Heaps

The *Priority Queue ADT*

The *Priority Queue ADT*

Data Model:

Operations:

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.
- ***p.insert(pri, val)***

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.
- ***p.insert(pri, val)*** inserts an item with priority ***pri*** and value ***val*** to ***p***.

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.
- ***p.insert(pri, val)*** inserts an item with priority ***pri*** and value ***val*** to ***p***.
- ***p.min()***

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.
- ***p.insert(pri, val)*** inserts an item with priority ***pri*** and value ***val*** to ***p***.
- ***p.min()*** returns the pair ***(pri, val)*** with the lowest priority in ***p***, or raises an ***Exception***, if ***p*** is empty

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.
- ***p.insert(pri, val)*** inserts an item with priority ***pri*** and value ***val*** to ***p***.
- ***p.min()*** returns the pair ***(pri, val)*** with the lowest priority in ***p***, or raises an ***Exception***, if ***p*** is empty
- ***p.delete_min()***

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that come out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.
- ***p.insert(pri, val)*** inserts an item with priority ***pri*** and value ***val*** to ***p***.
- ***p.min()*** returns the pair ***(pri, val)*** with the lowest priority in ***p***, or raises an ***Exception***, if ***p*** is empty
- ***p.delete_min()*** removes and returns the pair ***(pri, val)*** with the lowest priority in ***p***, or raises an ***Exception***, if ***p*** is empty.

The *Priority Queue ADT*

Data Model: A collection of priority-value pairs, that e out in an increasing priorities order

Operations:

- ***p = PriorityQueue()*** Creates an empty priority queue.
- ***p.insert(pri, val)*** inserts an item with priority ***pri*** and value ***val*** to ***p***.
- ***p.min()*** returns the pair ***(pri, val)*** with the lowest priority in ***p***, or raises an ***Exception***, if ***p*** is empty
- ***p.delete_min()*** removes and returns the pair ***(pri, val)*** with the lowest priority in ***p***, or raises an ***Exception***, if ***p*** is empty.
- ***len(p)*** returns the number of items in ***p***
- ***p.is_empty()*** returns ***True*** if ***p*** is empty, or ***False*** otherwise

The *Priority Queue ADT*

<u>Minimum Priority Queue</u>:

<u>Maximum Priority Queue</u>:

The *Priority Queue ADT*

<u>Minimum Priority Queue</u>:

p = MinPriorityQueue()
len(p)
p.is_empty()
p.insert(pri, val)
p.min()
p.delete_min()

<u>Maximum Priority Queue</u>:

The *Priority Queue ADT*

Minimum Priority Queue:

p = MinPriorityQueue()
len(p)
p.is_empty()
p.insert(pri, val)
p.min()
p.delete_min()

Maximum Priority Queue:

p = MaxPriorityQueue()
len(p)
p.is_empty()
p.insert(pri, val)
p.max()
p.delete_max()

Data Structures for *Priority Queue ADT*

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List			
Sorted Linked List			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List			
Sorted Linked List			
Balanced Search Tree			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$		
Sorted Linked List			
Balanced Search Tree			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	
Sorted Linked List			
Balanced Search Tree			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List			
Balanced Search Tree			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$		
Balanced Search Tree			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	
Balanced Search Tree			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$		

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$	$\theta(\log(n))$	

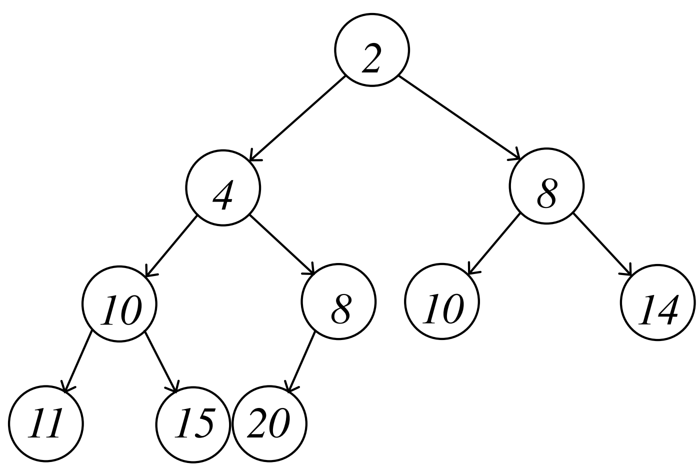
Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$	$\theta(\log(n))$	$\theta(\log(n))$

Data Structures for *Priority Queue ADT*

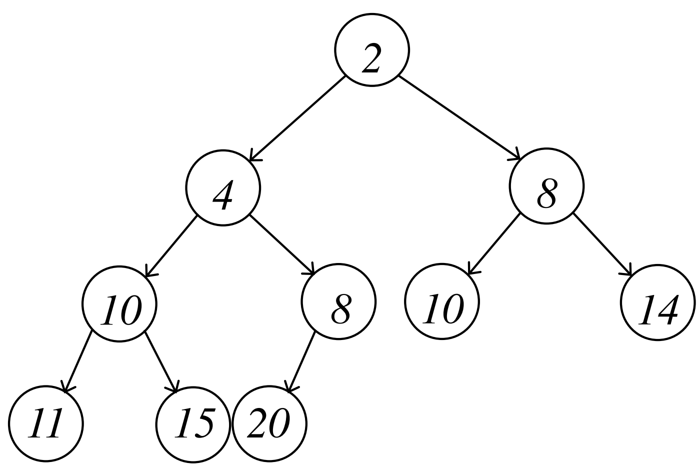
	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$	$\theta(\log(n))$	$\theta(\log(n))$
Heap			

Heap



Heap

Let T be a binary tree.

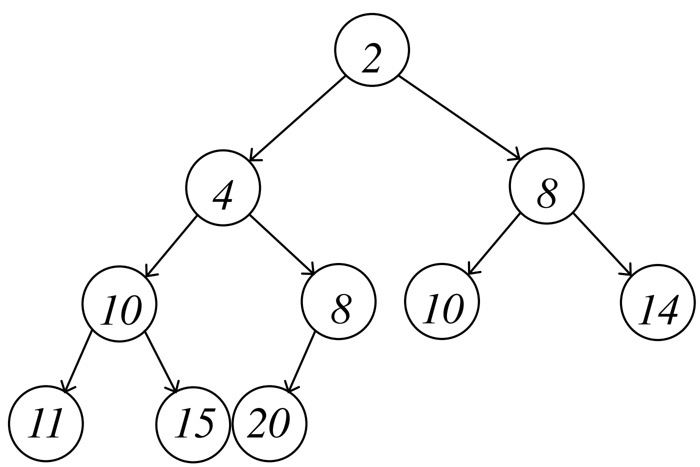


Heap

Let T be a binary tree. We say that T is a *Heap* if it satisfies:

1.

2.

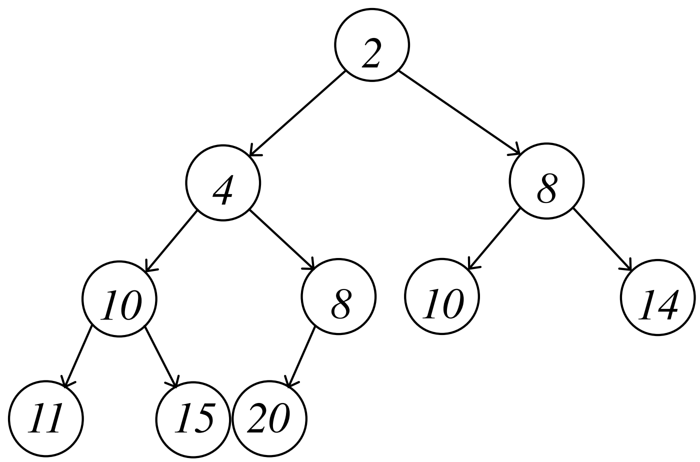


Heap

Let T be a binary tree. We say that T is a *Heap* if it satisfies:

1. Heap order property

2.

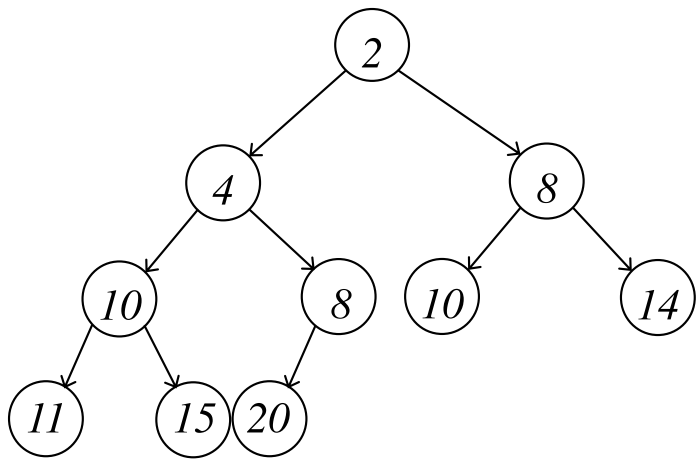


Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

1. Heap order property

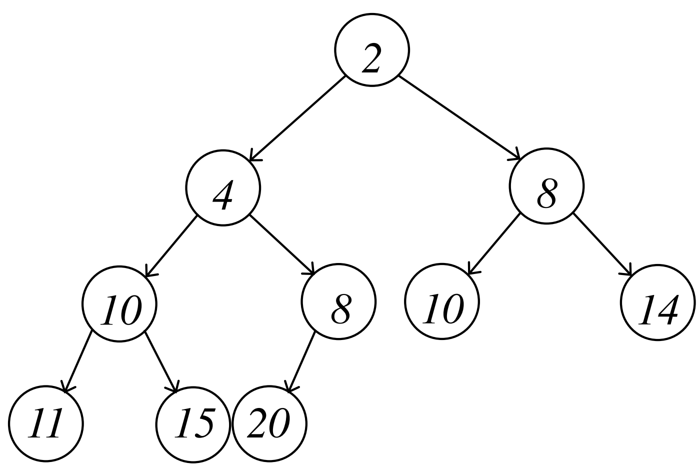
2. Nearly-complete property



Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

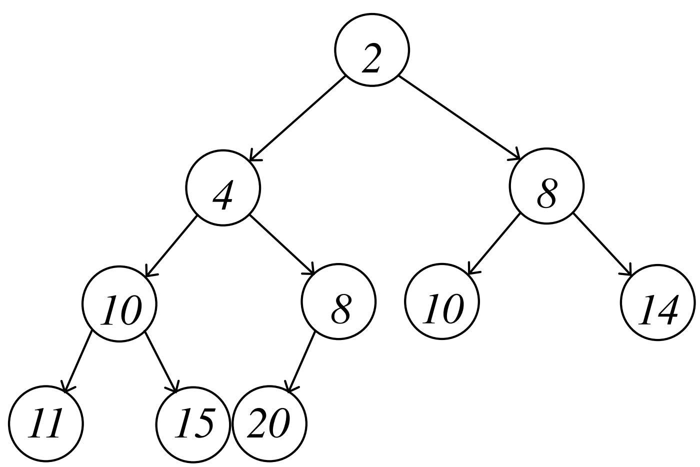
1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property



Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

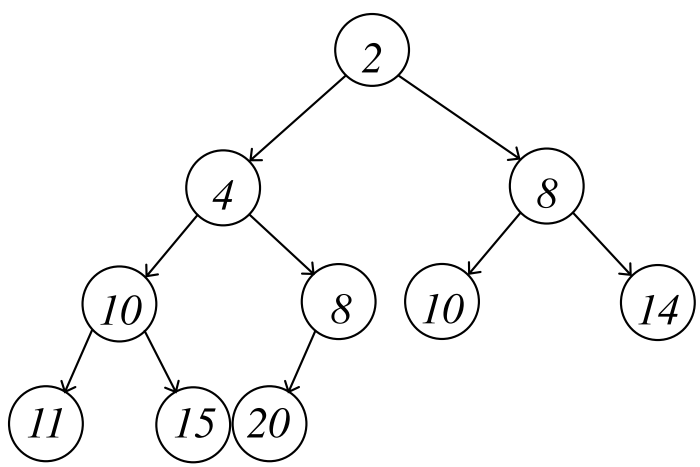
1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property – If *h* is the height of *T*, then all levels: *0, 1, 2, …, h-1* have the maximum number of nodes possible (that is, level *i* has 2^i nodes)



Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

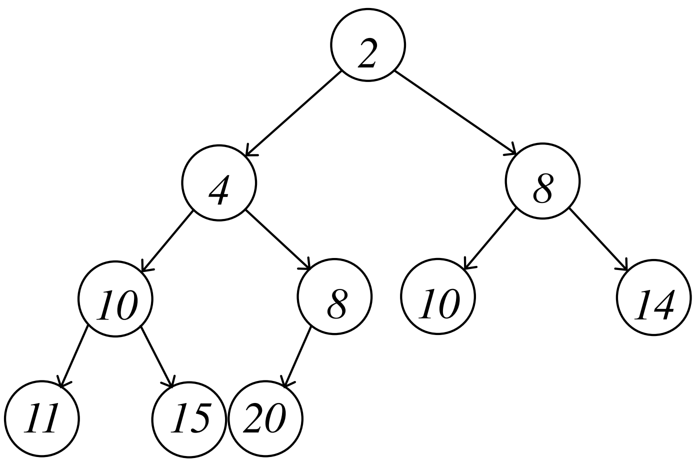
1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property – If *h* is the height of *T*, then all levels: *0, 1, 2, …, h-1* have the maximum number of nodes possible (that is, level *i* has 2^i nodes), and the remaining nodes, at level *h*, reside in the leftmost possible positions.



Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property – If *h* is the height of *T*, then all levels: *0, 1, 2, …, h-1* have the maximum number of nodes possible (that is, level *i* has 2^i nodes), and the remaining nodes, at level *h*, reside in the leftmost possible positions.

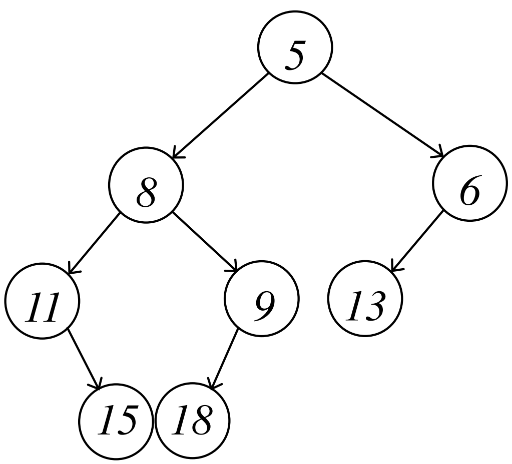


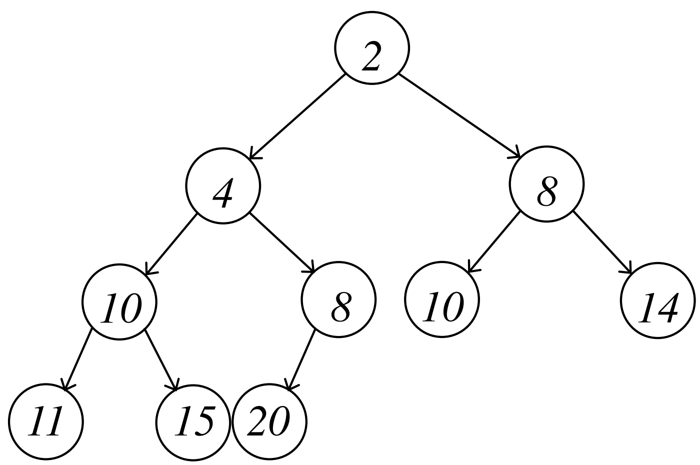
Heap

Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property – If *h* is the height of *T*, then all levels: *0, 1, 2, …, h-1* have the maximum number of nodes possible (that is, level *i* has 2^i nodes), and the remaining nodes, at level *h*, reside in the leftmost possible positions.



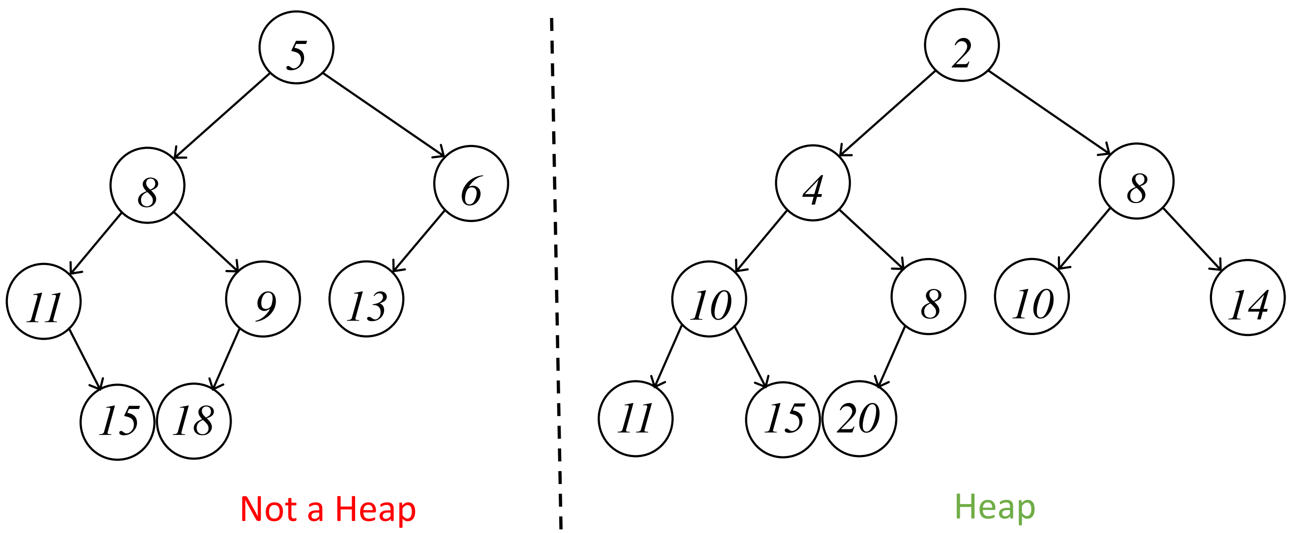


Heap

Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

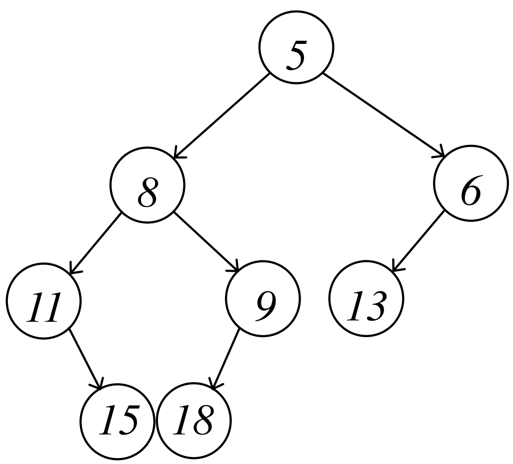
1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property – If *h* is the height of *T*, then all levels: *0, 1, 2, …, h-1* have the maximum number of nodes possible (that is, level *i* has 2^i nodes), and the remaining nodes, at level *h*, reside in the leftmost possible positions.



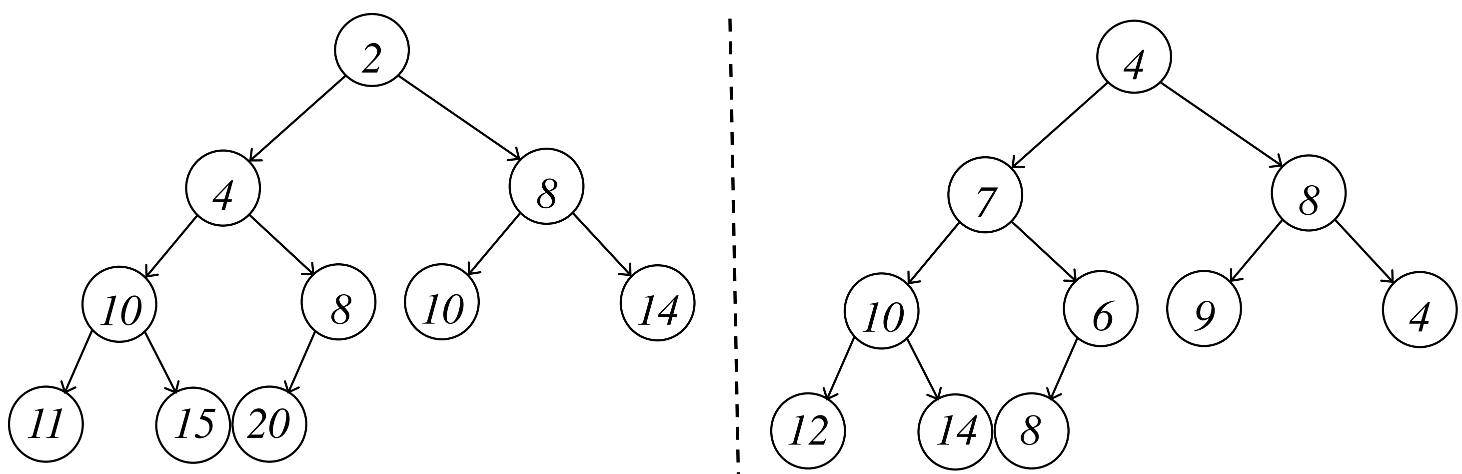
Heap

Let T be a binary tree. We say that T is a *Heap* if it satisfies:

1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property – If *h* is the height of *T*, then all levels: *0, 1, 2, …, h-1* have the maximum number of nodes possible (that is, level *i* has 2^i nodes), and the remaining nodes, at level *h*, reside in the leftmost possible positions.



Not a Heap

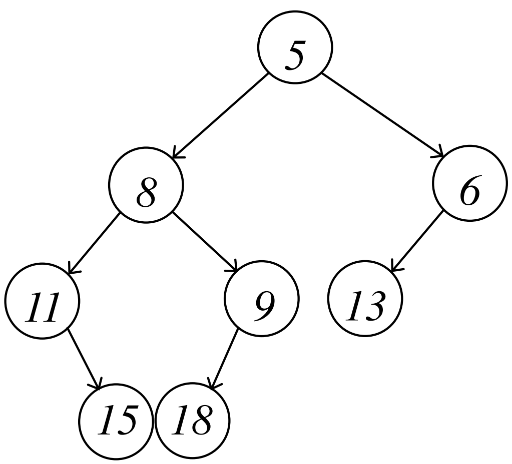


Heap

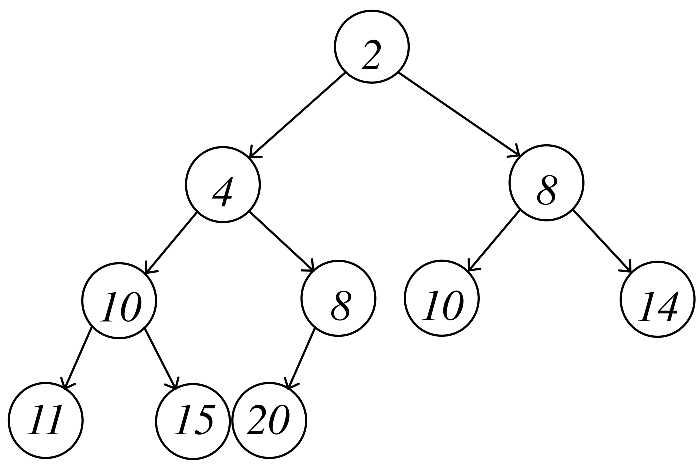
Heap

Let T be a binary tree. We say that T is a $Heap$ if it satisfies:

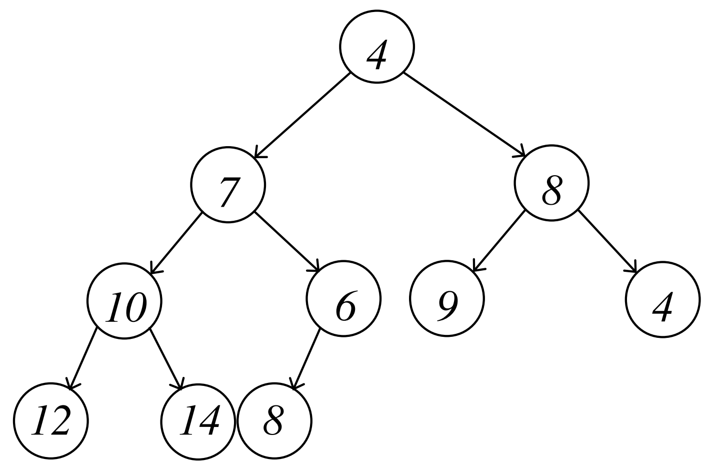
1. Heap order property – In every node *n* of *T*, the priority in *n* is less than or equal to the priorities in *n*'s children.
2. Nearly-complete property – If *h* is the height of *T*, then all levels: *0, 1, 2, …, h-1* have the maximum number of nodes possible (that is, level *i* has 2^i nodes), and the remaining nodes, at level *h*, reside in the leftmost possible positions.



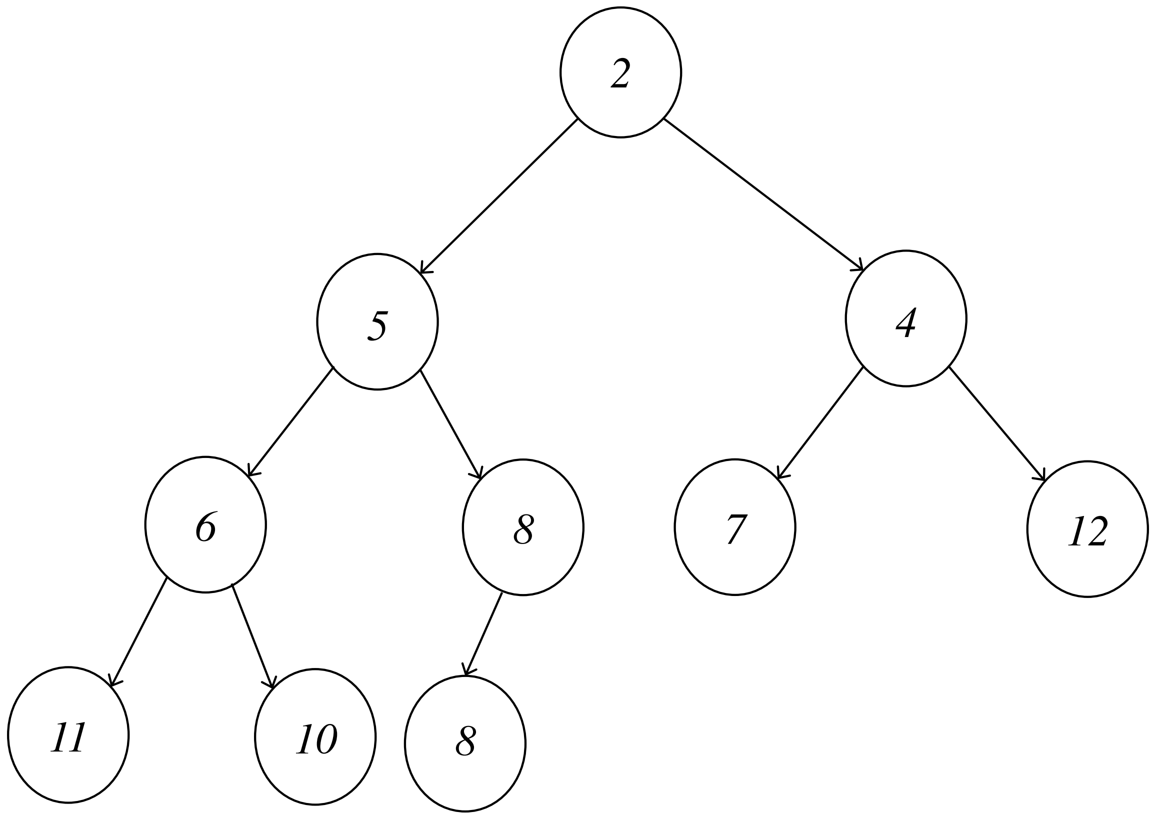
Not a Heap



Heap



Not a Heap


2
5
4
6
8
7
12
11
10
8

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$	$\theta(\log(n))$	$\theta(\log(n))$
Heap			

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$	$\theta(\log(n))$	$\theta(\log(n))$
Heap	$\theta(1)$		

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$	$\theta(\log(n))$	$\theta(\log(n))$
Heap	$\theta(1)$	$\theta(\log(n))$	

Data Structures for *Priority Queue ADT*

	Min	Insert	Delete Min
Unsorted Linked List	$\theta(1)$	$\theta(1)$	$\theta(n)$
Sorted Linked List	$\theta(1)$	$\theta(n)$	$\theta(1)$
Balanced Search Tree	$\theta(1)$	$\theta(\log(n))$	$\theta(\log(n))$
Heap	$\theta(1)$	$\theta(\log(n))$	$\theta(\log(n))$