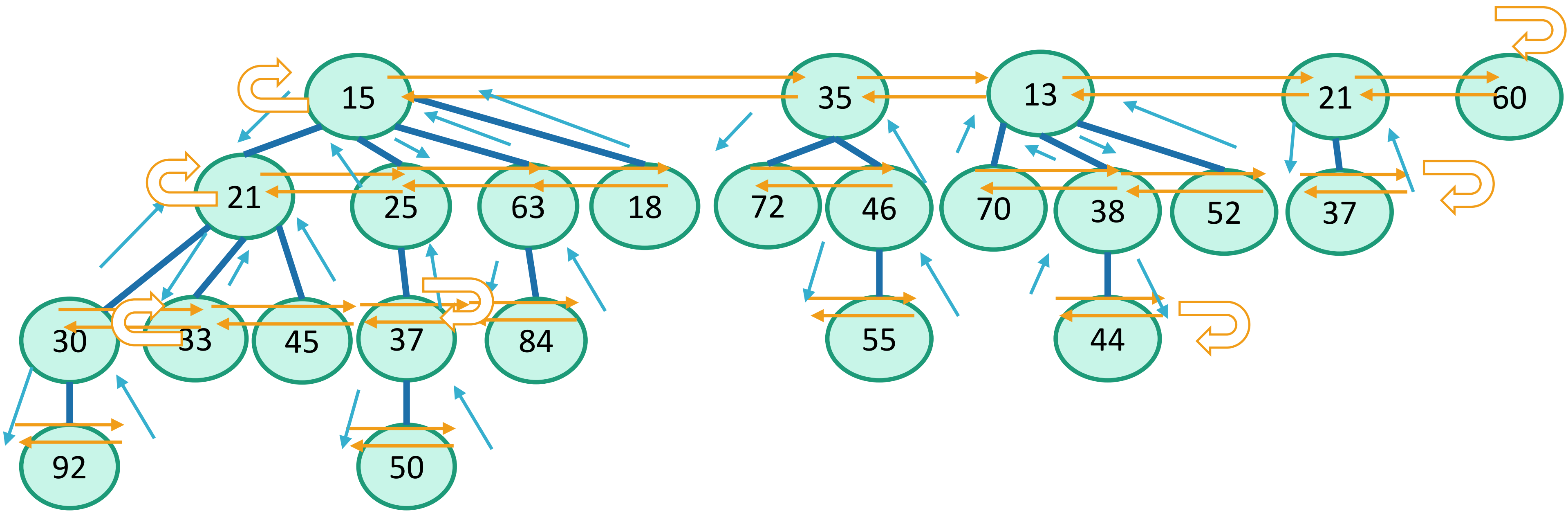
What is the problem Fibonacci heaps try to solve?



- Allow the heap to contain multiple trees ; store in circular doubly linked list

Fibonacci Heap

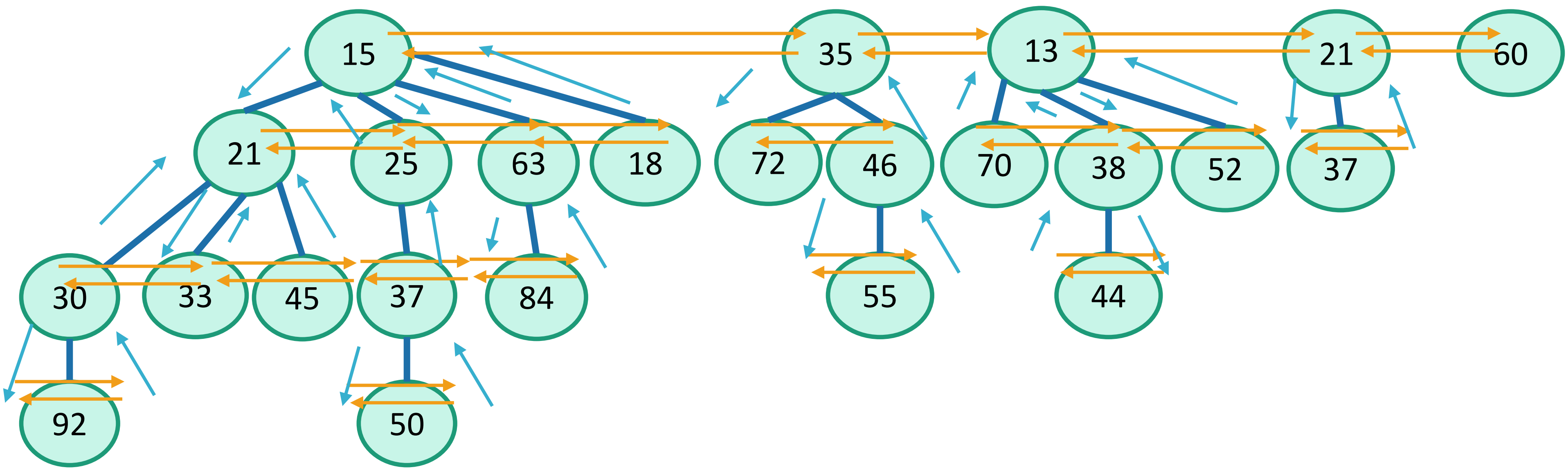
What is the problem Fibonacci heaps try to solve?



- Allow the heap to contain multiple trees ; store in circular doubly linked list
- Additional references : point up and down

Fibonacci Heap

What is the problem Fibonacci heaps try to solve?

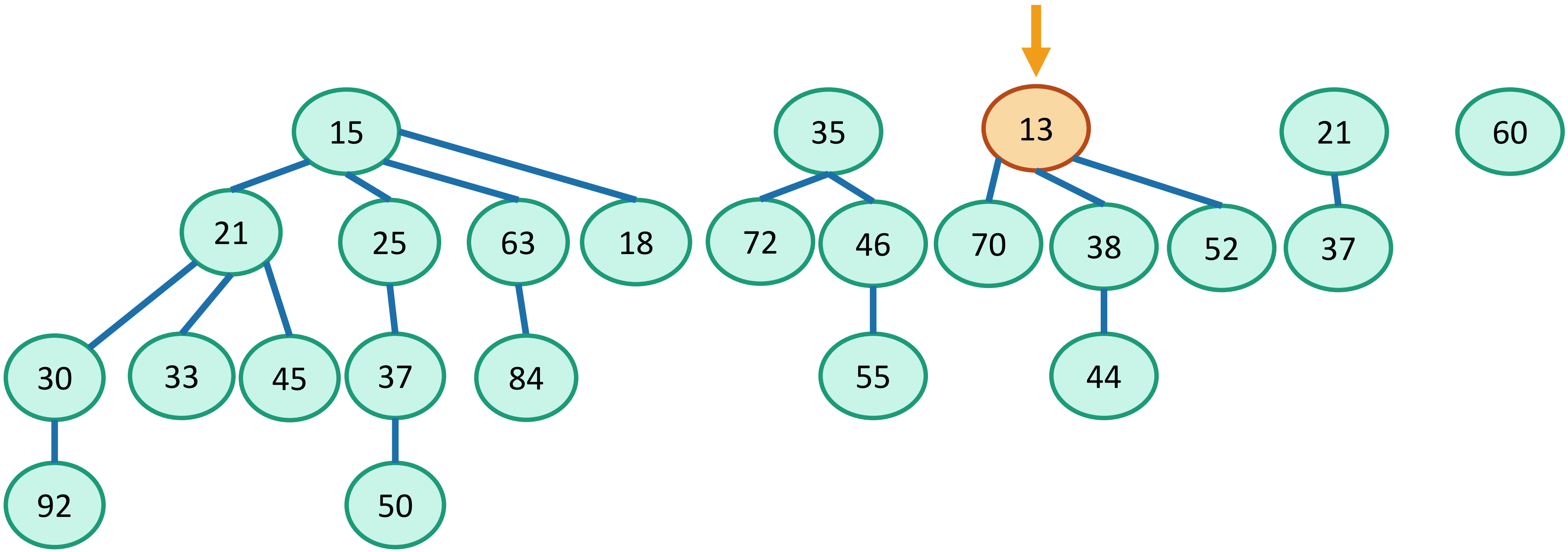


- Heap property

  No child is smaller than it's parent

Fibonacci Heap

What is the problem Fibonacci heaps try to solve?



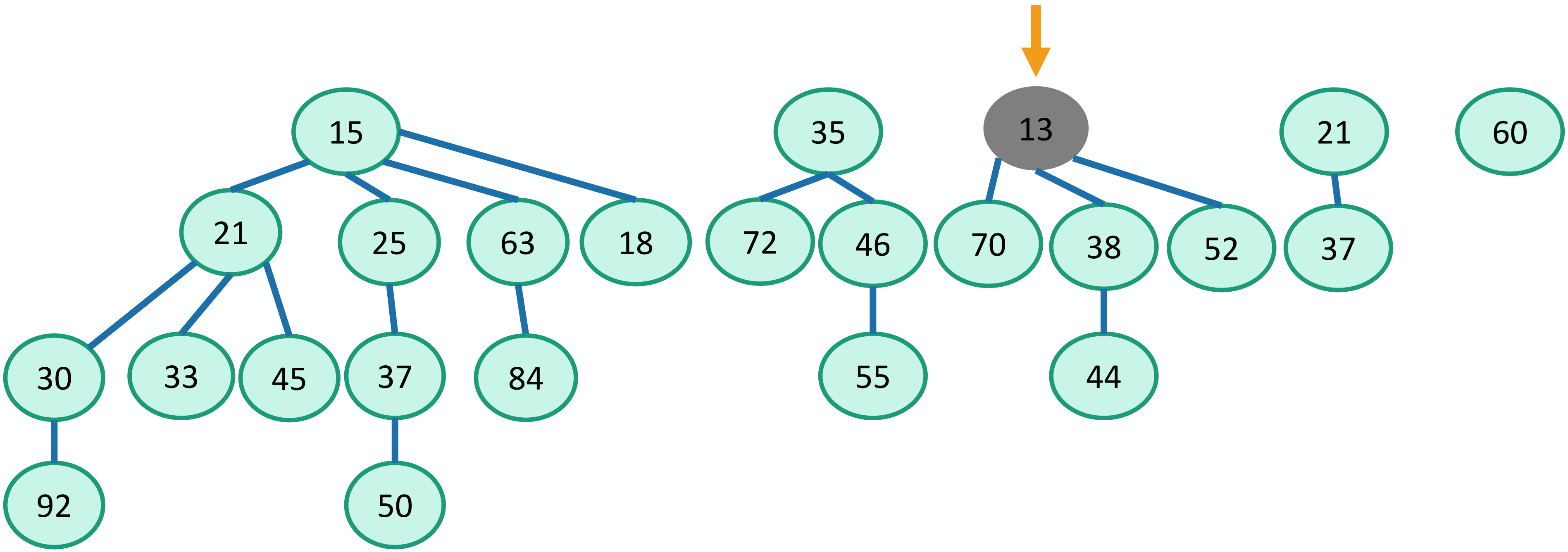
- Heap property

  No child is smaller than it's parent

smallest element of the heap is always one of the roots.

Fibonacci Heap

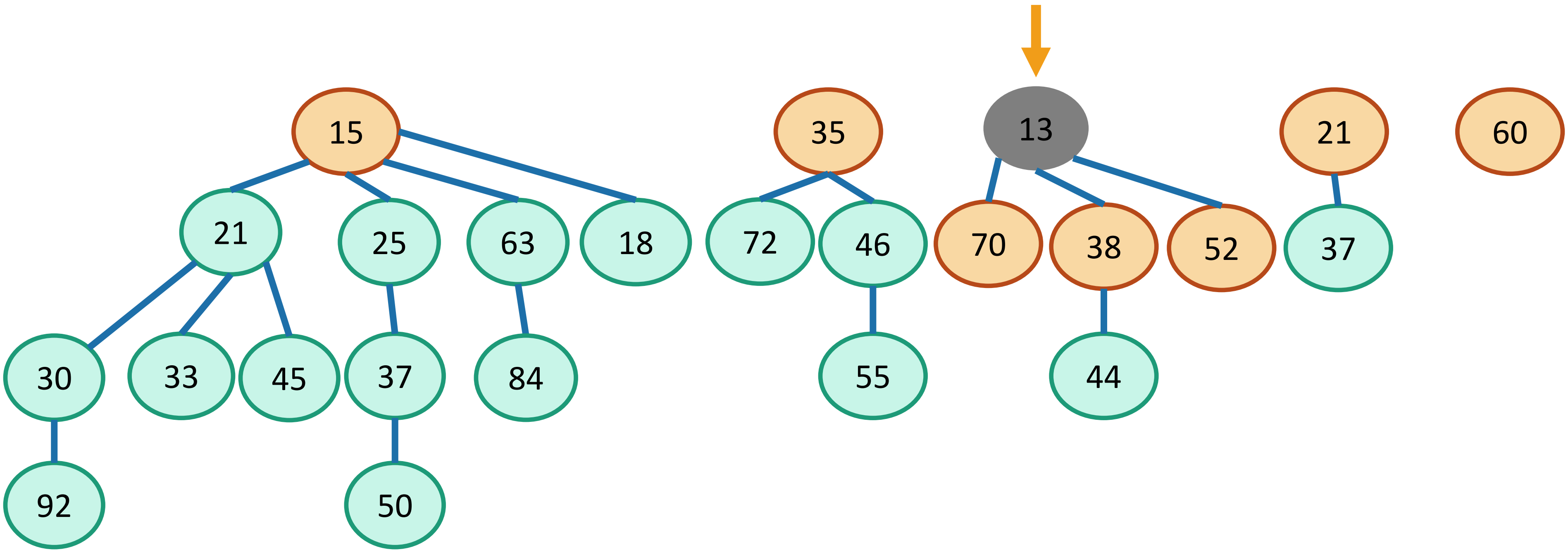
What is the problem Fibonacci heaps try to solve?



If we remove the smallest element,
Which element is the next smallest one?

Fibonacci Heap

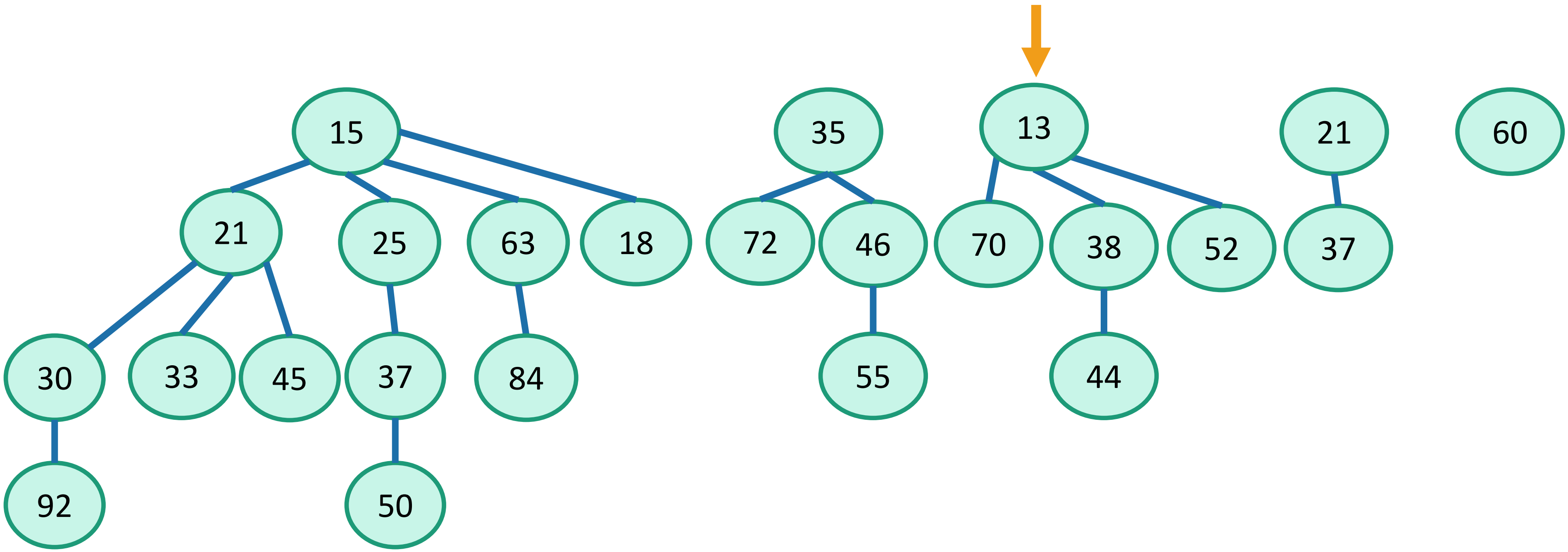
What is the problem Fibonacci heaps try to solve?



If we remove the smallest element,
Which element is the next smallest one?

Fibonacci Heap

Operations : GetMin

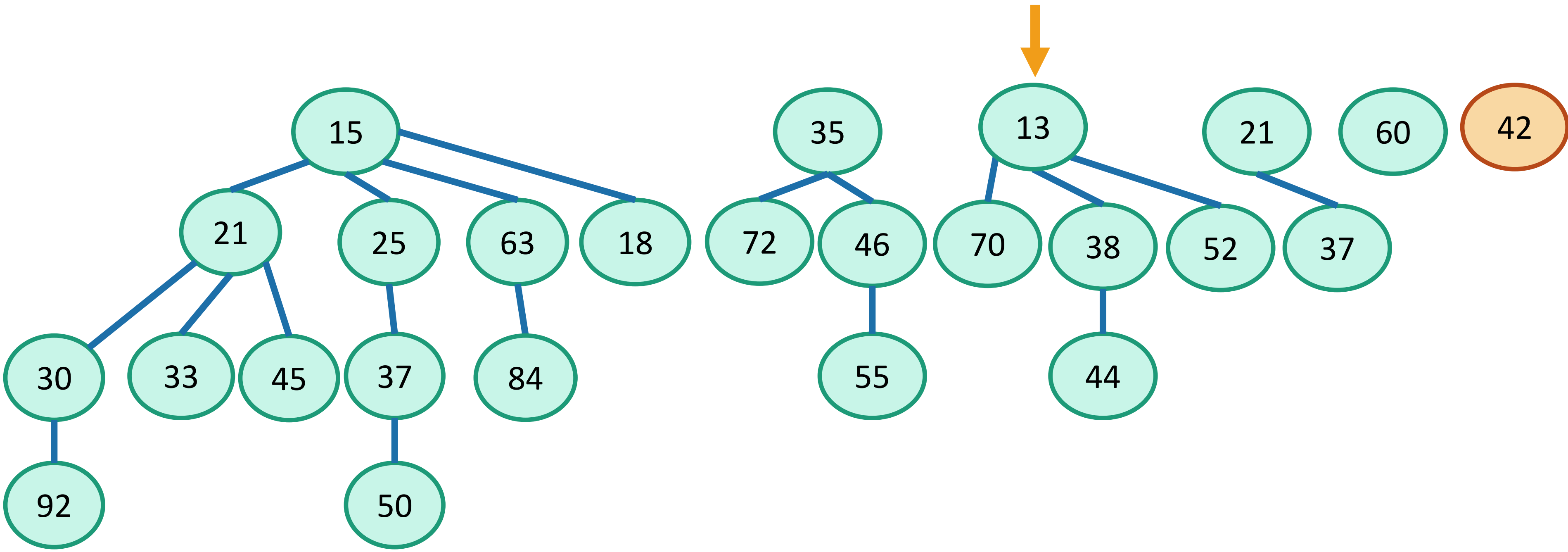


GetMin

O(1)

Fibonacci Heap

Operations : Insert

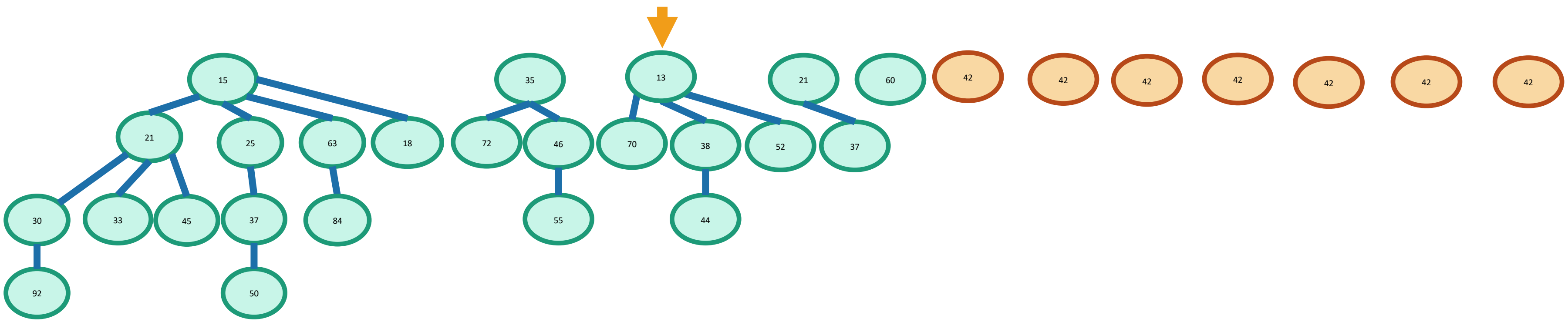


GetMin
O(1)

Insert
O(1)

Fibonacci Heap

Operations : Insert

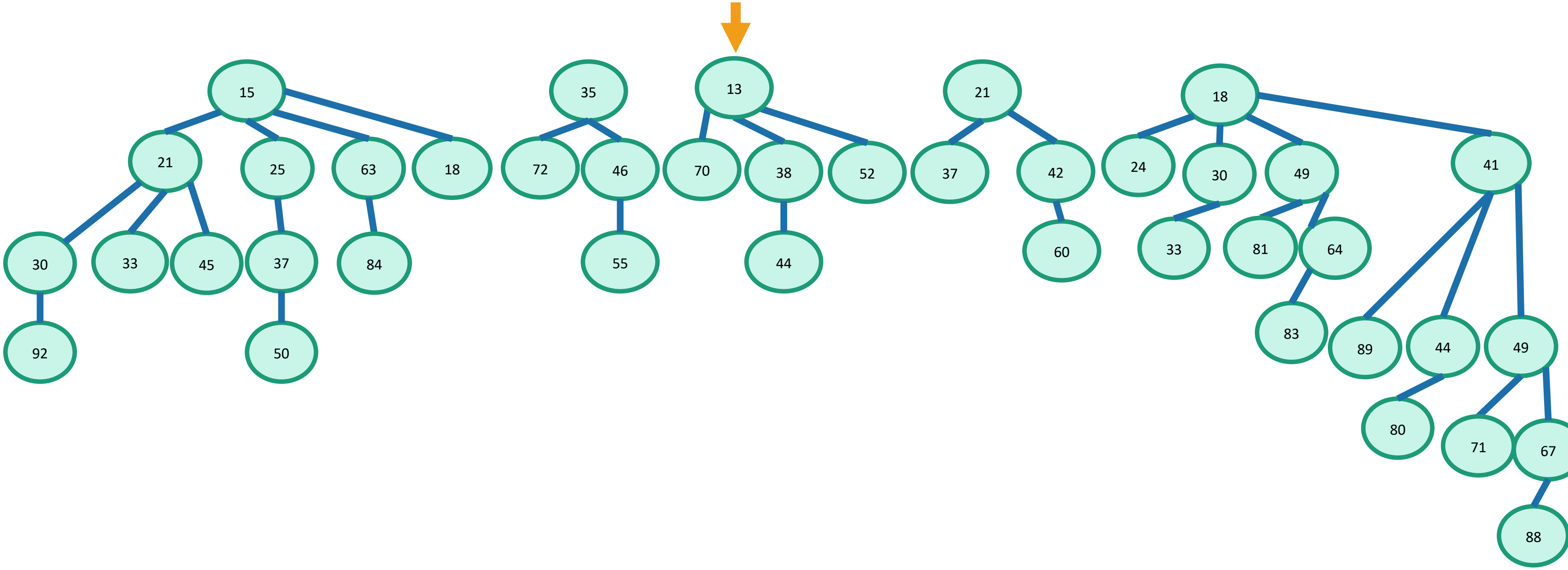


GetMin
O(1)

Insert
O(1)

Fibonacci Heap

Operations : Insert



GetMin
O(1)

Insert
O(1)

Fibonacci Heap

Amortized Analysis

Say :

- Insert : work 1
- → #roots +=1

- ExtractMin : work # roots + 10
- → #roots =5

100	• Insert	1
	• Insert	1
	• …	…
	• Insert	1
	• ExtractMin	110
	• ExtractMin	15
	• ExtractMin	15
50	• Insert	1
	• Insert	1
	• …	…
	• Insert	1
	• ExtractMin	65
	• ExtractMin	15

Fibonacci Heap

Amortized Analysis

- Insert : work 1
- → #roots +=1

- ExtractMin : work # roots + 10
- → #roots =5

ExtractMin : *O(15)*

Insert : *O(1)*

100	Insert	2
	Insert	2
	…	…
	Insert	2
	ExtractMin	10
	ExtractMin	15
	ExtractMin	15
50	Insert	2
	Insert	2
	…	…
	Insert	2
	ExtractMin	15
	ExtractMin	15

Fibonacci Heap

Amortized Analysis

- Insert : work 1
- → #roots +=1

- ExtractMin : work # roots + 10
- → #roots =5

ExtractMin : *O(5 + 10)*

Insert : *O(1)*

100

- Insert 2
- Insert 2
- … …
- Insert 2
- ExtractMin 10
- ExtractMin 15
- ExtractMin 15

50

- Insert 2
- Insert 2
- … …
- Insert 2
- ExtractMin 15
- ExtractMin 15

Fibonacci Heap

Amortized Analysis

- Insert : work 1
- → #roots +=1

- ExtractMin : work # roots + 10
- → #roots =5

ExtractMin : *O(#root after +extra work)*

Insert : *O(1)*

100

- Insert 2
- Insert 2
- … …
- Insert 2

- ExtractMin 10
- ExtractMin 15
- ExtractMin 15

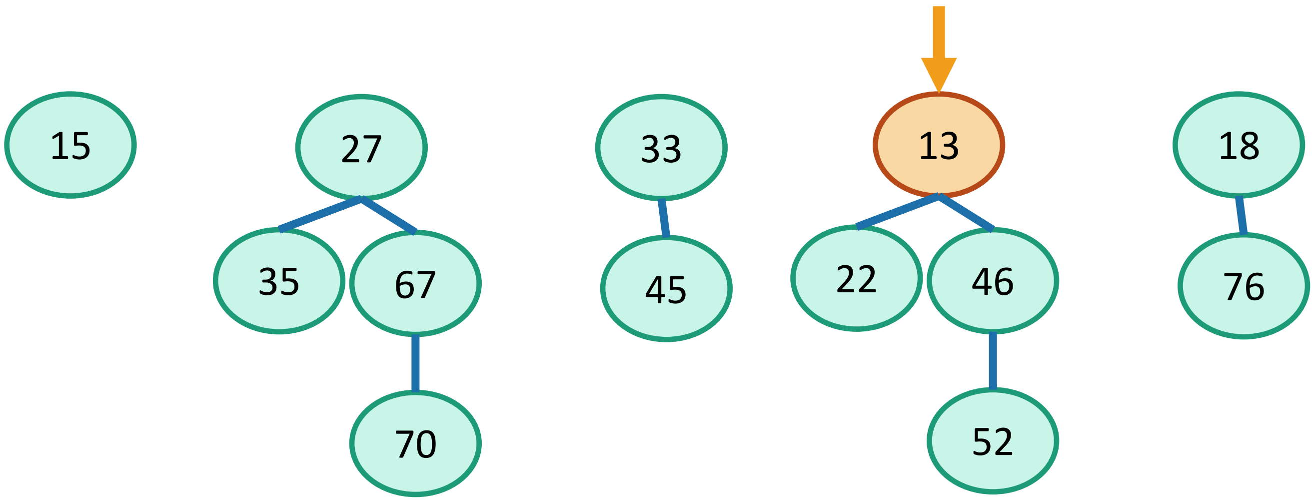
50

- Insert 2
- Insert 2
- … …
- Insert 2

- ExtractMin 15
- ExtractMin 15

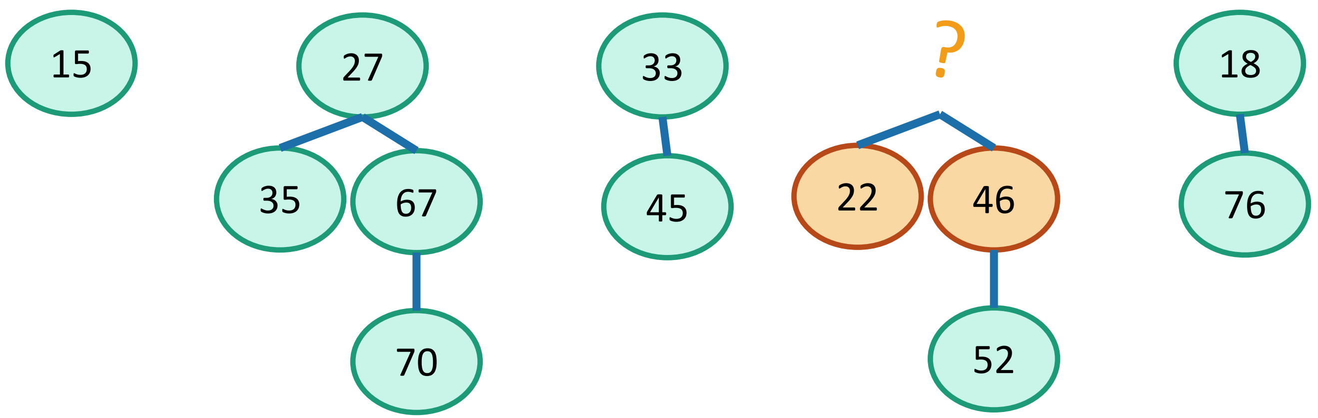
Fibonacci Heap

Operations : ExtractMin



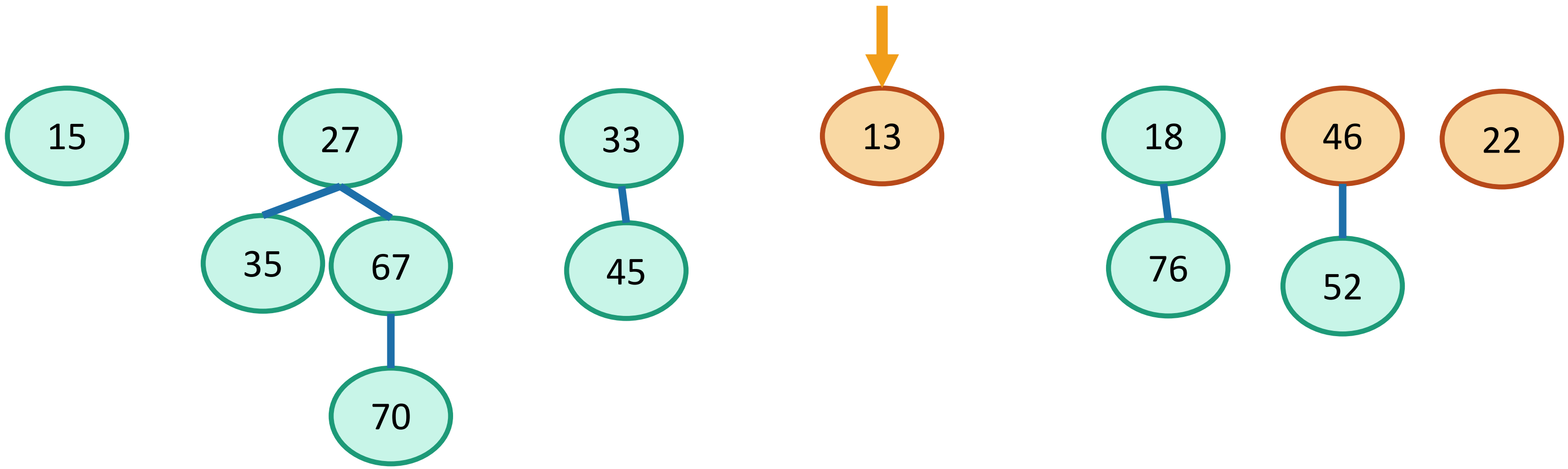
Fibonacci Heap

Operations : ExtractMin



Fibonacci Heap

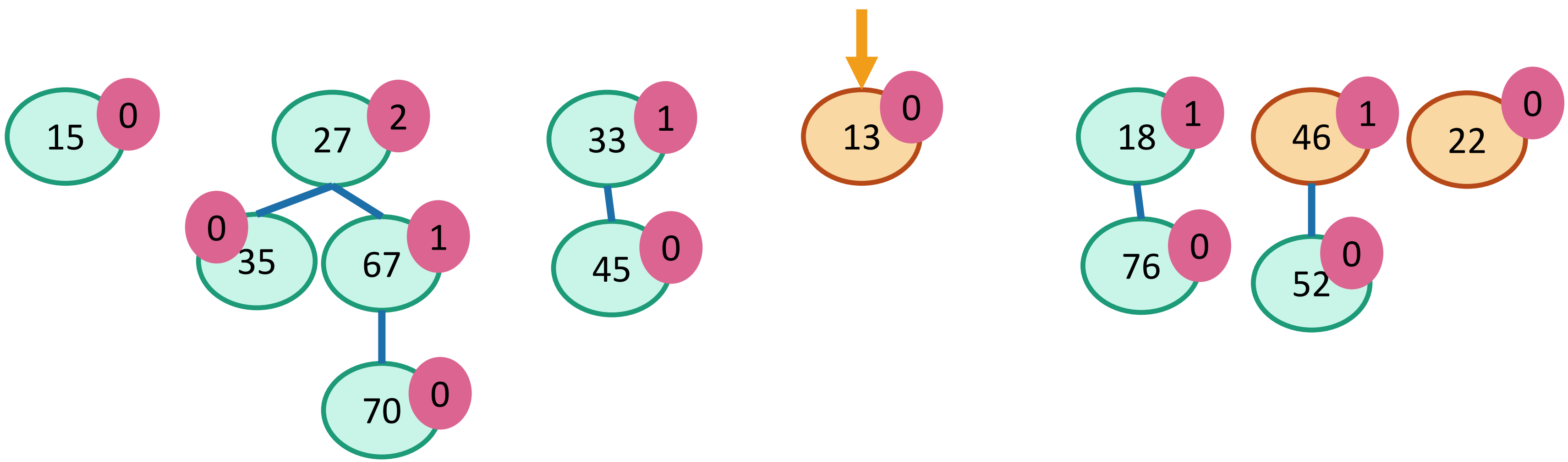
Operations : ExtractMin



- Remove Minimum

Fibonacci Heap

Operations : ExtractMin



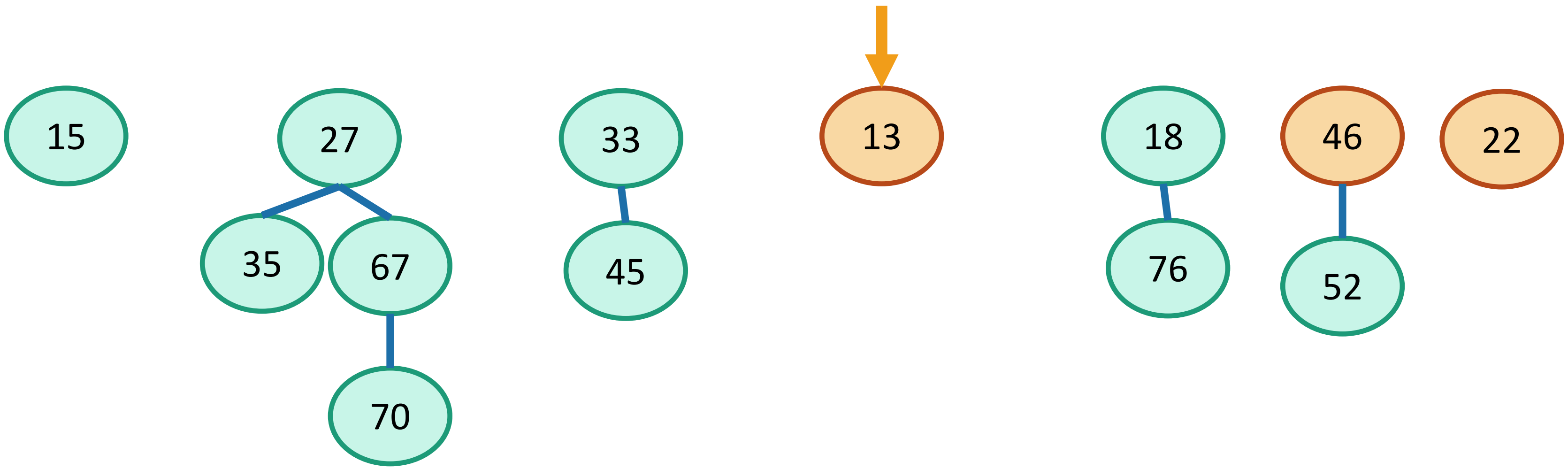
- Remove Minimum

↓ # trees

↓ # of children ~ degree

Fibonacci Heap

Operations : ExtractMin

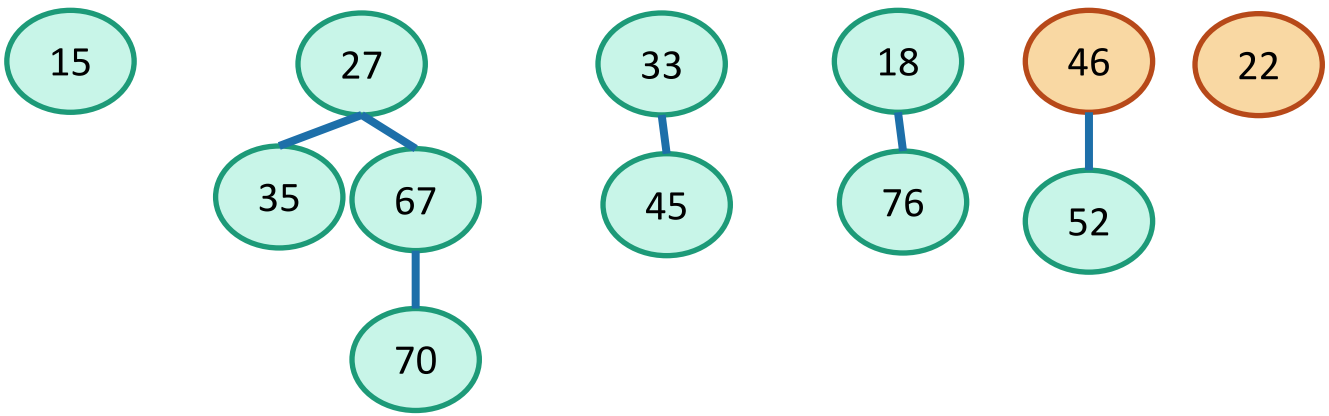


- Remove Minimum
  O (maximum degree)

↓ # trees
↓ # max degree

Fibonacci Heap

Operations : ExtractMin



↓ # trees

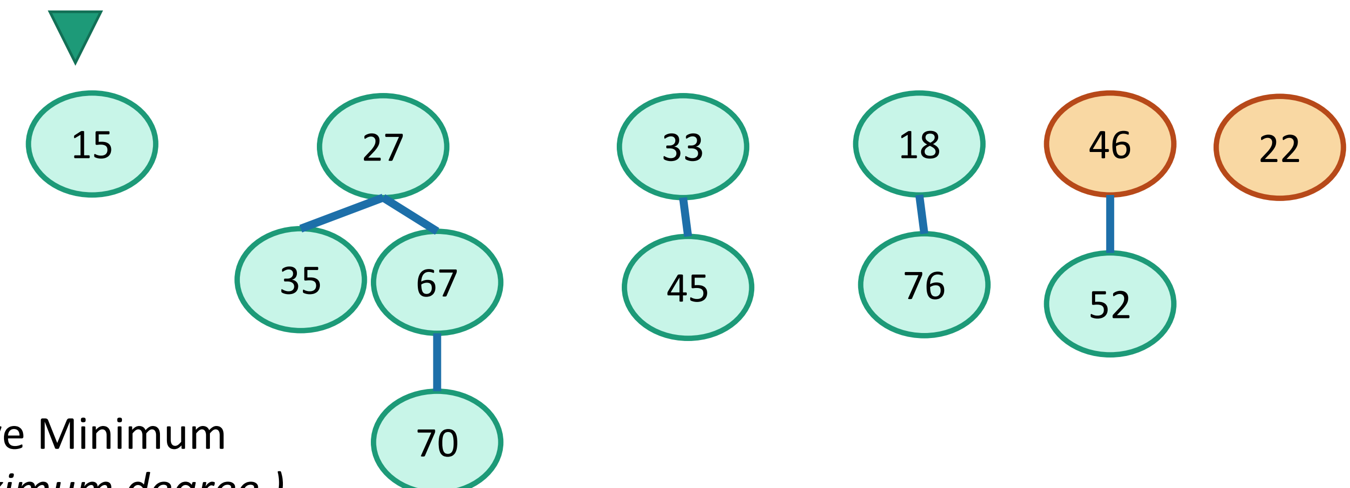
↓ # max degree

- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

Operations : ExtractMin

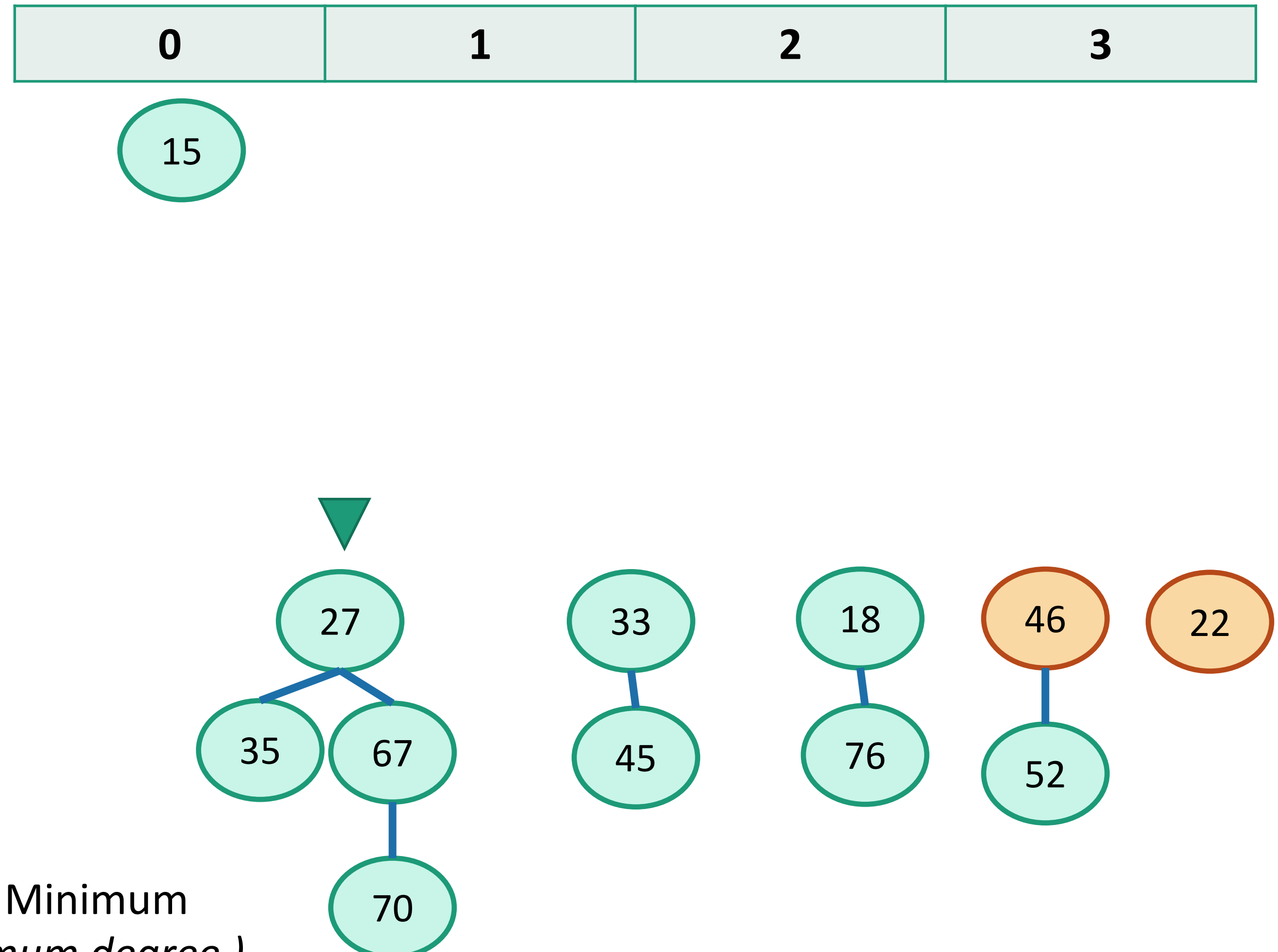
0	1	2	3



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

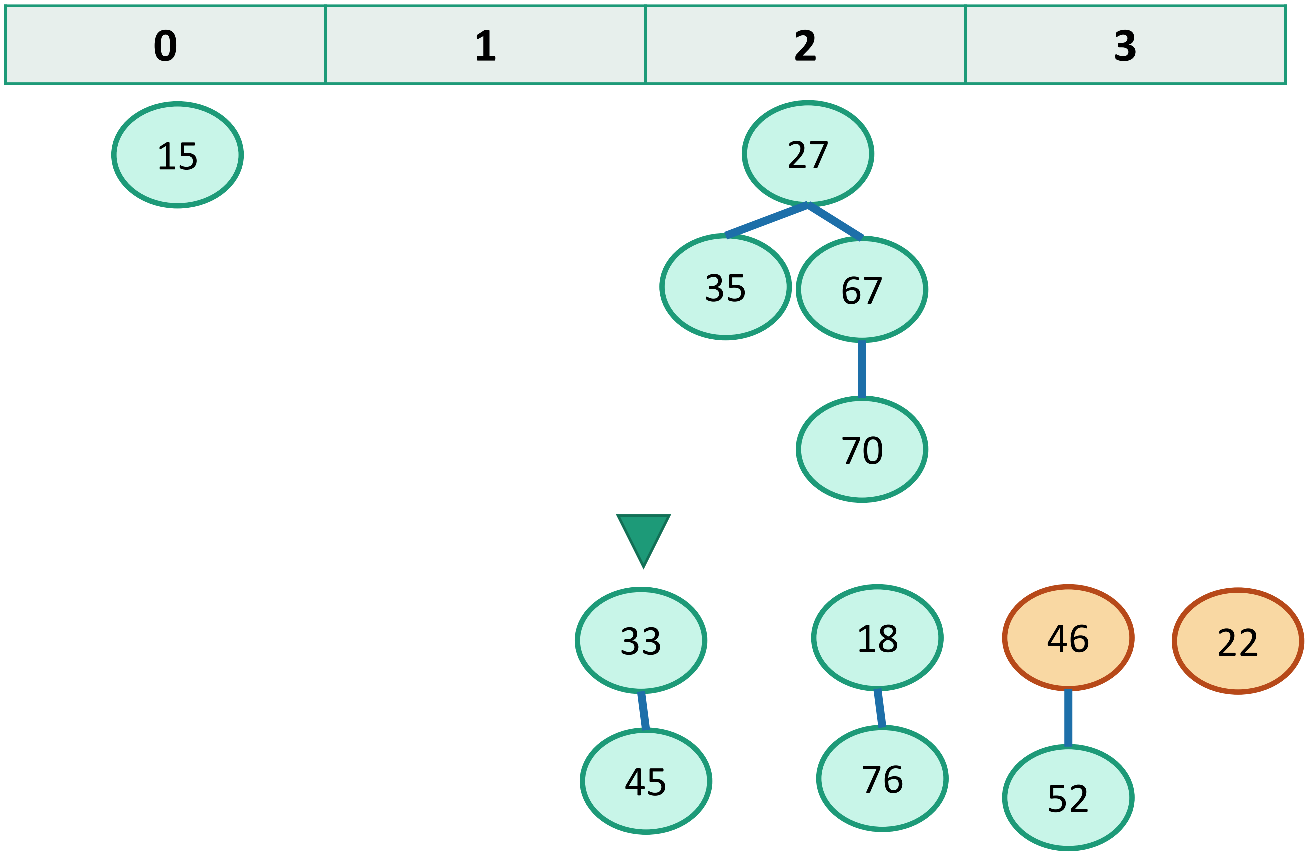
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

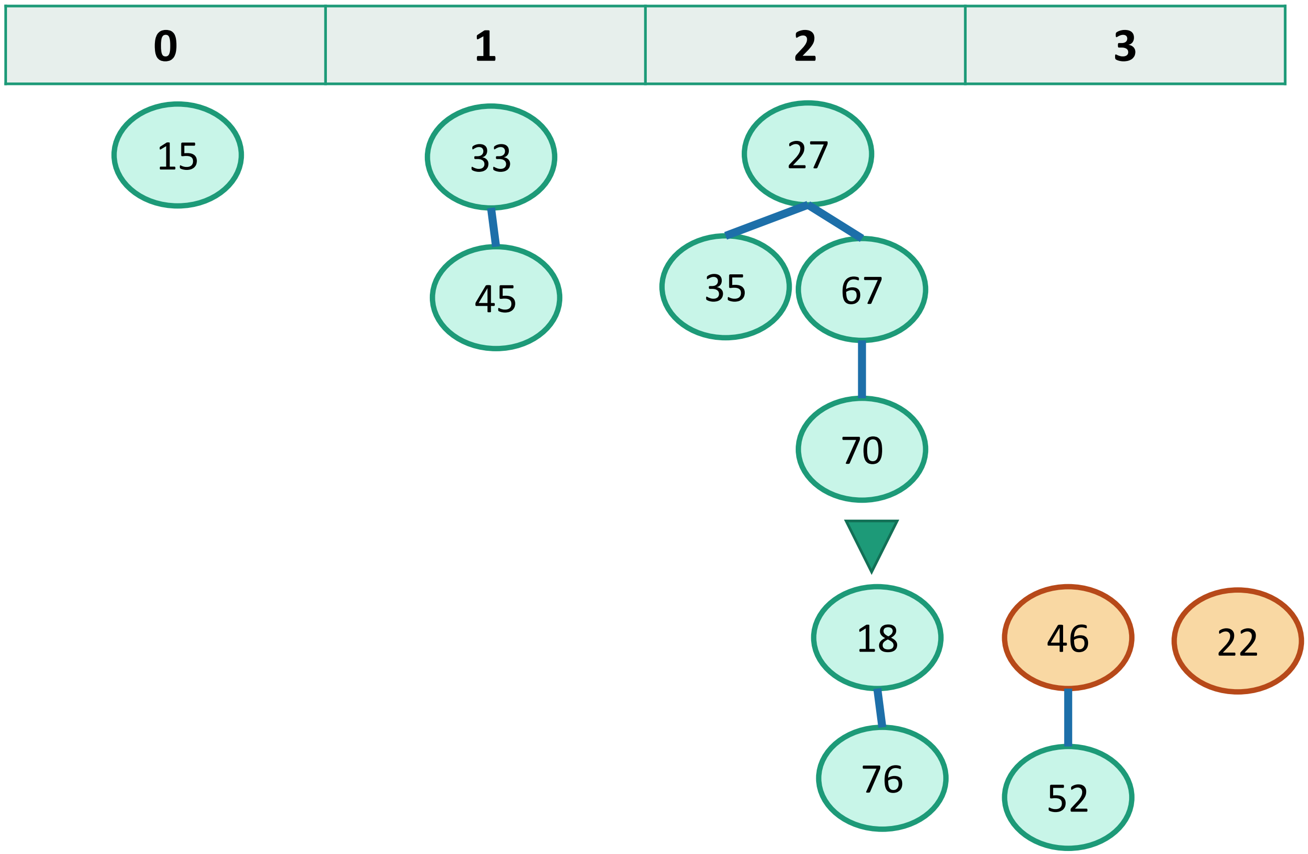
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

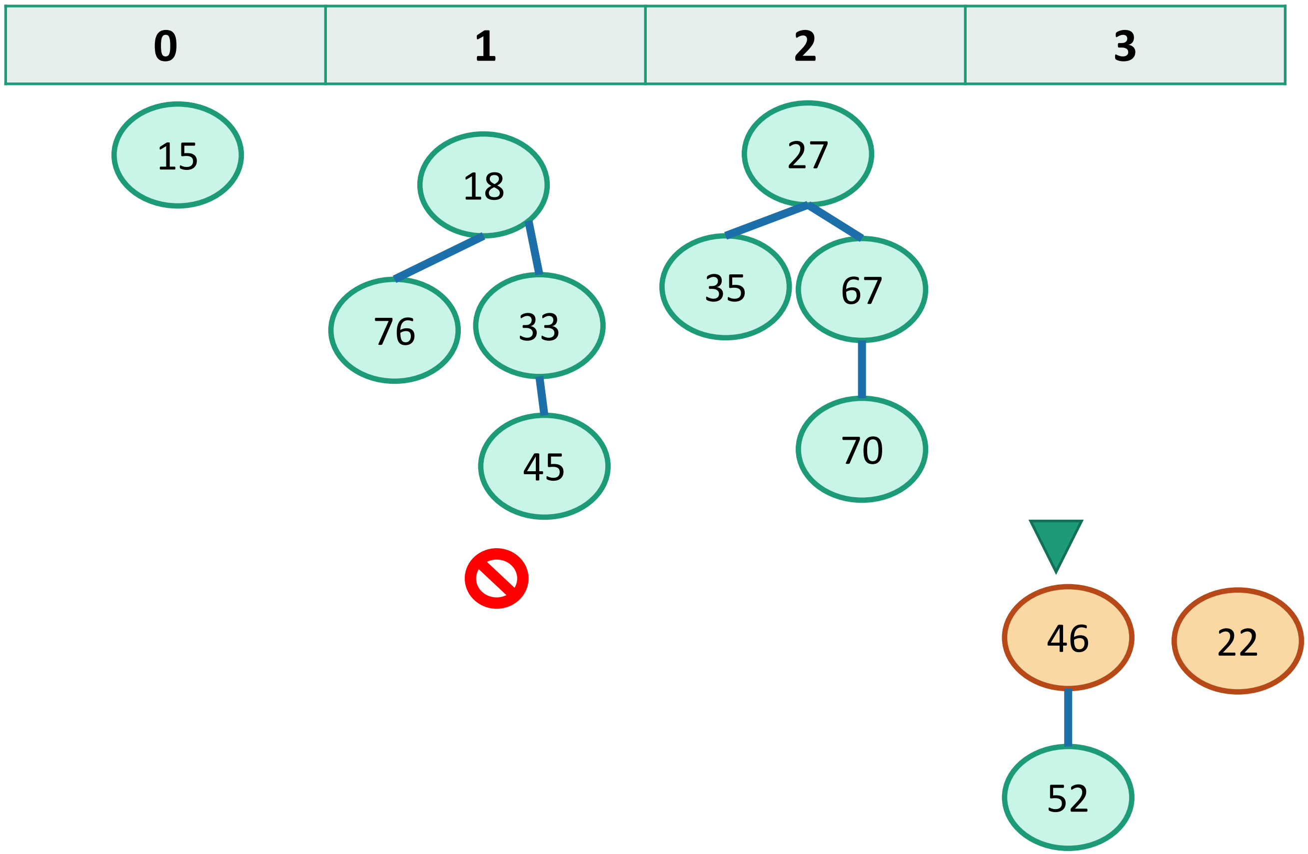
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

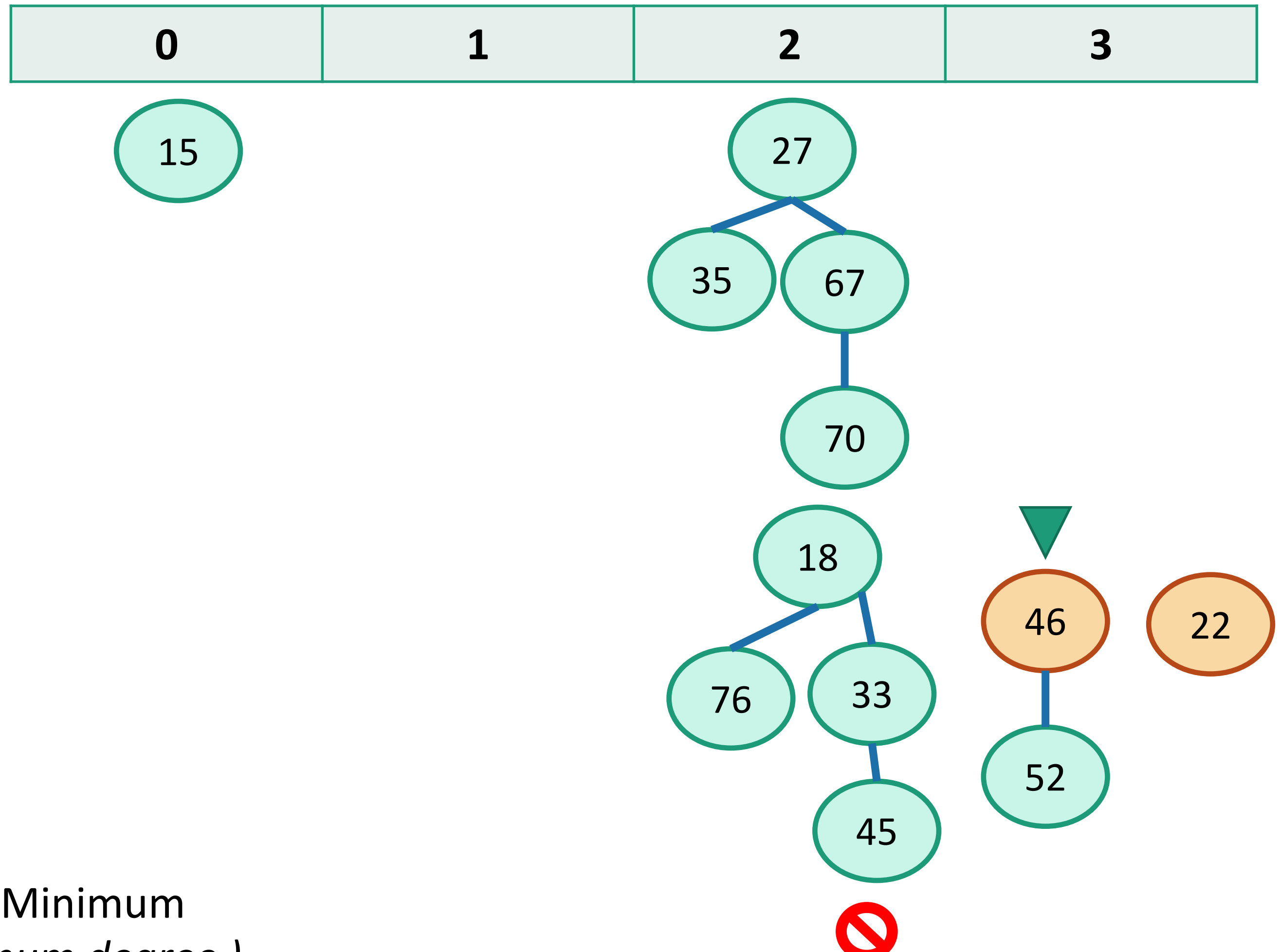
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

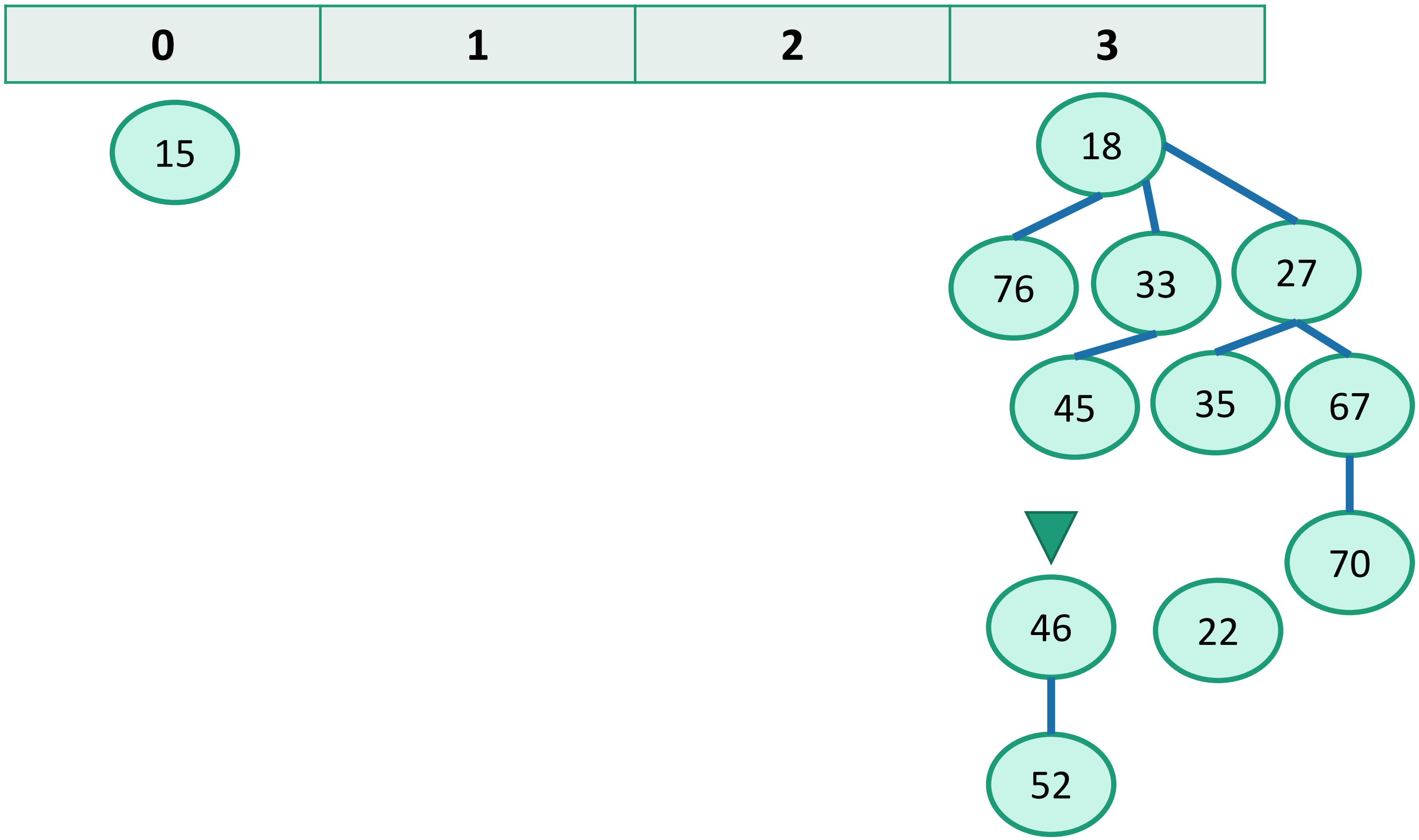
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

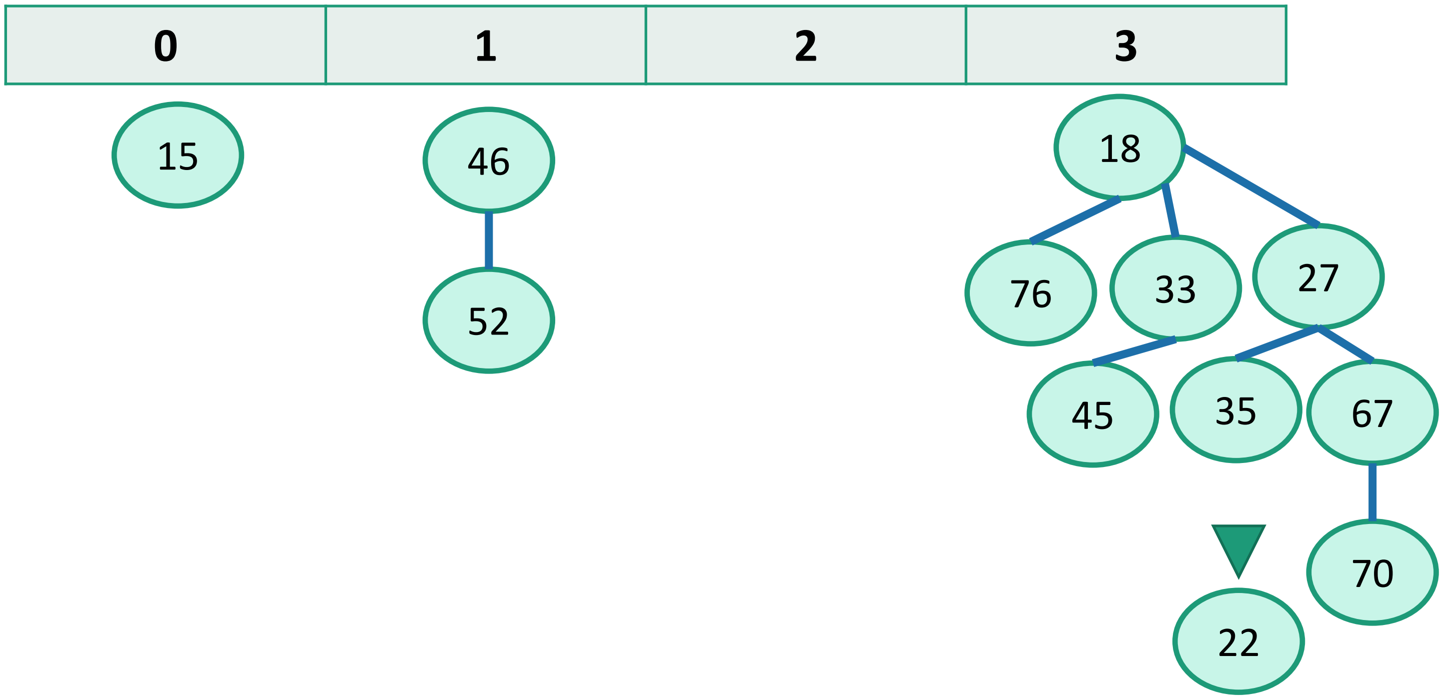
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

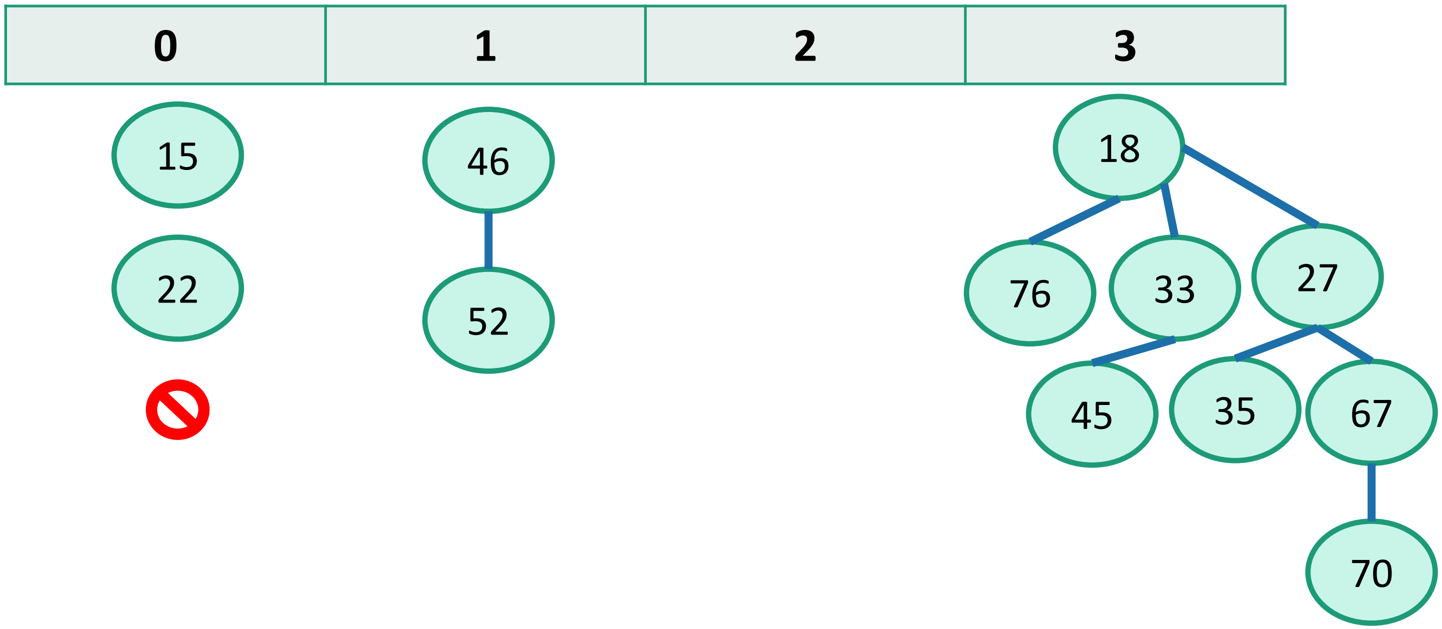
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

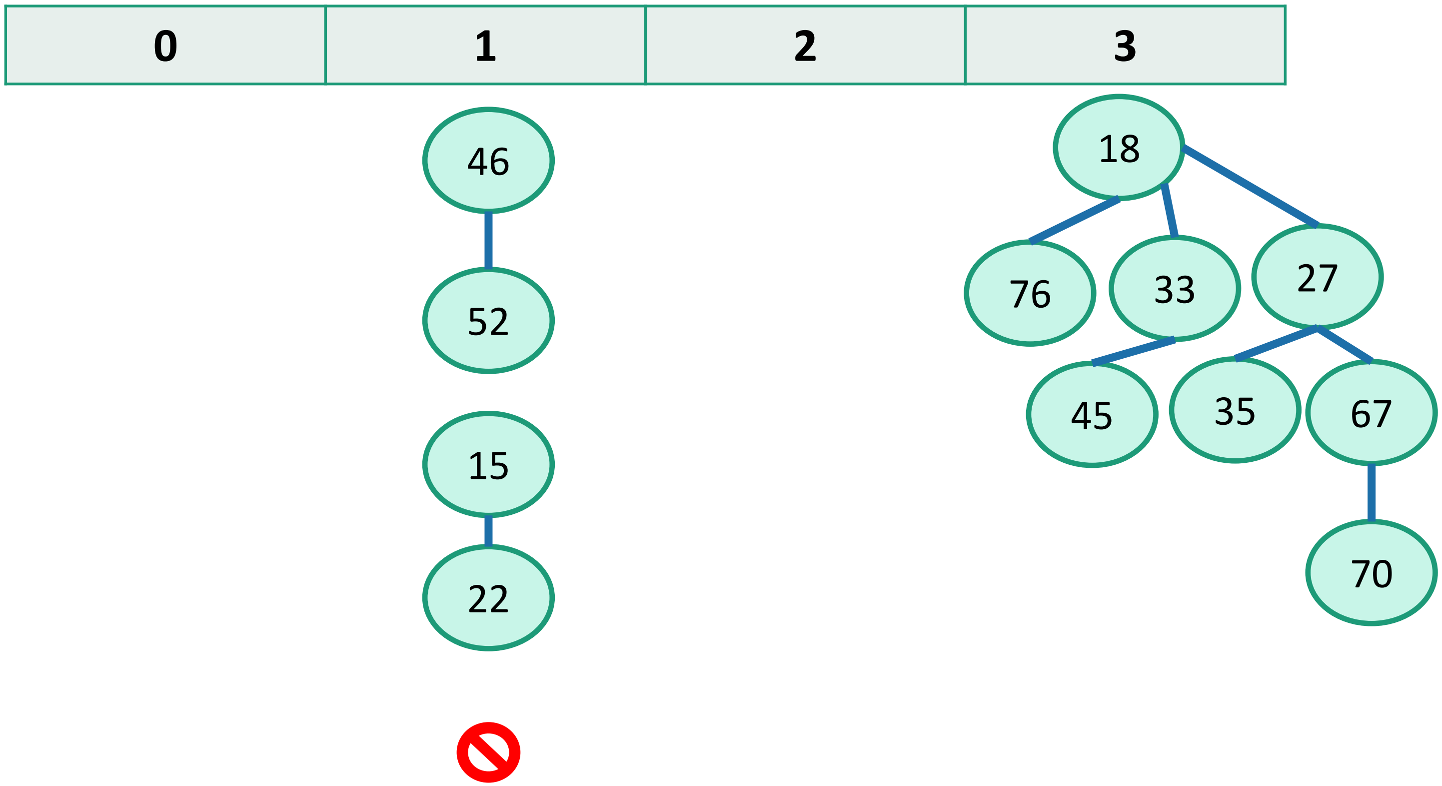
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

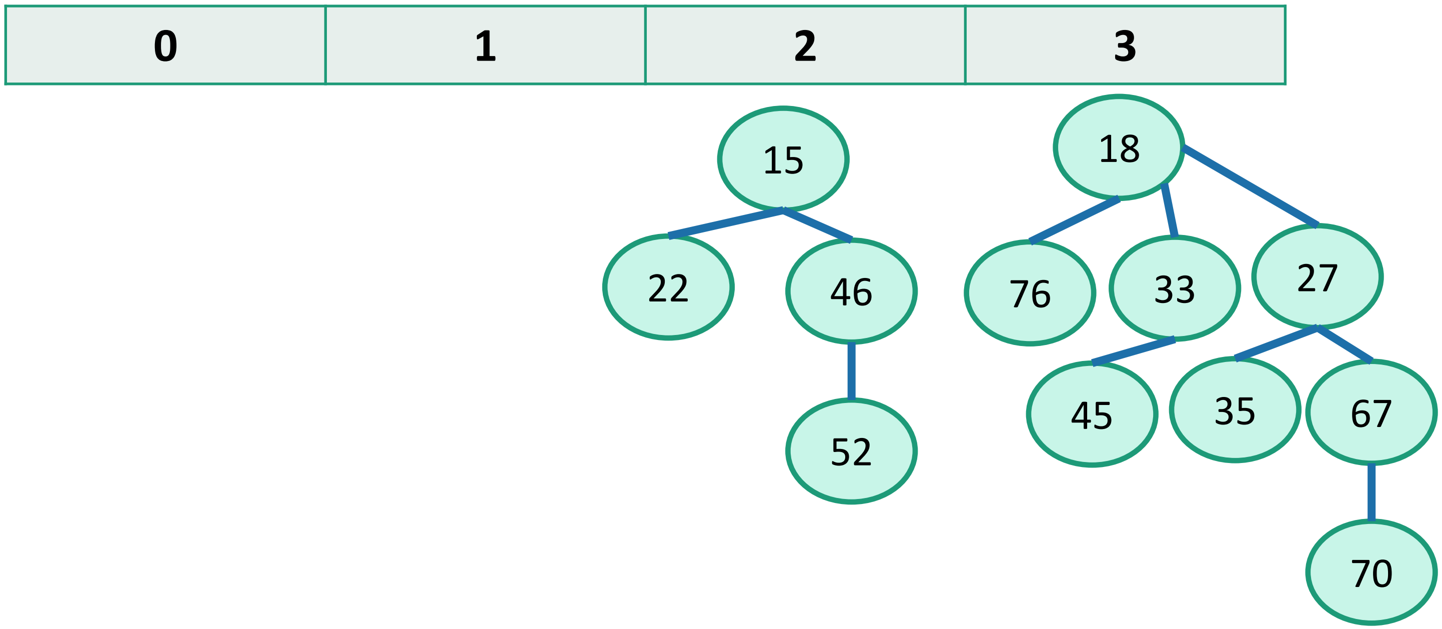
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging

Fibonacci Heap

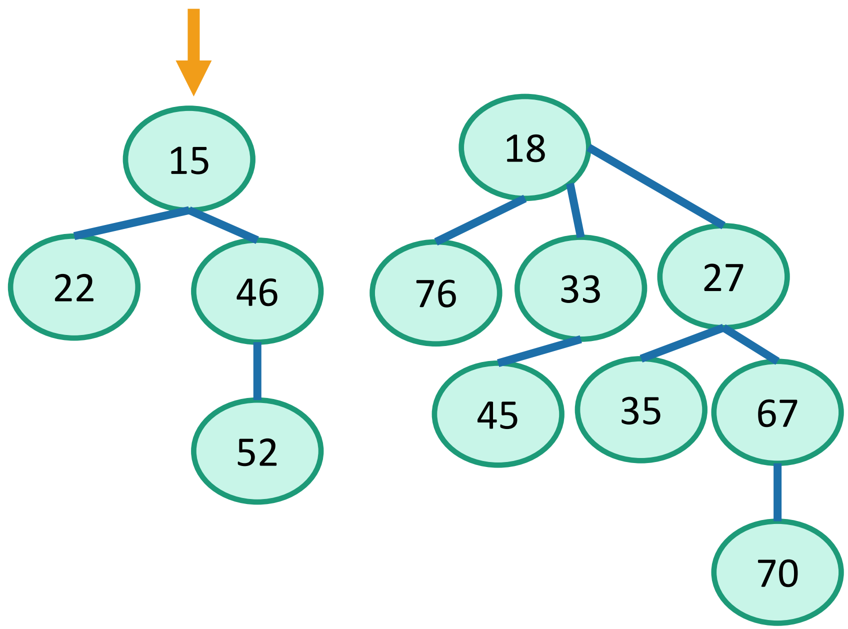
Operations : ExtractMin



- Remove Minimum
  O (maximum degree)
- Clean up : reduce #trees by merging
- *O (maximum degree + #trees)*

Fibonacci Heap

Operations : ExtractMin



- Remove Minimum

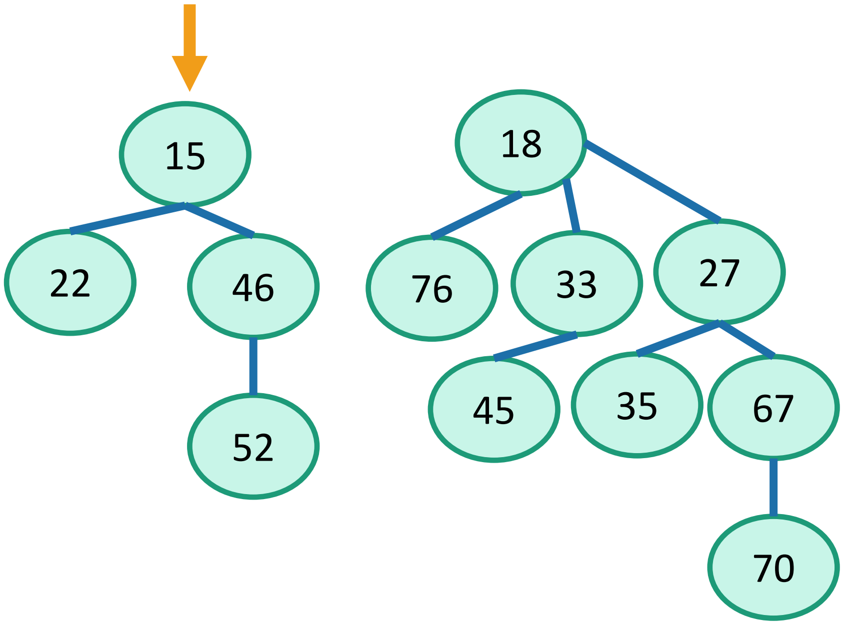
  O (maximum degree)
- Clean up : reduce #trees by merging

  O (maximum degree + #trees)
- Rebuild heap

  O (maximum degree)

Fibonacci Heap

Operations : ExtractMin



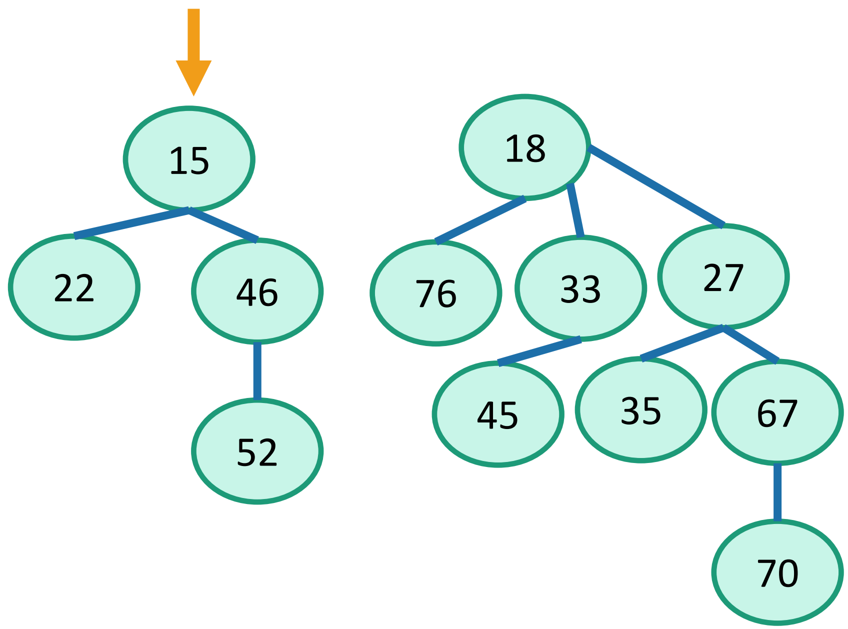
Actual cost : ***$c_i = O(rank(H)) + O(trees(H))$***

Its rank = number of children.

H = heap

Fibonacci Heap

Operations : ExtractMin



Actual cost : *O (maximum degree + #trees)*

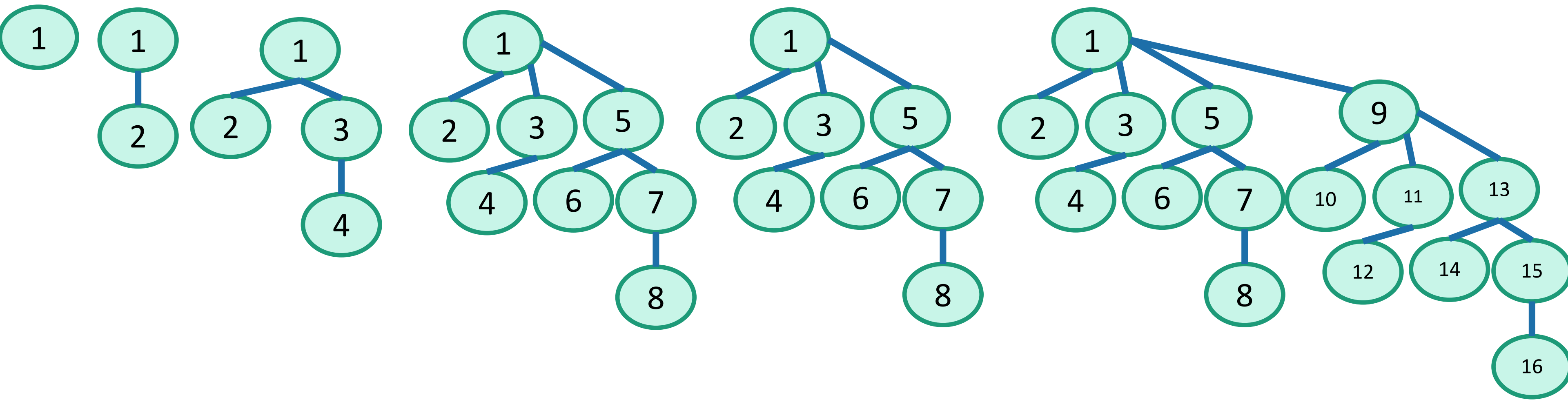
We had amortized analysis : *O (# roots after + max degree)*

O (max degree)

Fibonacci Heap

Bionomial tree

how large some tree with a given degree can be.



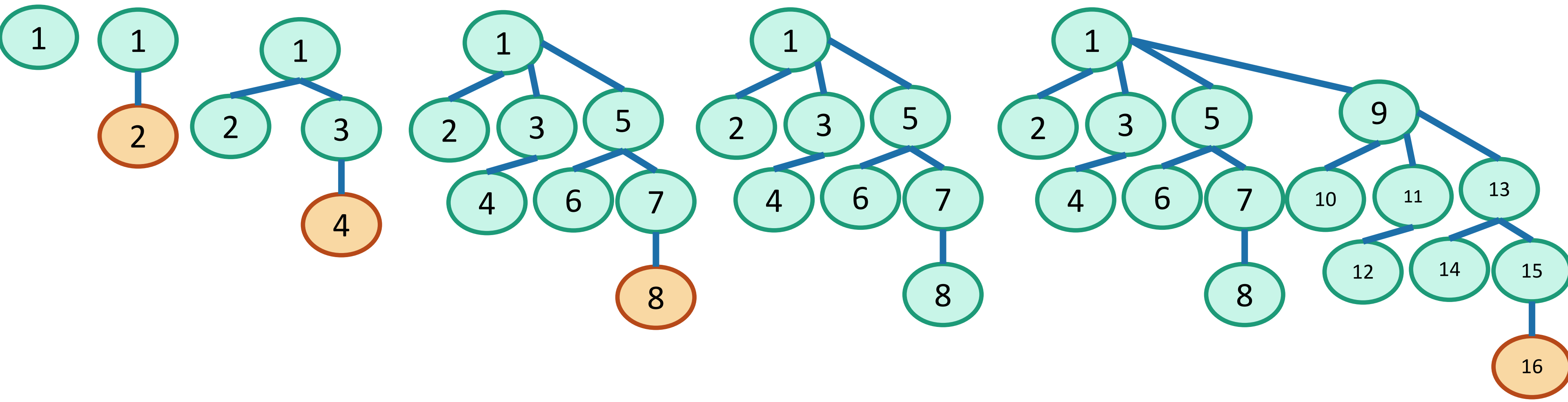
ExtractMin :

- Replace node with it's children
- Merge nodes with equal degree

Fibonacci Heap

Bionomial tree

how large some tree with a given degree can be.



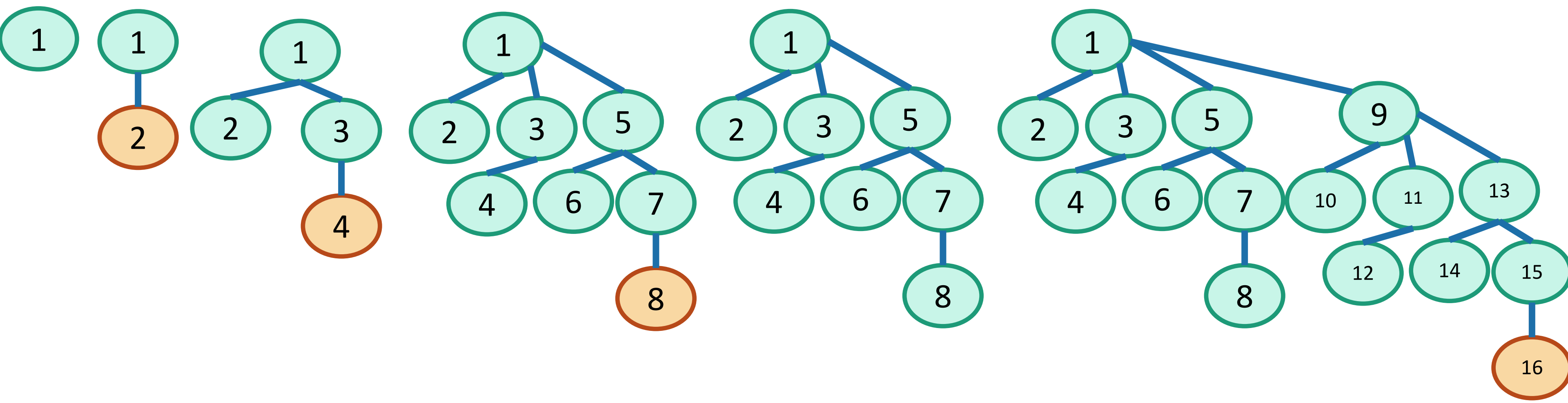
Degree d → 2^d nodes

K nodes → degree $\log_2(k)$

Fibonacci Heap

Bionomial tree

how large some tree with a given degree can be.



Degree d → 2^d nodes

Max degree $\log_2(n)$

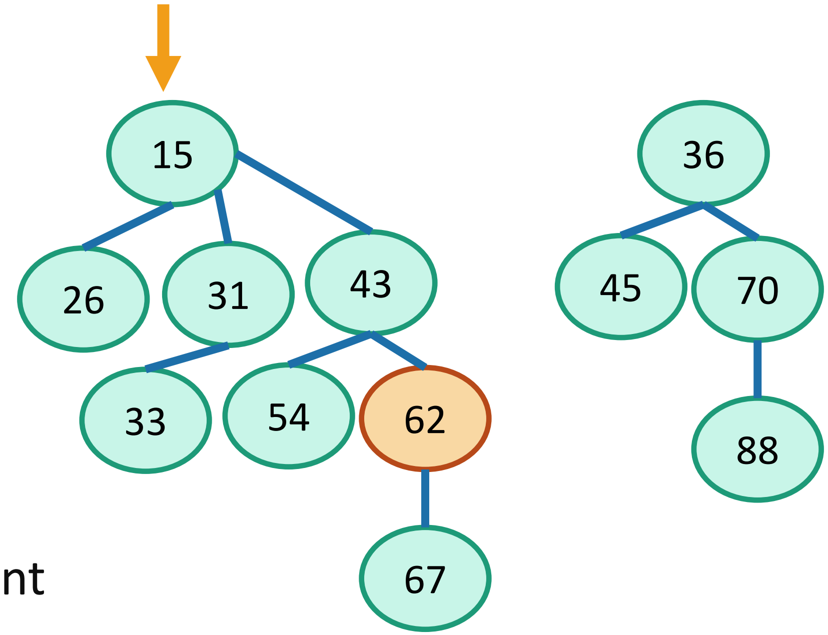
Fibonacci Heap

Time complexity comparison

Binary Heap			
GetMin	Insert	ExtractMin	DecreaseKey
O(1)	*O(logn)*	*O(logn)*	*O(logn)*
Fibonacci Heap			
GetMin	Insert	ExtractMin	DecreaseKey
O(1)	*O(1)*	*O(logn)*	*?*

Fibonacci Heap

Operations: DecreaseKey

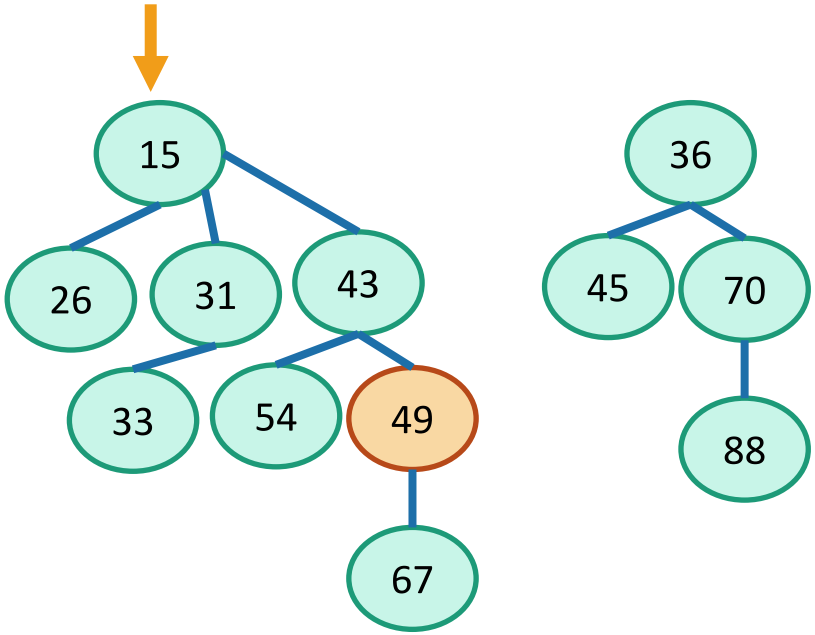


If new value >= parent

→Do nothing

Fibonacci Heap

Operations: DecreaseKey

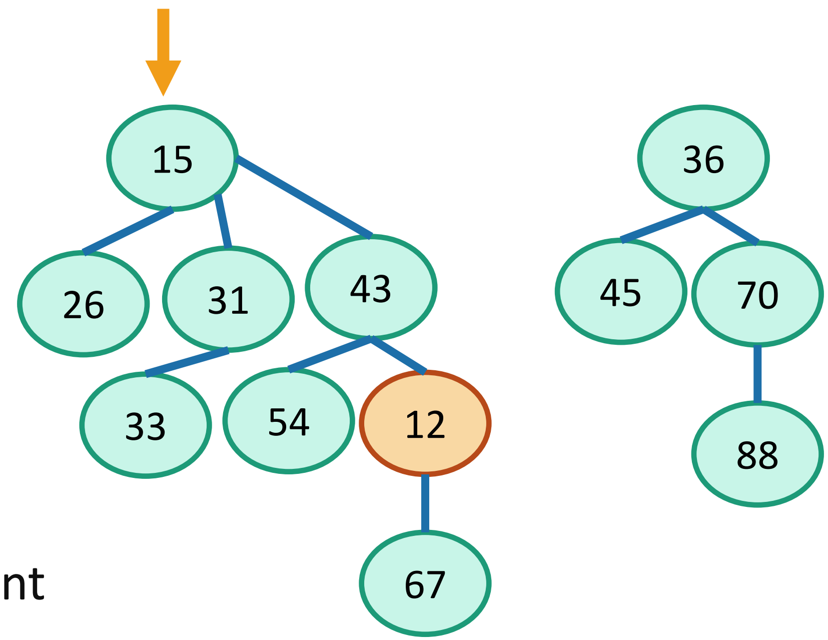


If new value >= parent :

→Do nothing

Fibonacci Heap

Operations: DecreaseKey



If new value >= parent

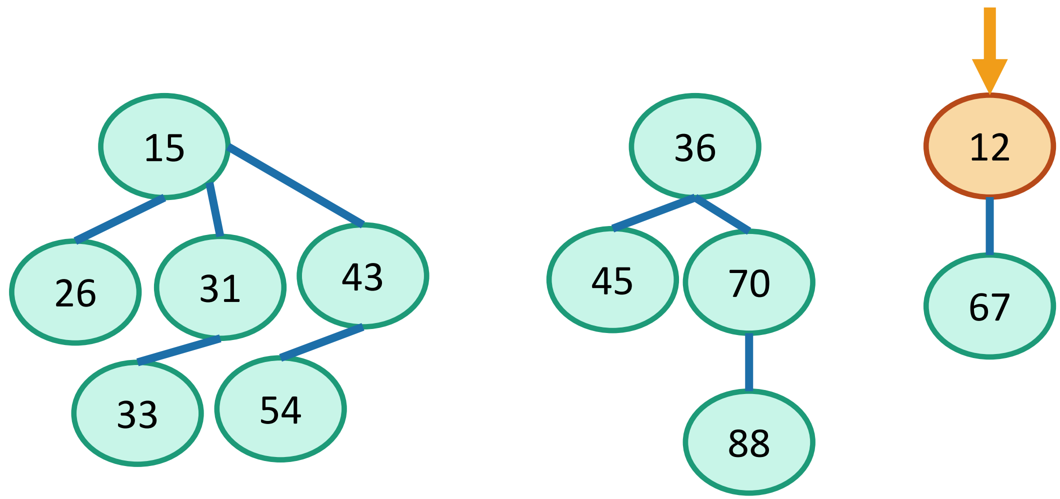
→Do nothing

If new value < parent :

→Cut out node

Fibonacci Heap

Operations: DecreaseKey



If new value >= parent

→Do nothing

If new value < parent :

→Cut out node

O(1)

#roots +=1

Same as insert!

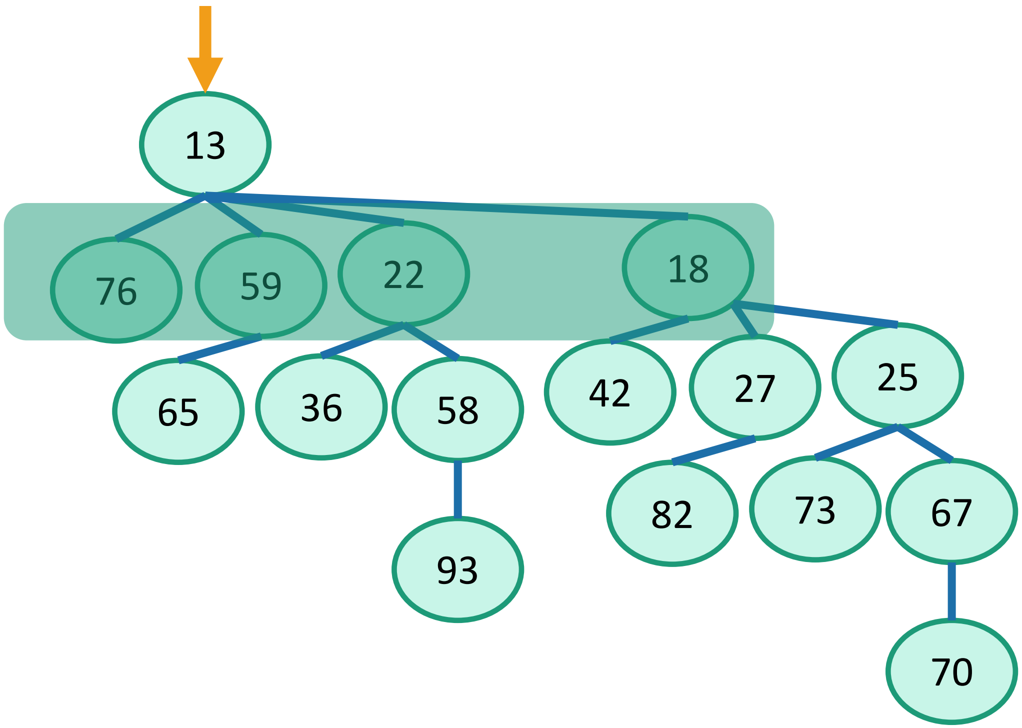
→

ExtractMin

O(logn)
amortized

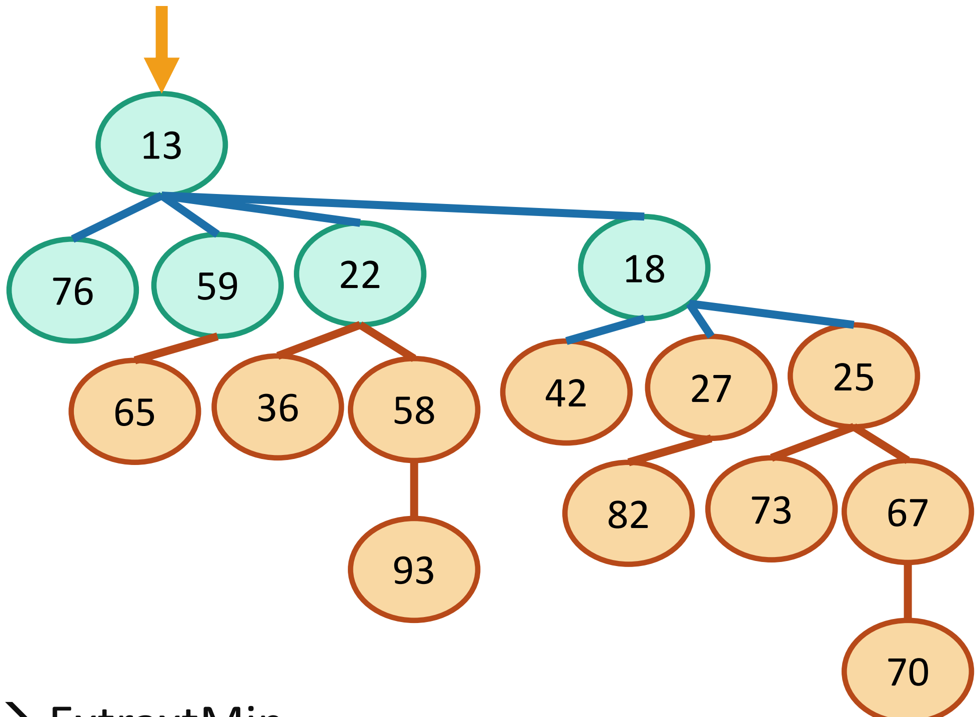
Fibonacci Heap

Operations: DecreaseKey



Fibonacci Heap

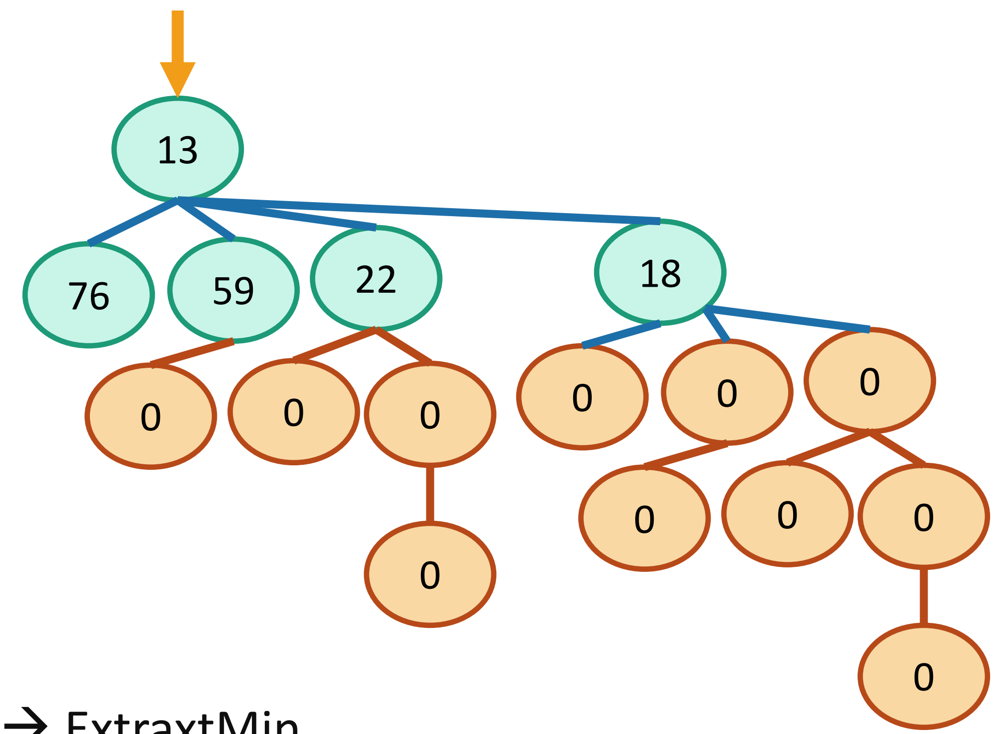
Operations: DecreaseKey



DecreaseKet to 0 → ExtraxtMin

Fibonacci Heap

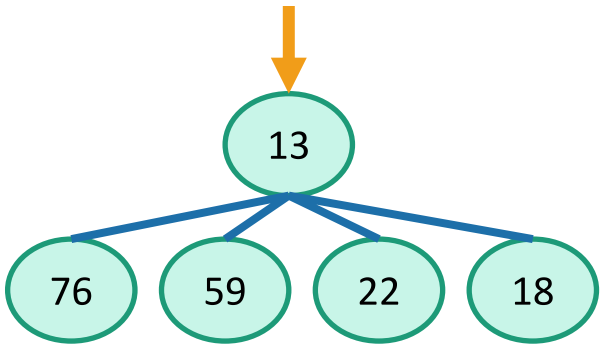
Operations: DecreaseKey



DecreaseKet to 0 → ExtraxtMin

Fibonacci Heap

Operations: DecreaseKey



Degree d :

2^d nodes → at least d+1 nodes

Max degree :

$\text{Log}_2(n)$ → n-1

Fibonacci Heap

- Don’t cut out nodes :

DecreaseKey slow

ExtractMin fast

- Cut out nodes :

DecreaseKey fast

ExtractMin slow

Fibonacci Heap

- Don’t cut out nodes :

DecreaseKey slow

ExtractMin fast

- Cut out nodes :

DecreaseKey fast

ExtractMin slow

Cut out at most one child per node :

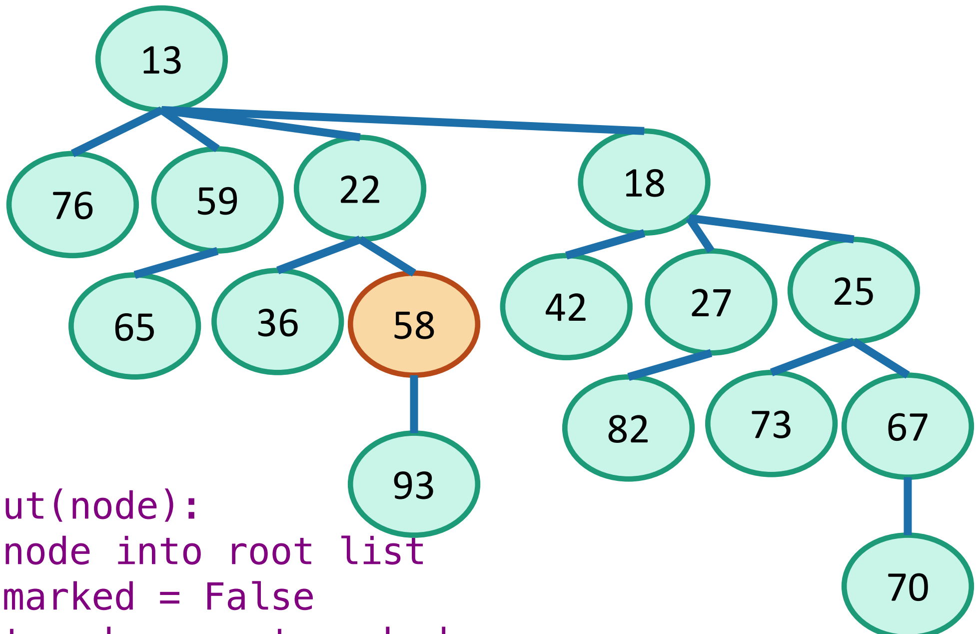
DecreaseKey fast

ExtractMin fast

Fibonacci Heap

Operations: DecreaseKey

Cut out at most one child per node

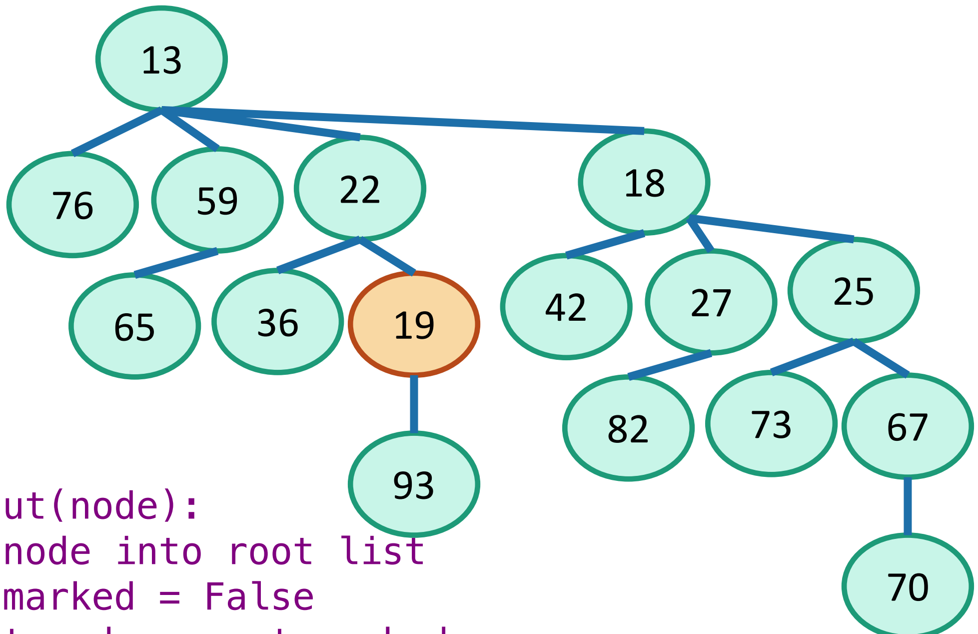



```
def cut_out(node):
    move node into root list
    node.marked = False
    if not node.parent.marked:
        node.parent.marked = True
    else :
        cut_out(node.parent)
```

Fibonacci Heap

Operations: DecreaseKey

Cut out at most one child per node

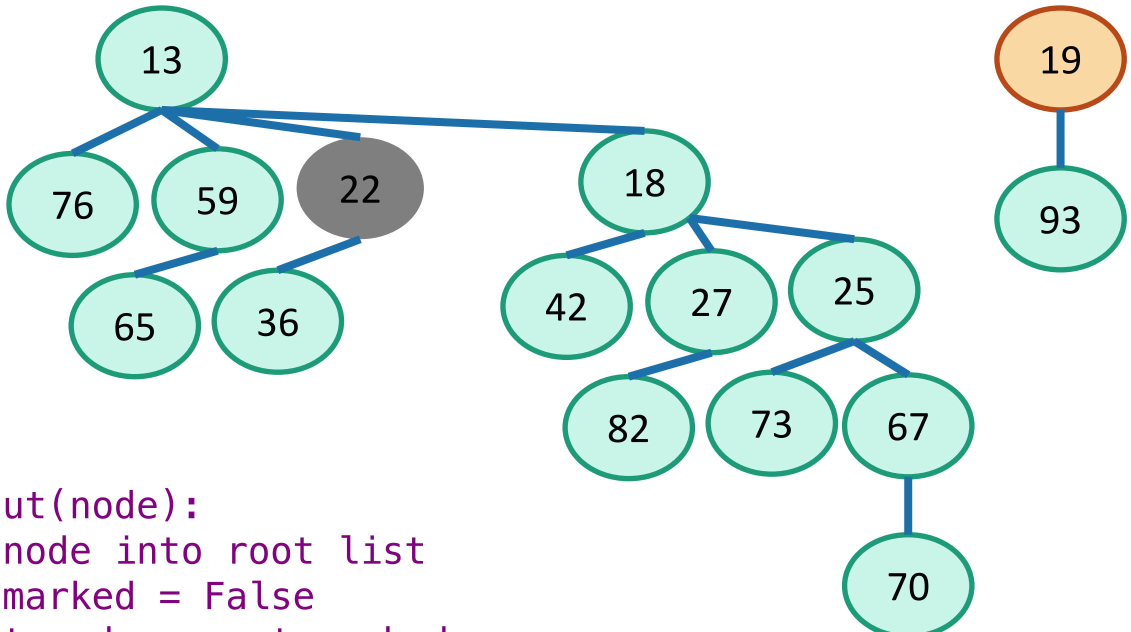



```
def cut_out(node):
    move node into root list
    node.marked = False
    if not node.parent.marked:
        node.parent.marked = True
    else :
        cut_out(node.parent)
```

Fibonacci Heap

Operations: DecreaseKey

Cut out at most one child per node

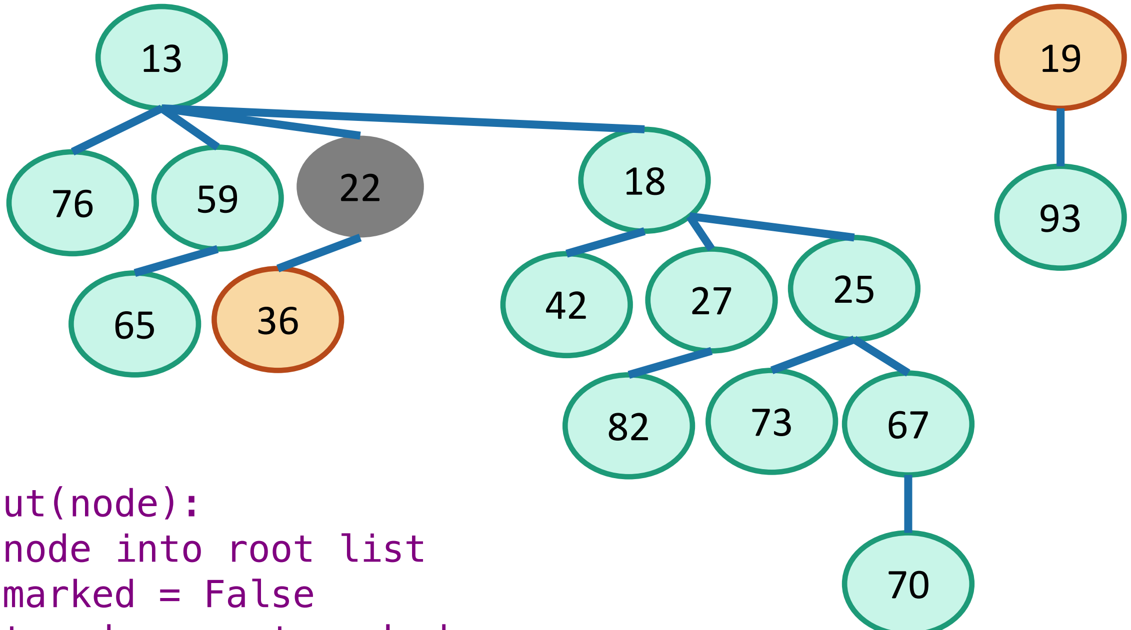



```
def cut_out(node):
    move node into root list
    node.marked = False
    if not node.parent.marked:
        node.parent.marked = True
    else :
        cut_out(node.parent)
```

Fibonacci Heap

Operations: DecreaseKey

Cut out at most one child per node

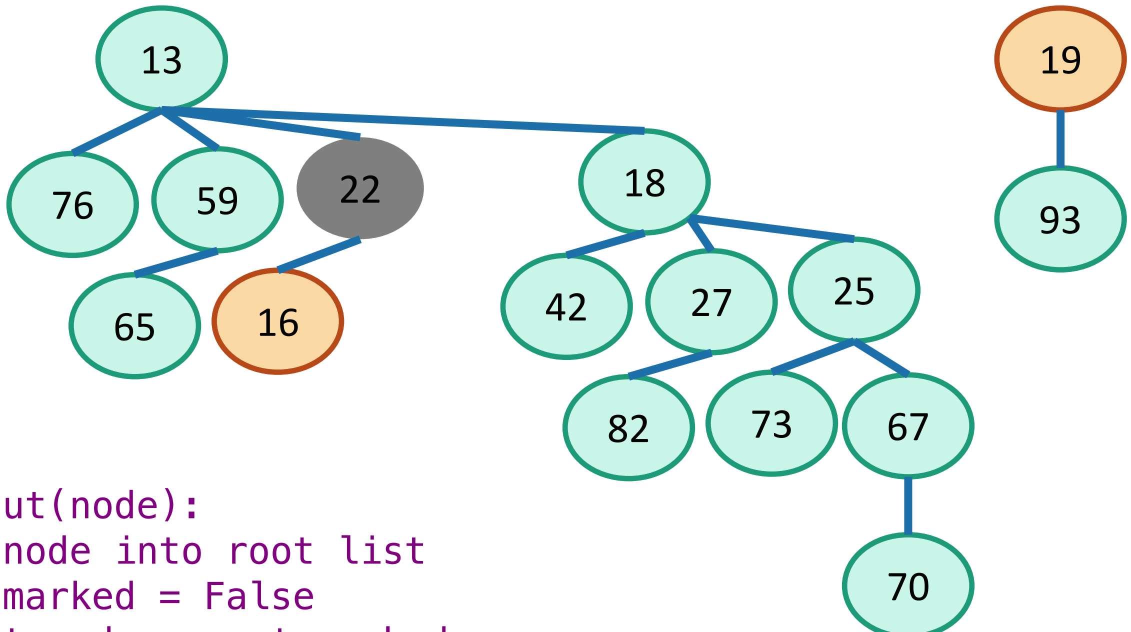



```
def cut_out(node):
    move node into root list
    node.marked = False
    if not node.parent.marked:
        node.parent.marked = True
    else :
        cut_out(node.parent)
```

Fibonacci Heap

Operations: DecreaseKey

Cut out at most one child per node

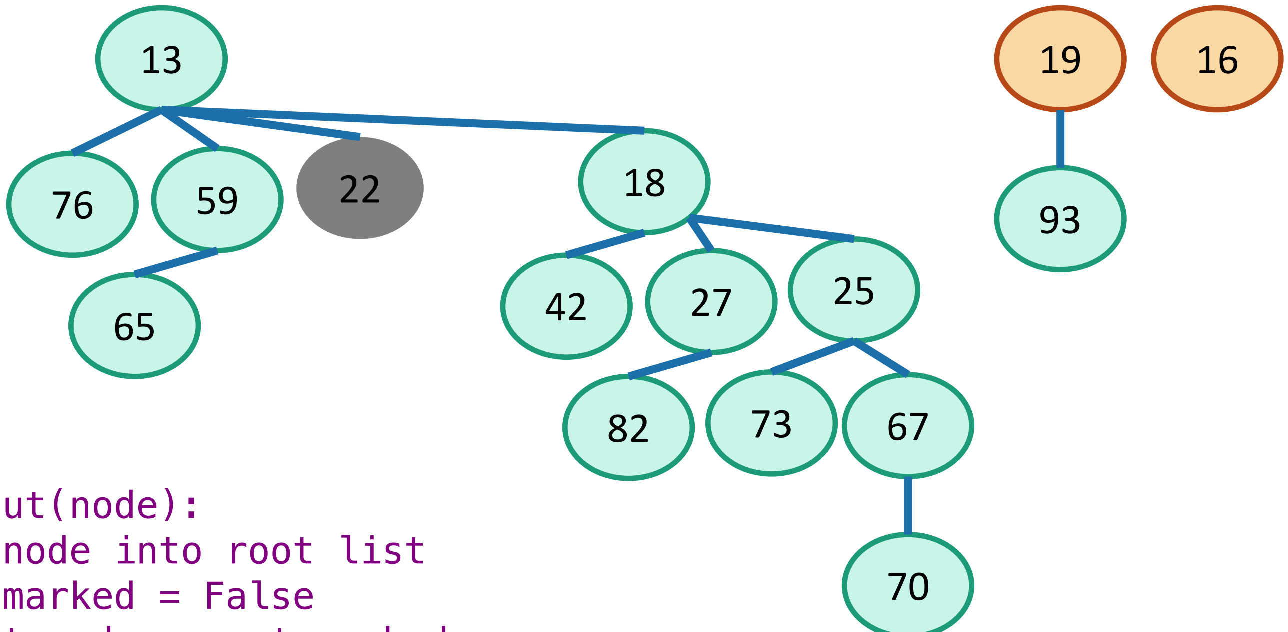



```
def cut_out(node):
    move node into root list
    node.marked = False
    if not node.parent.marked:
        node.parent.marked = True
    else :
        cut_out(node.parent)
```

Fibonacci Heap

Operations: DecreaseKey

Cut out at most one child per node

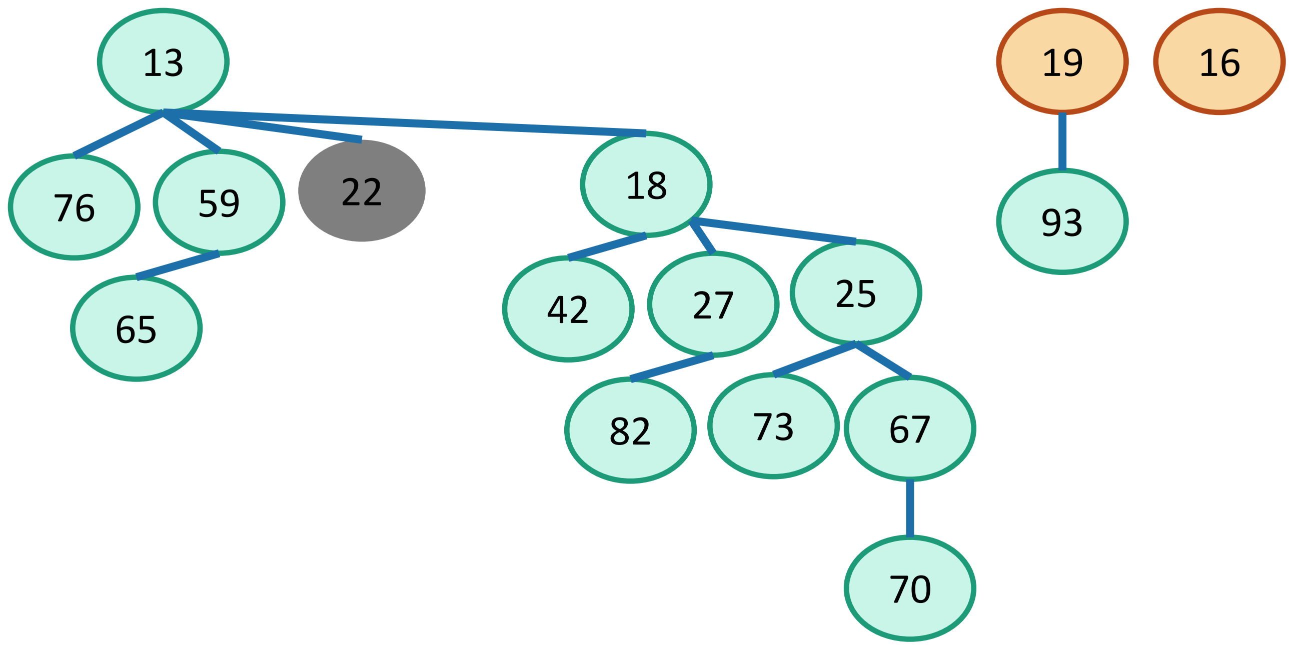



```
def cut_out(node):
    move node into root list
    node.marked = False
    if not node.parent.marked:
        node.parent.marked = True
    else :
        cut_out(node.parent)
```

Fibonacci Heap

Operations: DecreaseKey

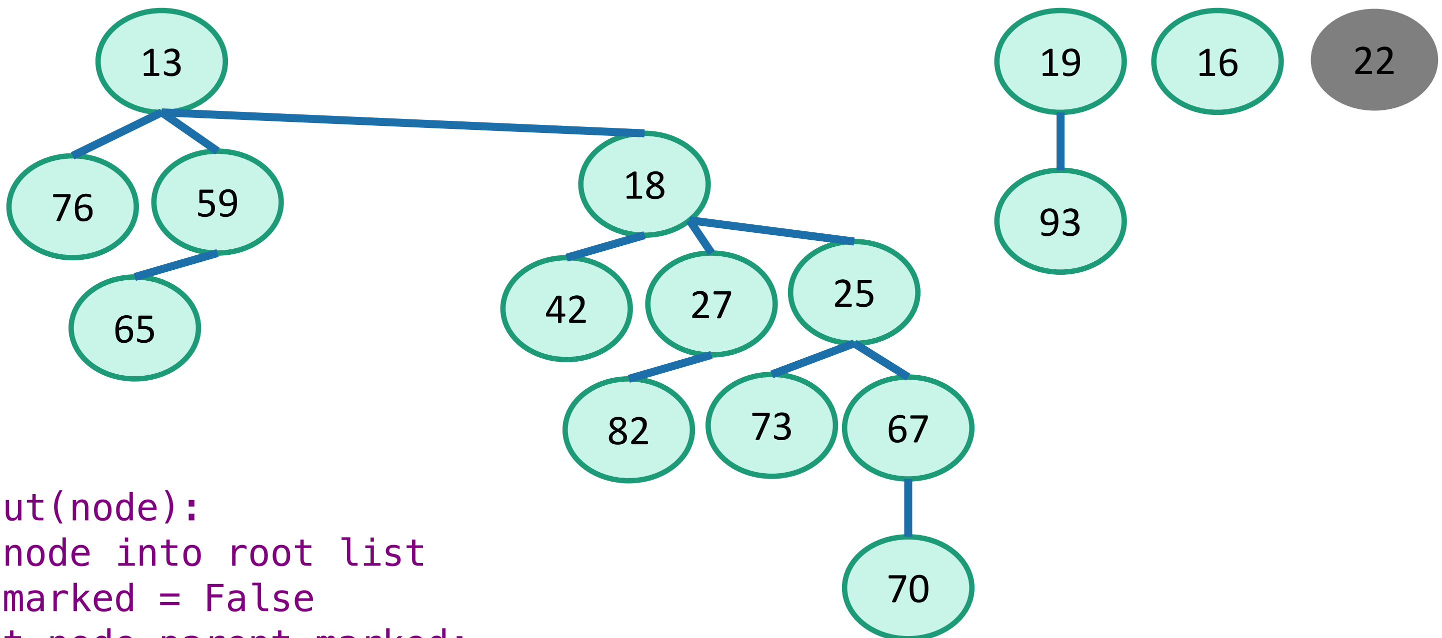
Cut out at most one child per node



Fibonacci Heap

Operations: DecreaseKey

Cut out at most one child per node




```
def cut_out(node):
    move node into root list
    node.marked = False
    if not node.parent.marked:
        node.parent.marked = True
    else :
        cut_out(node.parent)
```

Fibonacci Heap
Operations: DecreaseKey

Binary Heap			
GetMin	Insert	ExtractMin	DecreaseKey
O(1)	*O(logn)*	*O(logn)*	*O(logn)*
Fibonacci Heap			
GetMin	Insert	ExtractMin	DecreaseKey
O(1)	*O(1)*	*O(logn)*	*O(1)*

Fibonacci Heap

Operations: DecreaseKey

1. Why does DecreaseKey take O(1) time?
2. How is ExtractMin still fast?
3. Do node degrees grow logarithmically?

Fibonacci Heap

Why does DecreaseKey take O(1) time?

K DecreaseKey → cut out ≤ 2k nodes

Worst case : *O(n)*. , Amortized : *O(1)*

Fibonacci Heap

How is ExtractMin still fast?

- Insert :

  Add 1 tree

  Clean up 1 node

- DecreaseKey :

  Add ≤ 2 trees

  Clean up ≤ 2 nodes

Fibonacci Heap

Do node degrees grow logarithmically?

- degree d tree : 2^d nodes

→ *max degree O(logn)*

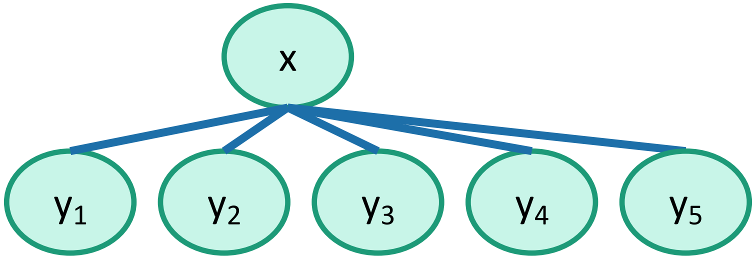
Fibonacci Heap
Do trees grow exponentially?

- Smallest degree d tree : 2^d nodes

→ *max degree O(logn)*

Fibonacci Heap

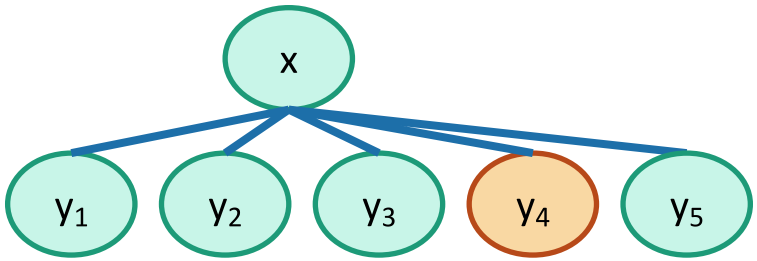
Do trees grow exponentially?



Minimum degree?

Fibonacci Heap

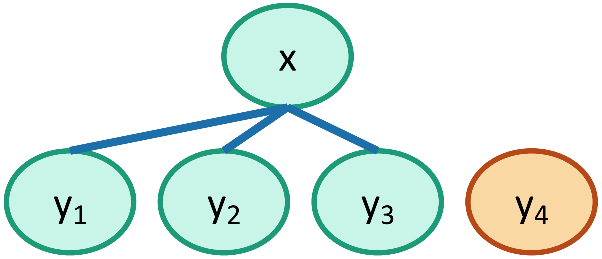
Do trees grow exponentially?



Minimum degree?

Fibonacci Heap

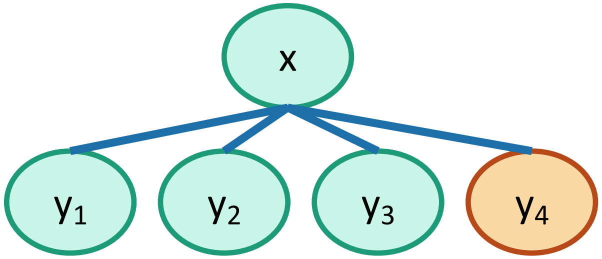
Do trees grow exponentially?



Minimum degree?

Fibonacci Heap

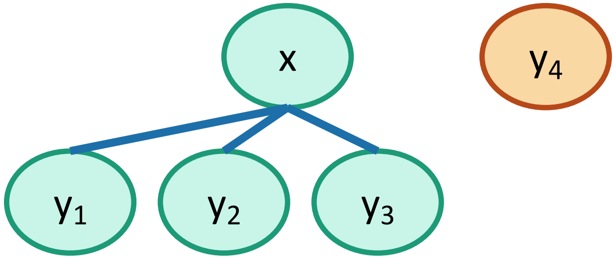
Do trees grow exponentially?



degree ≥ 3

Fibonacci Heap

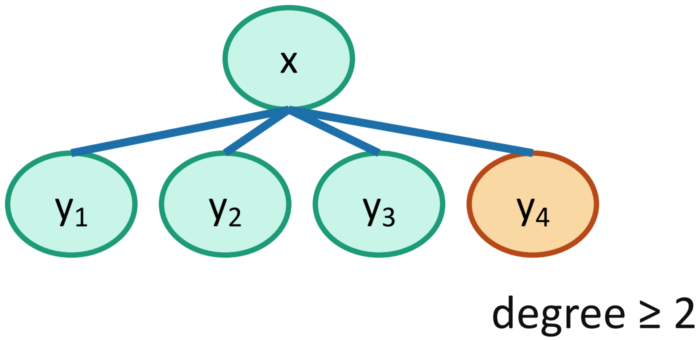
Do trees grow exponentially?



degree ≥ 2

Fibonacci Heap

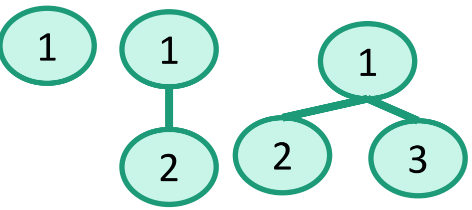
Do trees grow exponentially?



degree of *i th* child ≥ *i-2*

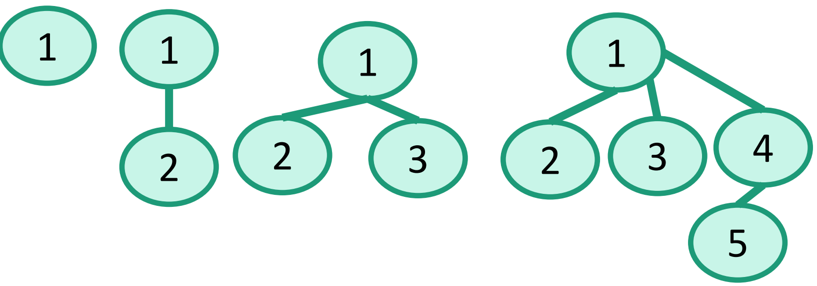
Fibonacci Heap

Do trees grow exponentially?



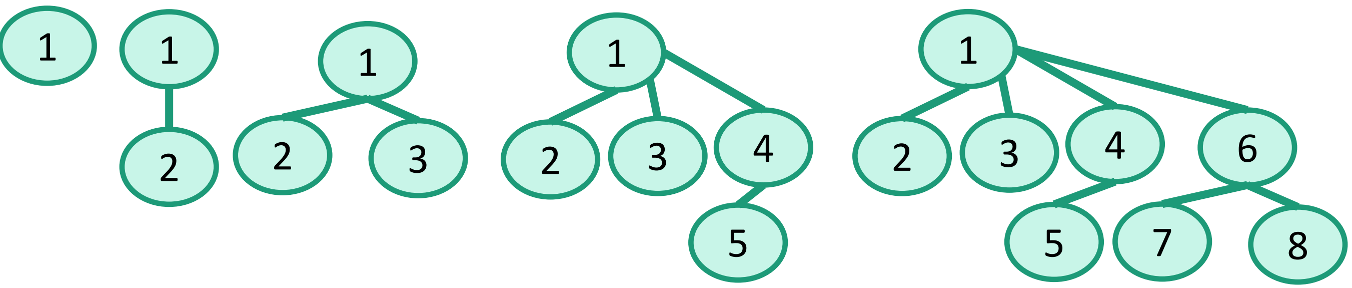
Fibonacci Heap

Do trees grow exponentially?



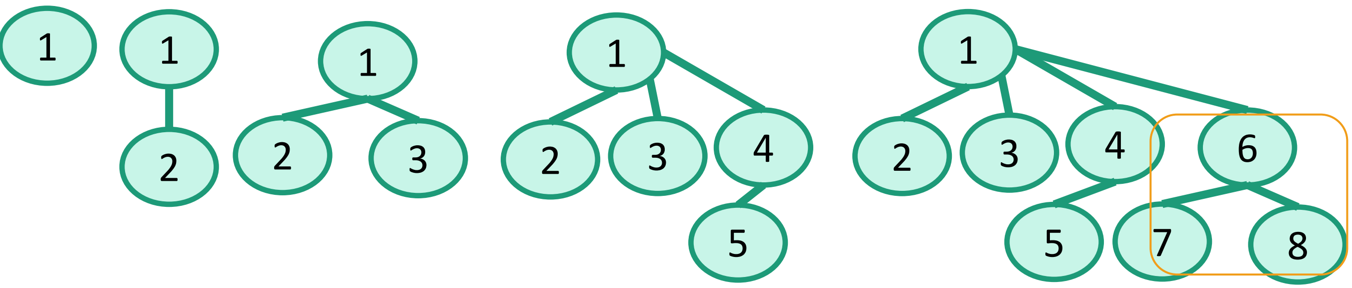
Fibonacci Heap

Do trees grow exponentially?



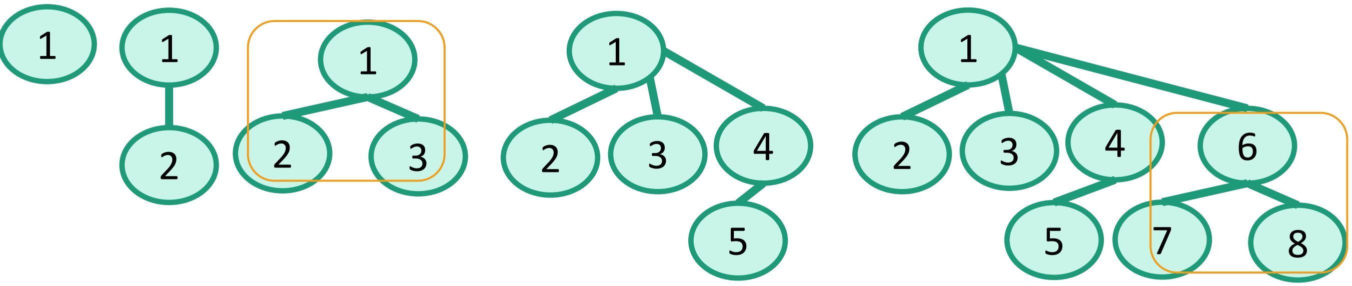
Fibonacci Heap

Do trees grow exponentially?



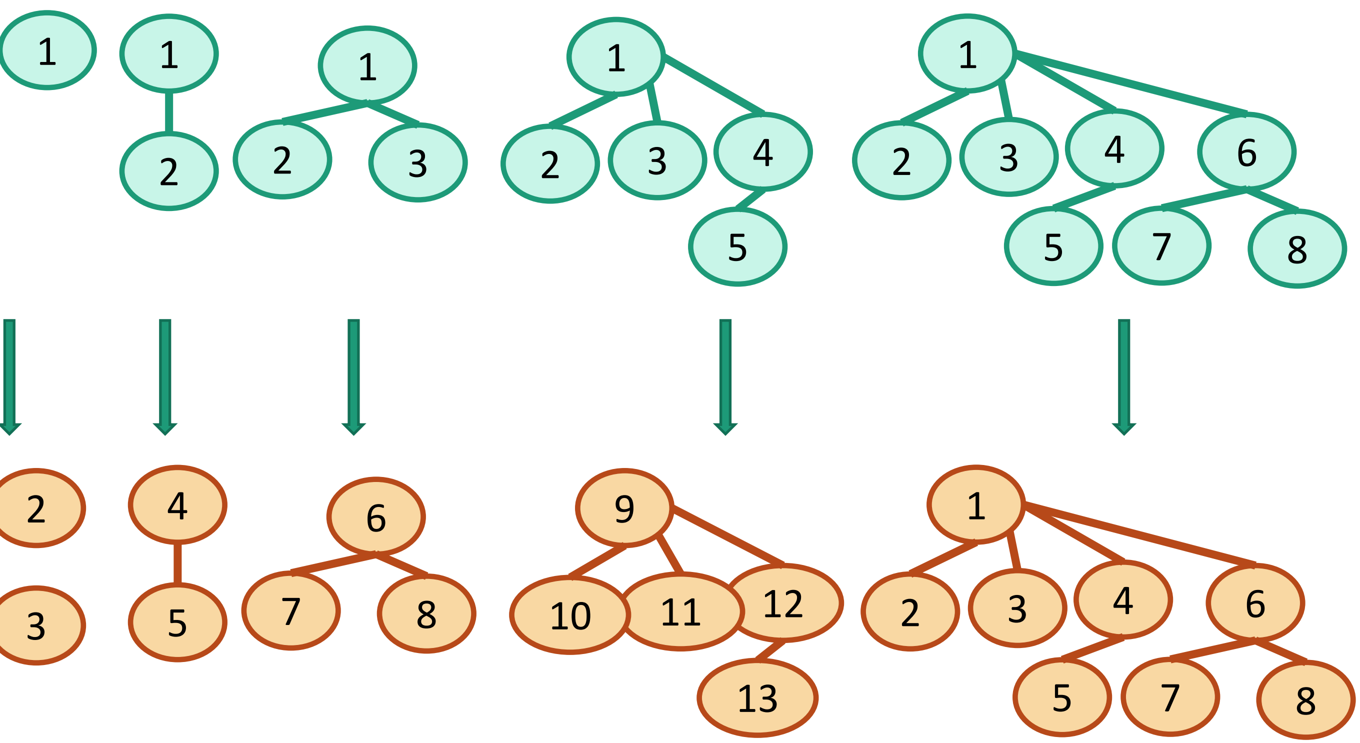
Fibonacci Heap

Do trees grow exponentially?



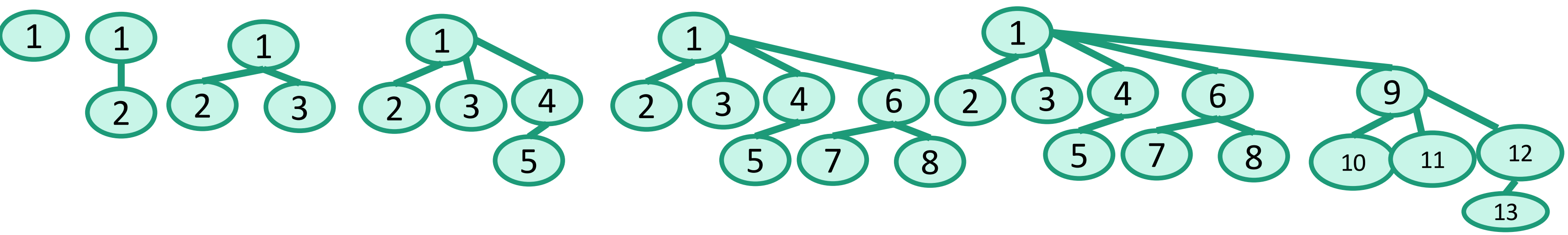
Fibonacci Heap

Do trees grow exponentially?



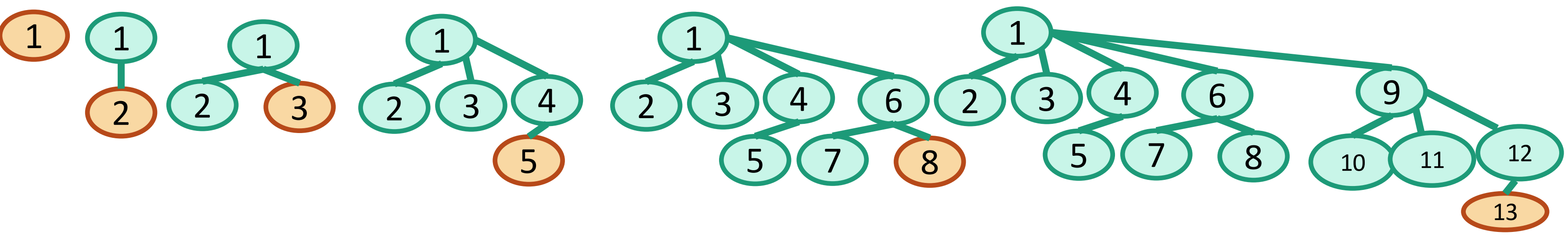
Fibonacci Heap

Do trees grow exponentially?



Fibonacci Heap

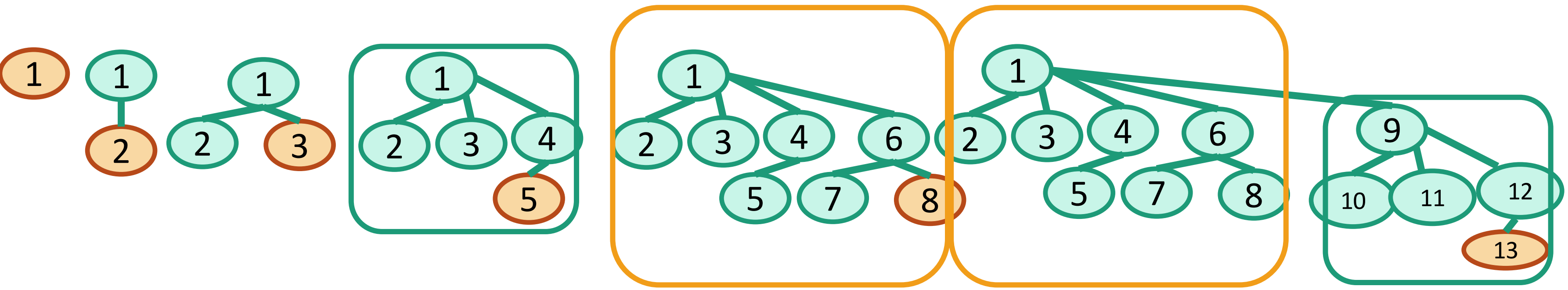
Do trees grow exponentially?



Fibonacci numbers : 0, 1, 1, 2, 3, 5, 8, 13

Fibonacci Heap

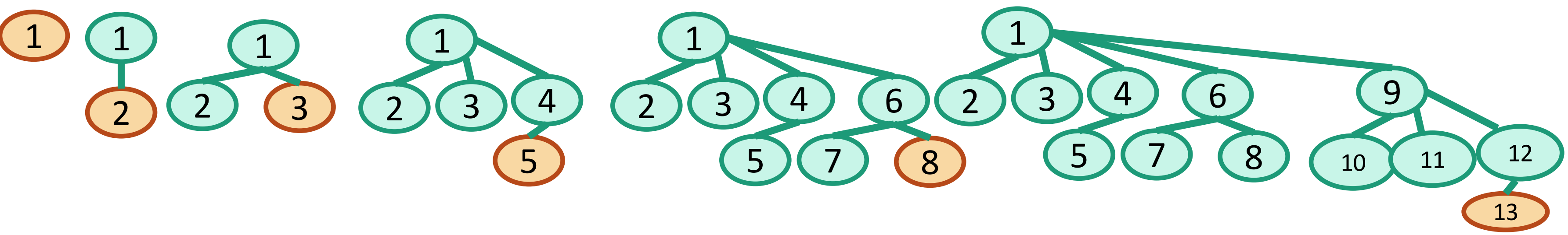
Do trees grow exponentially?



Fibonacci numbers : 0, 1, 1, 2, 3, 5, 8, 13

Fibonacci Heap

Do trees grow exponentially?



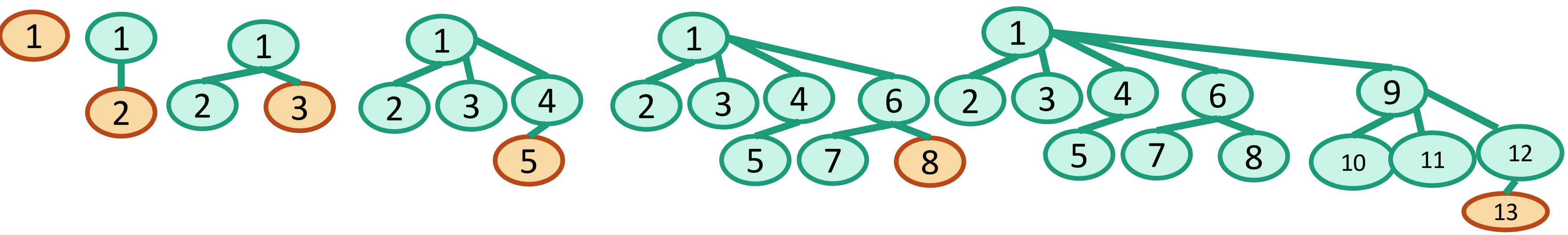
Fibonacci numbers : 0, 1, 1, 2, 3, 5, 8, 13

Ration of two adjacent values in the sequence : golden ratio

$$\frac{233}{144} \approx 1.68 \approx \varphi = \frac{1+\sqrt{5}}{2}$$

Fibonacci Heap

Do trees grow exponentially? Yes !



"Size of tree with degree d" $\geq \varphi^d$

Fibonacci Heap vs Binary Heap

Binary Heap			
GetMin	Insert	ExtractMin	DecreaseKey
O(1)	*O(logn)*	*O(logn)*	*O(logn)*
Fibonacci Heap			
GetMin	Insert	ExtractMin	DecreaseKey
O(1)	*O(1)*	*O(logn)*	*O(1)*