DATA STRUCTURES

Binary Heap

Session Objectives

- To learn about the concepts of binary heap and priority queue.

Session Outcomes

- At the end of this session, participants will be able to
  - understand Binary heap and priority queue

Agenda

- Binary heap
  - Min heap and max heap
  - Heap operations
  - Priority queue

Binary Heap

Dr. B. Bharathi
SSNCE

Binary Heap

A special kind of binary tree. It has two properties that are not generally true for other trees:

Completeness

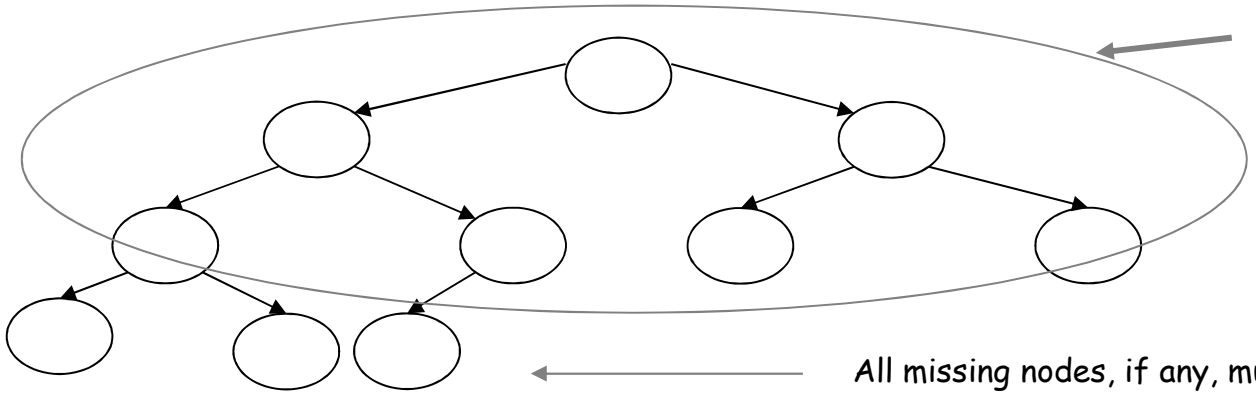
The tree is complete, which means that nodes are added from top to bottom, left to right, without leaving any spaces. A binary tree is completely full if it is of height, *h*, and has $2^{h+1}-1$ nodes.

Heapness

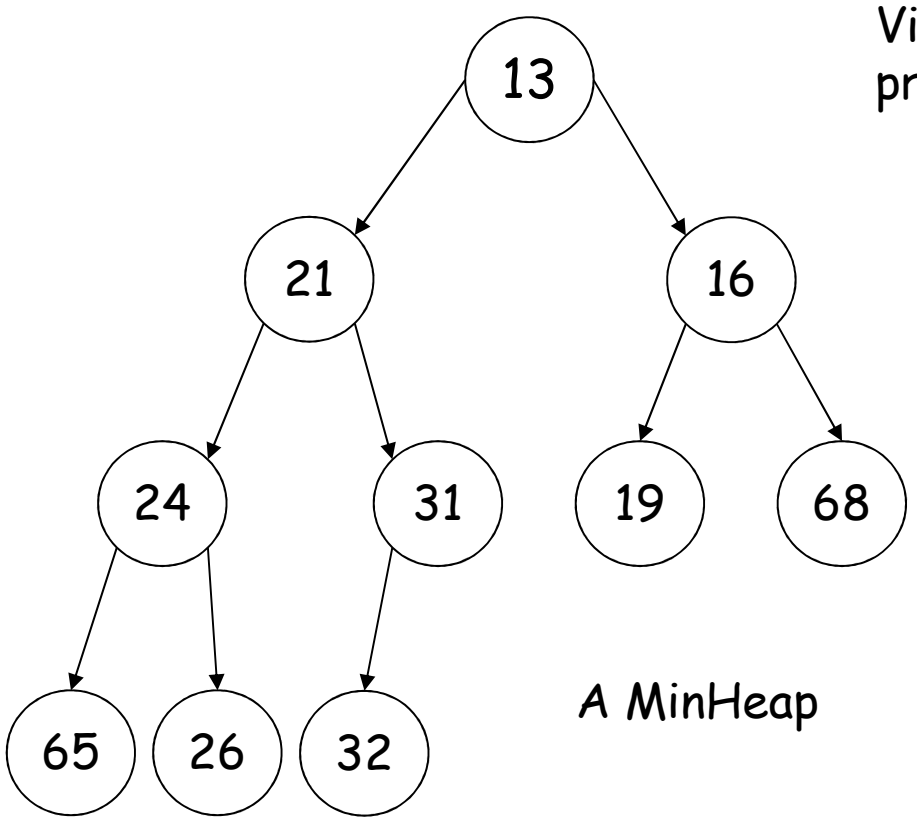
The item in the tree with the highest priority is at the top of the tree, and the same is true for every subtree.

Max Heap and Min Heap?

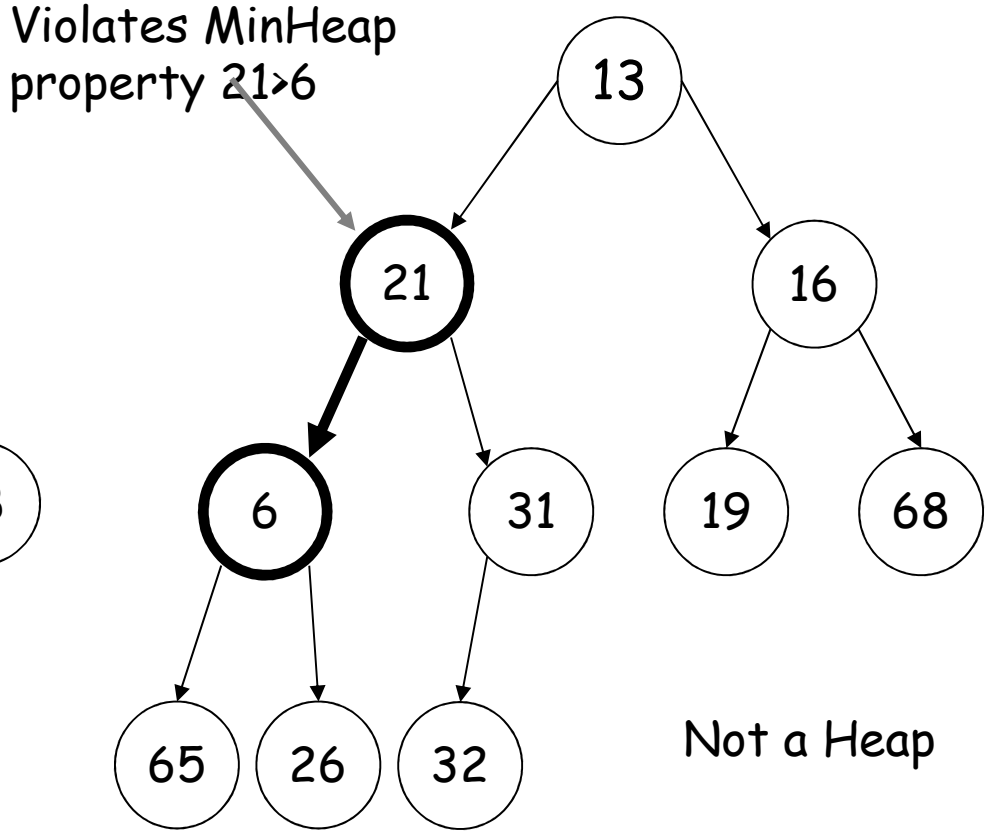
- A **binary heap** is a *complete binary tree* with one (or both) of the following heap order properties:
  - **MinHeap property:** Each node must have a key that is less or equal to the key of each of its children.
  - **MaxHeap property:** Each node must have a key that is greater or equal to the key of each of its children.
- A binary heap satisfying the MinHeap property is called a MinHeap.
- A binary heap satisfying the MaxHeap property is called a MaxHeap.
- A binary heap with all keys equal is both a MinHeap and a MaxHeap.
- Recall: A complete binary tree may have missing nodes only on the right side of the lowest level.



MinHeap and non-MinHeap examples

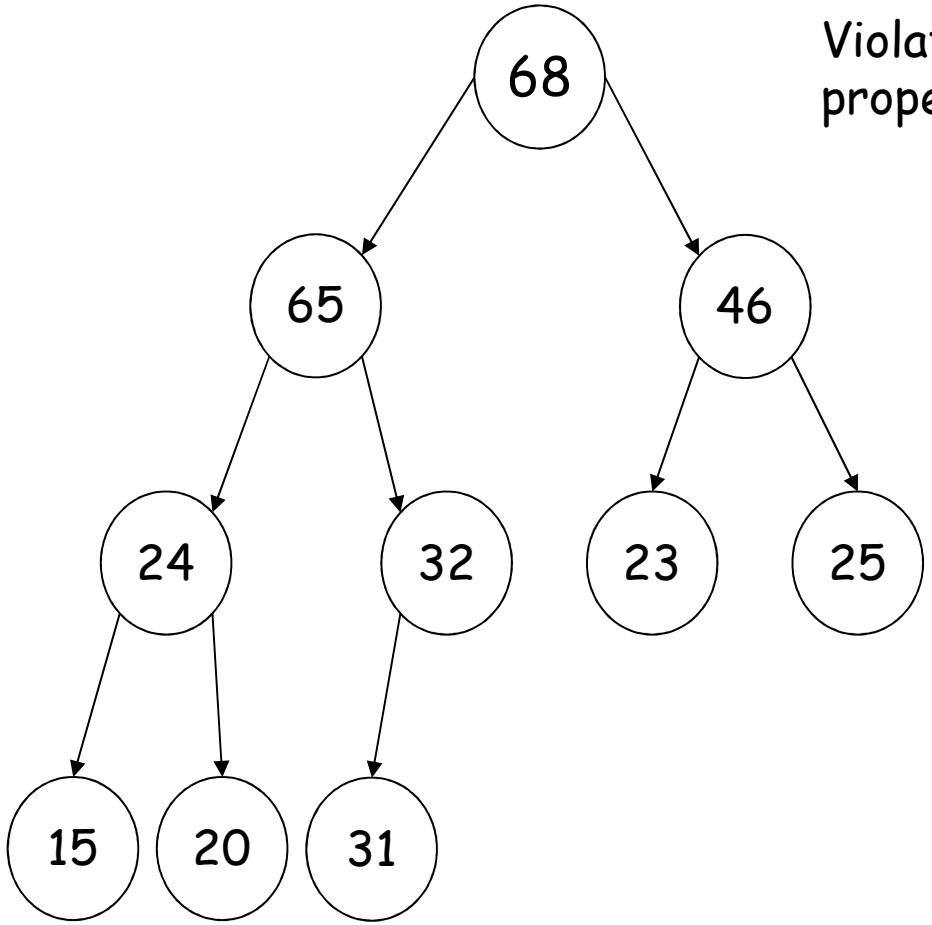


A MinHeap

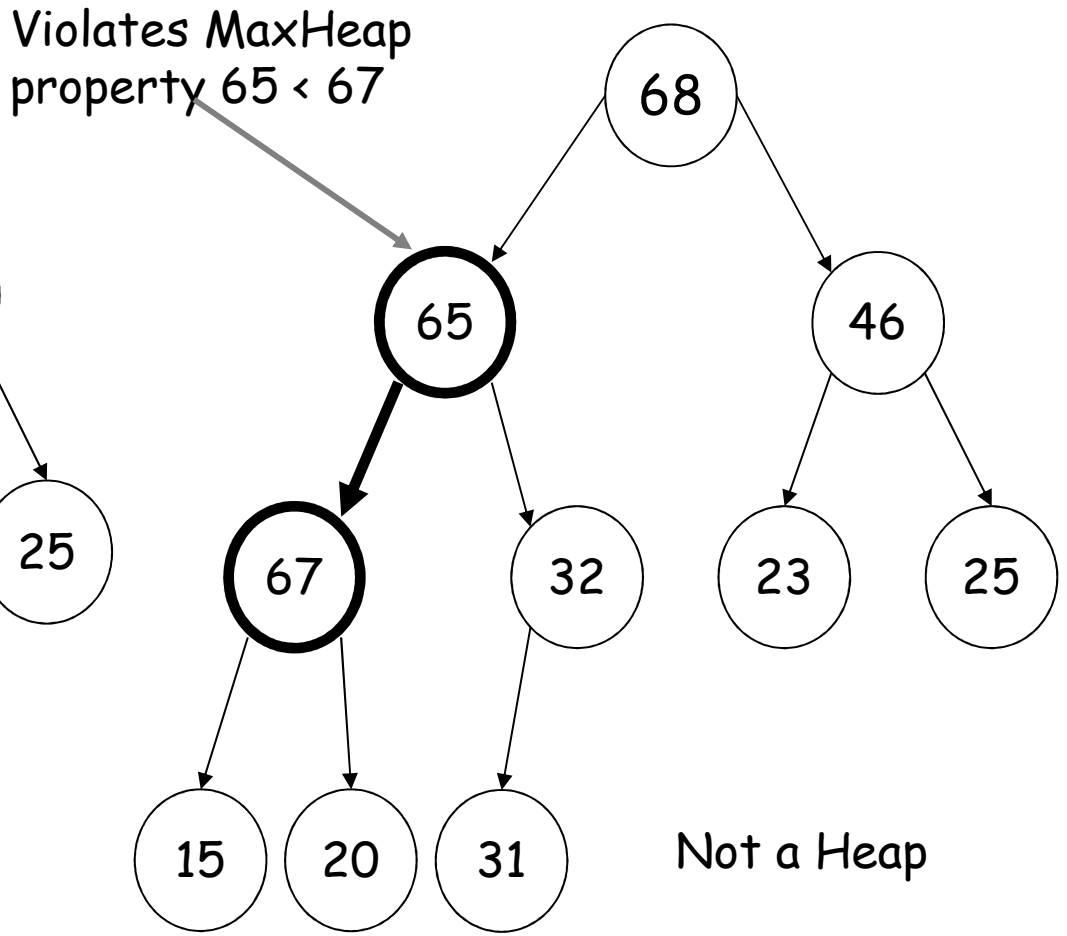


Not a Heap

MaxHeap and non-MaxHeap examples



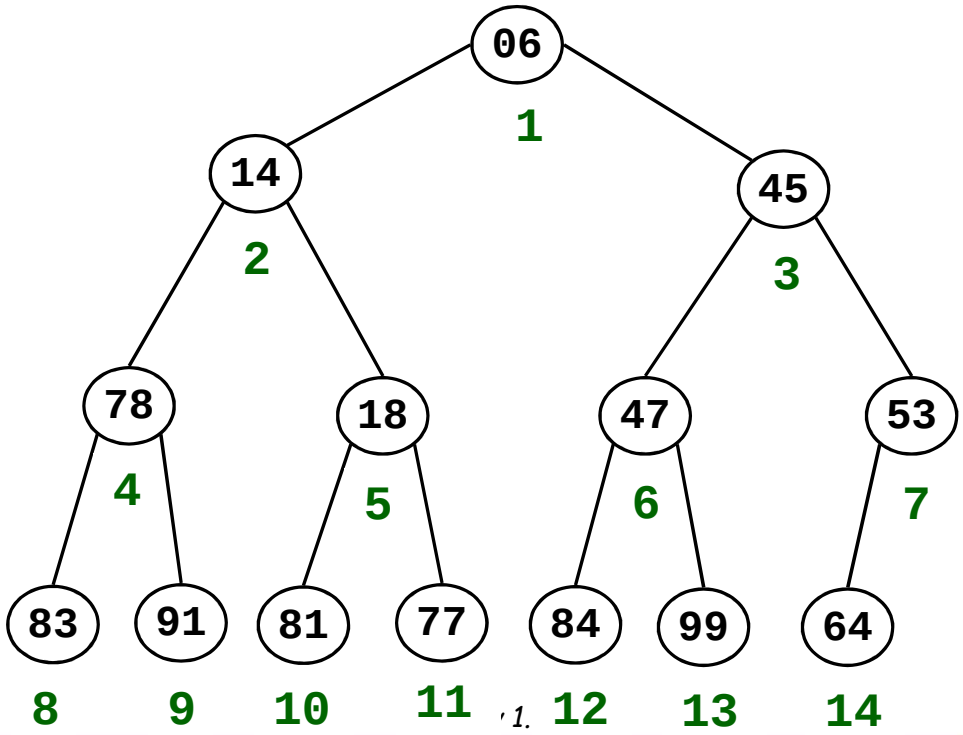
A MaxHeap



Not a Heap

Binary Heaps: Array Implementation

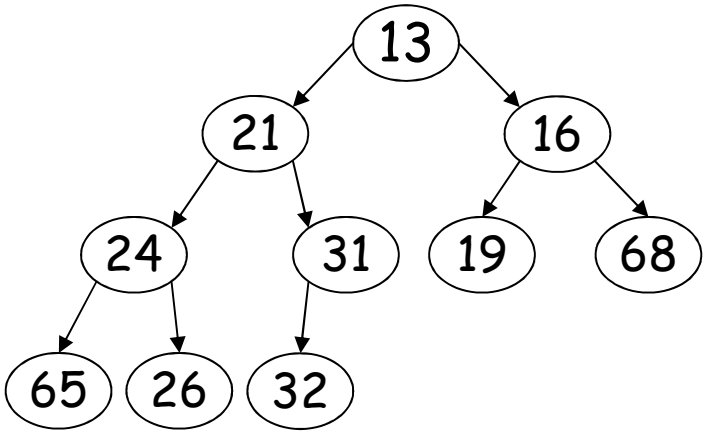
- Implementing binary heaps.
  - Use an array: no need for explicit parent or child pointers.
    - `Parent(i) = [i/2]`
    - `Left(i) = 2i`
    - `Right(i) = 2i + 1`



Array Representation of a Binary Heap

- A heap is a dynamic data structure that is represented and manipulated more efficiently using an array.
- Since a heap is a complete binary tree, its node values can be stored in an array, without any gaps, in a breadth-first order, where

Value(node $_i$) ⟶ array[i] , for i ≥ 1



	13	21	16	24	31	19	68	65	26	32
0	1	2	3	4	5	6	7	8	9	10

- The root is array[1].
- The parent of array[i] is array[i/2], where i > 1
- The left child, if any, of array[i] is array[2i].
- The right child, if any, of array[i] is array[2i+1].

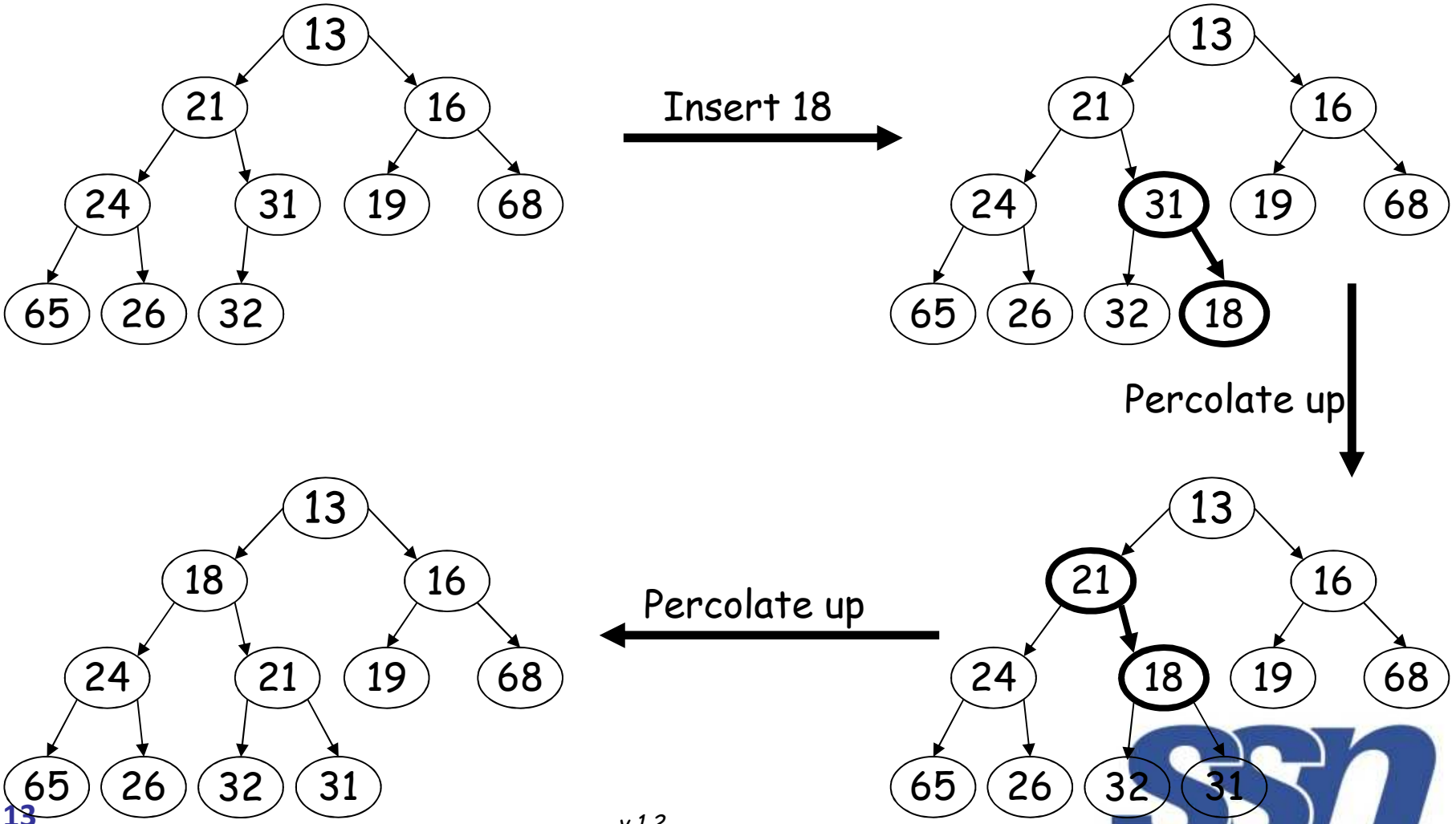
MinHeap enqueue

The process of swapping an element with its parent, in order to restore the heap order property is called percolate up, sift up, or reheapification upward.

The steps for enqueue are:

1. Enqueue the key at the end of the heap.
2. As long as the heap order property is violated, percolate up.

MinHeap Insertion Example



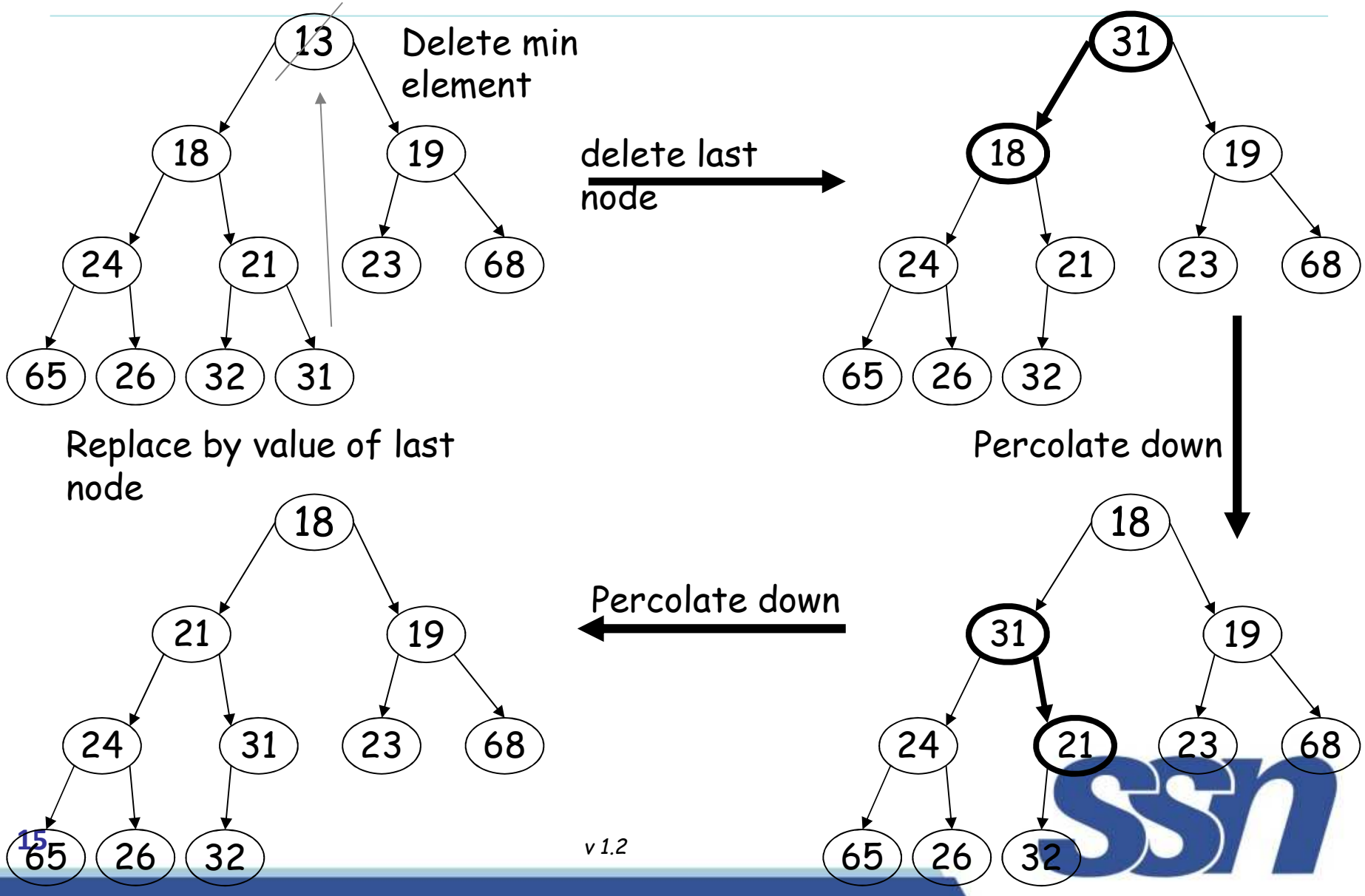
MinHeap dequeue

The process of swapping an element with its child, in order to restore the heap order property is called percolate down, sift down, or reheapification downward.

The steps for deletion are:

1. Replace the key at the root by the key of the last leaf node.
2. Delete the last leaf node.
3. As long as the heap order property is violated, percolate down.

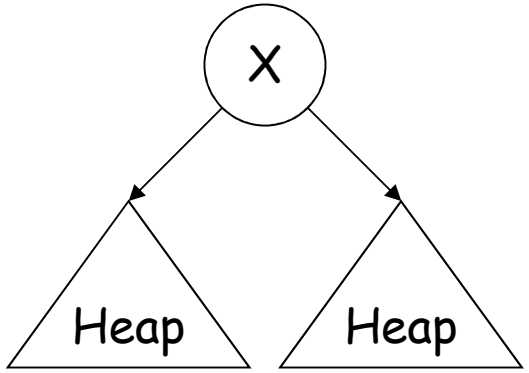
MinHeap Dequeue Example



Heap Applications: Priority Queue

- A heap can be used as the underlying implementation of a priority queue.
- A priority queue is a data structure in which the items to be inserted have associated priorities.
- Items are withdrawn from a priority queue in order of their priorities, starting with the highest priority item first.
- Priority queues are often used in resource management, simulations, and in the implementation of some algorithms (e.g., some graph algorithms, some backtracking algorithms).

Priority Queue Algorithm



X is the element with highest priority

```
priorityQueueEnque(priority
 Queue,el)
{
 If(priorityQueue is full)
     return PQueueFull;
 Insert el at the end of the
  priorityQueue;
 While(e1 is not in the root node
  and e1 < parent(e1))
       swap(e1,parent(e1));
 }
```

Contd.

```
priorityQueueDequeue(priorityQueue)
{
   if(priorityQueue is empty)
            return PQueueEmpty;
   Extract the highest priority element from the root;
   If(root is a leaf node)
   { delete root ; return; }
   Copy the element from the last leaf to the root;
   delete last leaf;
   p = root;
   while(p is not a leaf node and p > any of its children)
       swap p with the smaller child;
   return;
}
```

Priority Queue Declaration

```
typedef struct heapstruct *priorityqueue;
#define mindata 0
struct heapstruct
{
      int capacity;
      int size;
      elementtype *elements;
};
priorityqueue init(int maxelements)
{
      priorityqueue h;
      if(maxelements<minpqsize)
            printf("PQueue is too small");
      h=malloc(sizeof(struct heapstruct)
      if(h==NULL)
            printf("Out of space");
      h->elements=malloc(maxelements+1)*sizeof(elementtype));
      if(h->elements==NULL)
            printf("Out of Space");
      h->capacity=maxelements;
      h->size=0;
      h->elements[0]=mindata;
      return h;
}
```

Procedure for insertion

```
void insert(elementtype x, priorityqueue h)
{
    int i;
    if(isfull(h))
    {
        printf("PQueue is full");
        return;
    }
    for(i=++h->size;h->elements[i/2]>x; i/=2)
        h->elements[i] = h->elements[i/2];
    h->elements[i]=x;
}
```

Function to perform Deletemin in a binary heap

```
elementtype deletemin(Priorityqueue h)
{
    int i, child;
    elementtype minelement, lastelement;
    if(isempty(h))
    {
        printf("PQueue is empty");
        return h->elements[0];
    }
    minelement=h->elements[1];
    lastelement=h->elements[h->size--];
    for(i=1;(i *2)< =h->size;i=child)
    {
        child=i*2;
    if(child!=h->size && h->elements[child+1] < h->elements[child])
        child++:
    if(lastelement > h->elements[child])
        h->elements[i]=h->elements[child];
    else
        break;
    }
    h->elements[i]=lastelement;
    return minelement;
}
```

Summary

- Binary heap
  - Min heap and max heap
  - Heap operations
  - Priority queue