DATA STRUCTURE AND ALGORITHMS

LECTURE 6

Priority Queues and Heaps

Reference links:

https://cs.nyu.edu/courses/fall17/CSCI-UA.0102-007/notes.php

https://www.comp.nus.edu.sg/~stevenha/cs2040.html

https://visualgo.net/en/heap

[M.Goodrich, chapter 9]

Lecture outline

- Priority Queue ADT - Kiểu Hàng đợi ưu tiên
  - Introduction
  - Specification
  - Implementation
- Heap ADT– Kiểu Đống
  - Definitions
  - Array-based Implementation of Heap
  - Using Heap representing Priority Queue
- Application: Heap-Sort – Sắp xếp vun đống

Priority Queue ADT

Priority Queue: Introduction

- Priority is a kind of queue
- Dequeue gets element with the highest priority
  - Different from normal queue: dequeue the first enqueued element first.
- Priority is based on a comparable value (key) of each object (smaller value higher priority, or higher value higher priority)
- Example Applications:
  - printer -> print (dequeue) the shortest document first
  - operating system -> run (dequeue) the shortest job first

Priority Queue: Specification

- Priority Queue ADT Methods

insert(k, v)	*//Creates an entry with key k and value v in the priority queue.*
min()	*// Returns a priority queue entry (k,v) having minimal key*
removeMin()	*// Removes an entry (k,v) having minimal key from the priority queue*
size()	*//Returns the number of entries in the priority queue.*
isEmpty()	*// Returns a boolean indicating whether the priority queue is empty*

For more details see [M. Goodrich, p361]

- Compare with Queue ADT Methods

enqueue(e)	⇨	insert(k, v)
first()	⇨	min()
dequeue()	⇨	removeMin()

Priority Queue: Specification

Method	Return Value	Priority Queue Contents
insert(5,A)		{ (5,A) }
insert(9,C)		{ (5,A), (9,C) }
insert(3,B)		{ (3,B), (5,A), (9,C) }
min( )	(3,B)	{ (3,B), (5,A), (9,C) }
removeMin( )	(3,B)	{ (5,A), (9,C) }
insert(7,D)		{ (5,A), (7,D), (9,C) }
removeMin( )	(5,A)	{ (7,D), (9,C) }
removeMin( )	(7,D)	{ (9,C) }
removeMin( )	(9,C)	{ }
removeMin( )	null	{ }
isEmpty( )	true	{ }

Example operations on a priority queue with interger key
[M.Goodrich, p361]

Priority Queue: Implementation

- Priority Queue Class in Java
  https://docs.oracle.com/javase/9/docs/api/java/util/PriorityQueue.html
- Implement using list (array-based or linked list)
  - Unsorted list
    - insert takes O(1) time
    - removeMin takes O(N) time - *Find min algorithm*
  - Sorted list
    - insert takes O(N) time - *Insertion sort algorithm*
    - removeMin takes O(1) time

⇨ Need an other data structure for better running time of both methods.

Priority Queue: Effective Stratergy

- Use list for implementing a priority queue ADT is an interesting trade-off (sự đánh đổi thú vị)
- There a more efficient strategy, using a data structure called a binary heap (đống nhị phân)
  - To perform both insertion and removal methods in logarithmic time
  - The fundamental way the heap achieves this improvement is to use the structure of a binary tree to find a compromise between elements being sorted in somehow. (sử dụng cấu trúc cây nhị phân với sự thỏa hiệp giữa các phần tử được sắp xếp theo cách nào đó)

Heaps ADT

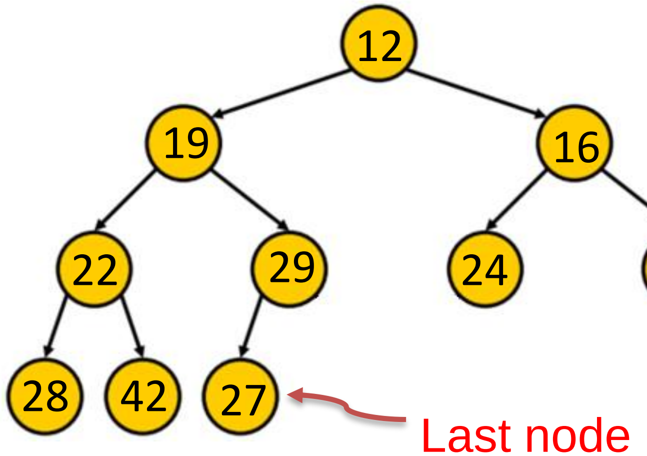
Heap: What is a heap?

- A heap is a binary tree storing keys at its nodes and satisfying the following properties:
  - Heap-Order: for every internal node v other than the root, key(v) ≥ key(parent(v)) - Minimum Heap
  - Complete Binary Tree: let h be the height of the heap, size of tree (number of nodes) between 2^h and 2^{h+1}-1. The last node of the heap is the rightmost node of maximum depth
  - Example:

    Tree T is a heap (integer key)

    h = 3

    $2^3 \leq$ size(T)=10 $\leq 2^4$-1



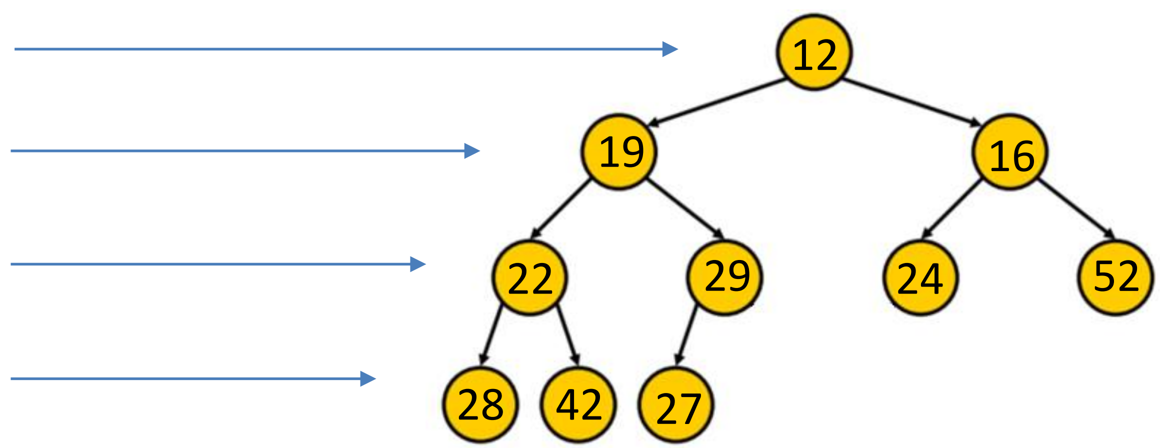
Heap: The height

- **Theorem**: A heap store n nodes has height $O(\log n)$

  Proof: (apply the complete binary tree property)

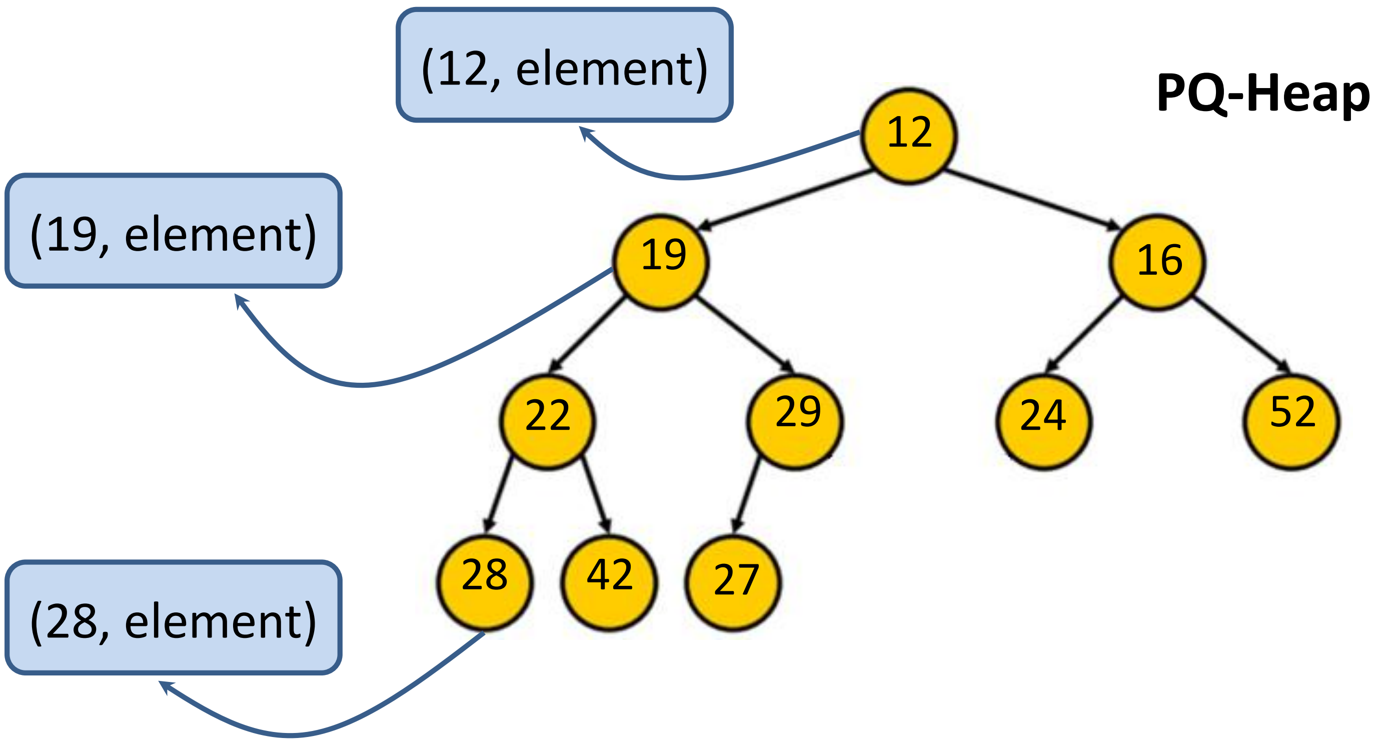
  - Let h be the height of a heap storing n keys (n nodes)
  - Since there are 2^i nodes at depth $i = 0, \ldots, h - 1$ and at least one node at depth h, we have $n \geq 1 + 2 + 4 + \ldots + 2^{h-1} + 1$
  - Thus, $n \geq 2h$, i.e., $h \leq \log n$

height	nodes
0	1
1	2
h-1	2^{h-1}
h	3



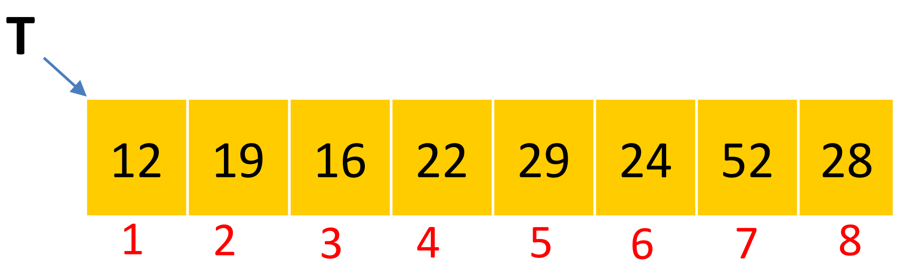
Heap: Representing Priority Queue

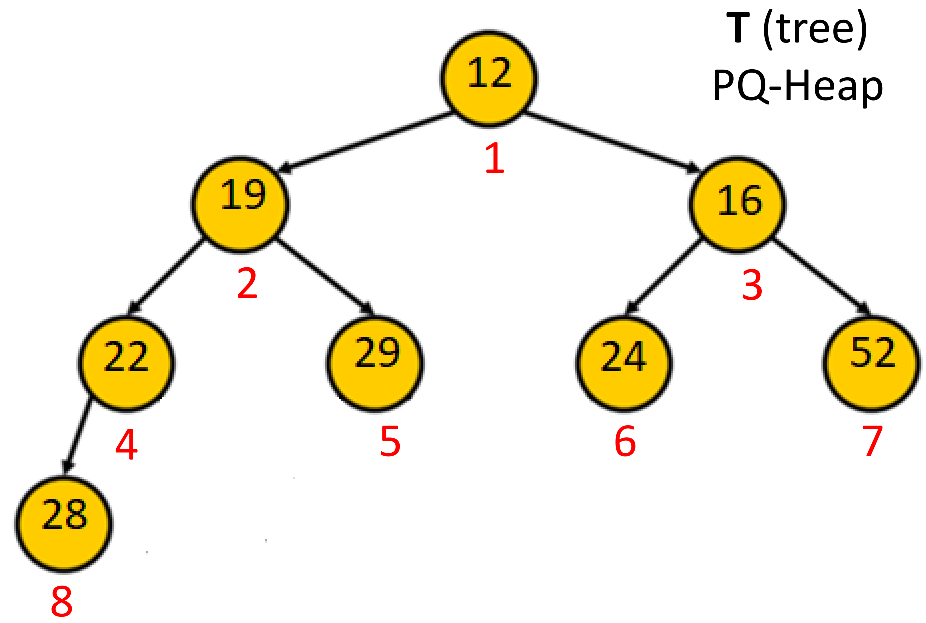
- We can use a heap to implement a priority queue
  - Store a (key, element) item at each node
  - Keep track of the position of the last node



PQ-Heap: Array-based Implementation

- Represent a heap with n keys in an array of length n
- For the node at index i
  - the left child is at index 2i
  - the right child is at index 2i + 1
  - the parent is at (i-1)/2 (i>0)

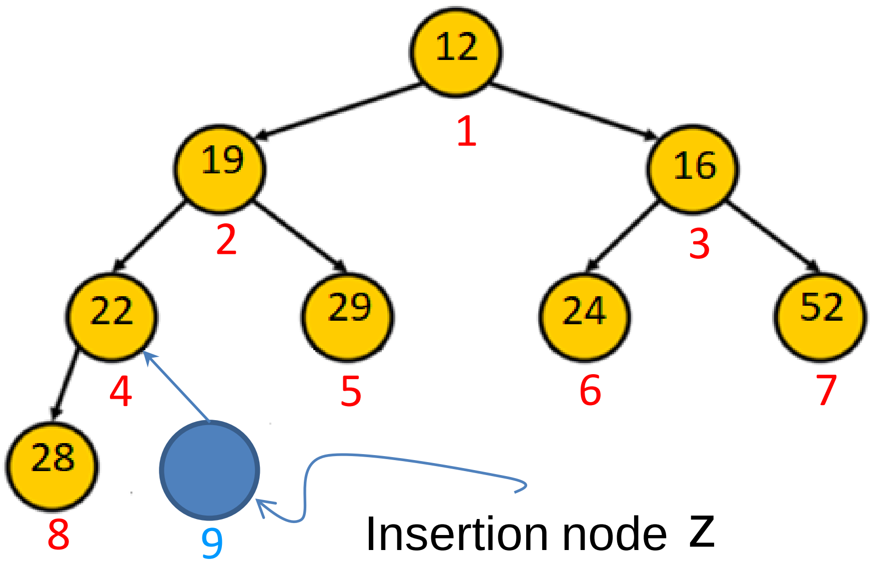


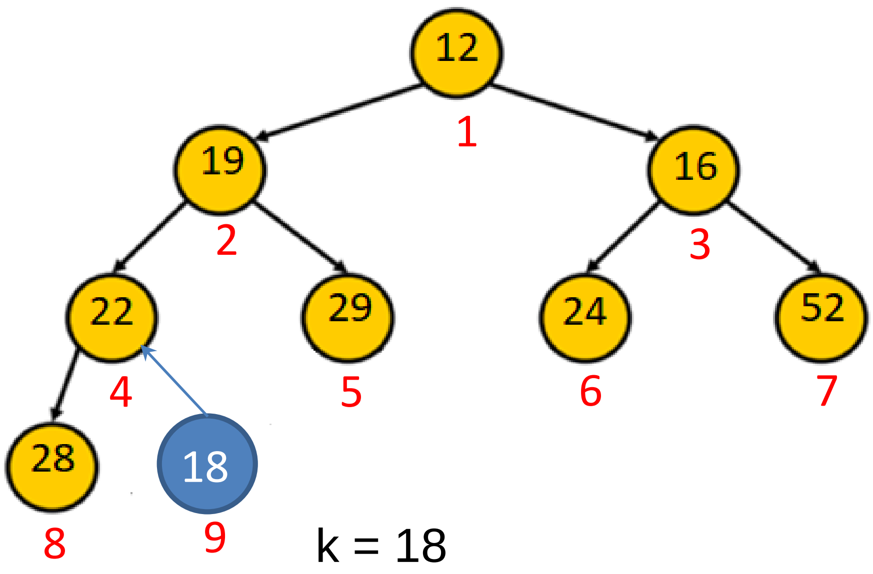


- How to insert, removeMin nodes from the PQ-heap T

PQ-Heap: Insert

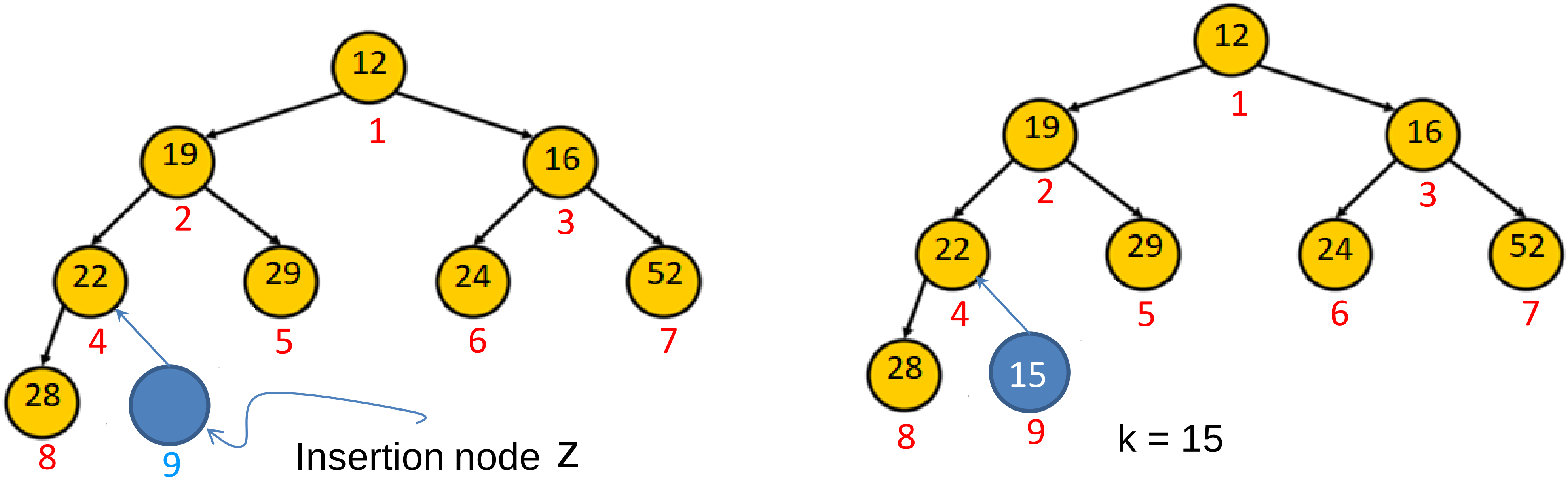
- Insert a node with key k to the PQ-Heap: the algorithm has three steps
  - Create a hole in the next available location z (after last node)
  - Store k at z
  - Restore the heap-order property (up-heap)





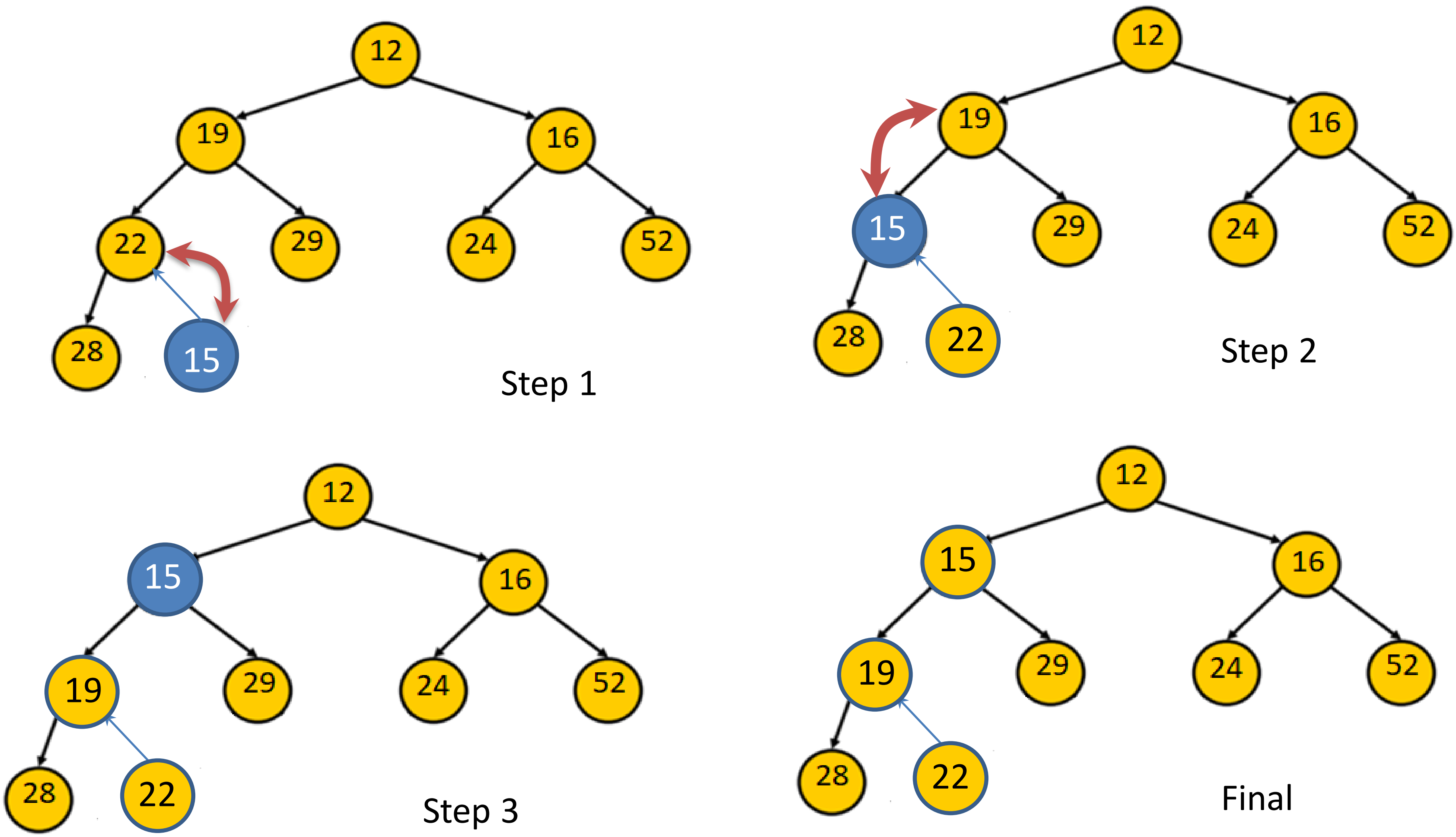
PQ-Heap: Insert

- Up-heap algorithm – thuật toán vun đống lên
  - Restores the heap-order property by swapping k along an upward path from the insertion node
  - Upheap terminates when the key k reaches the root or a node whose parent has a key smaller than or equal to k



PQ-Heap: Insert

- Up-heap algorithm – thuật toán vun đống lên



PQ-Heap: Insert

- Insert a node to a PQ-heap algorithm

Algorithm heapInsert(k, e):

__Input__: A key-element pair (k,e)

__Output__: An update of the array T, of n elements, for a heap, to add (k, e)

n $\leftarrow$ n + 1;   *// add a new node*

T[n] $\leftarrow$ (k,e);   *// put key and element to new node*

i $\leftarrow$ n;

while (i > 1 and T[i/2] > T[i])

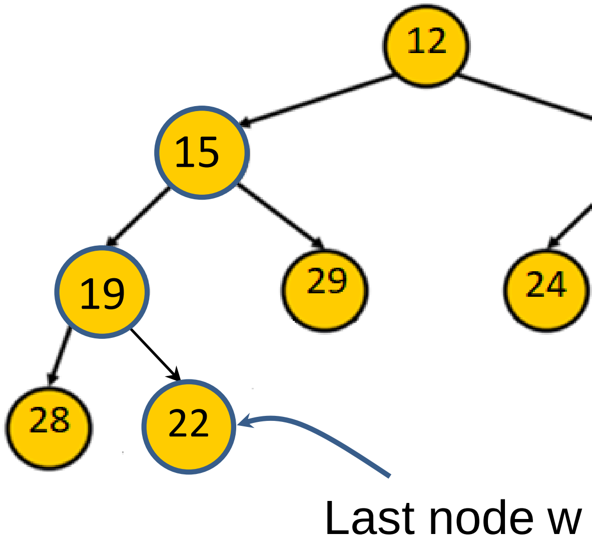
    swap (T[i/2], T[i]);   *// up-heap*

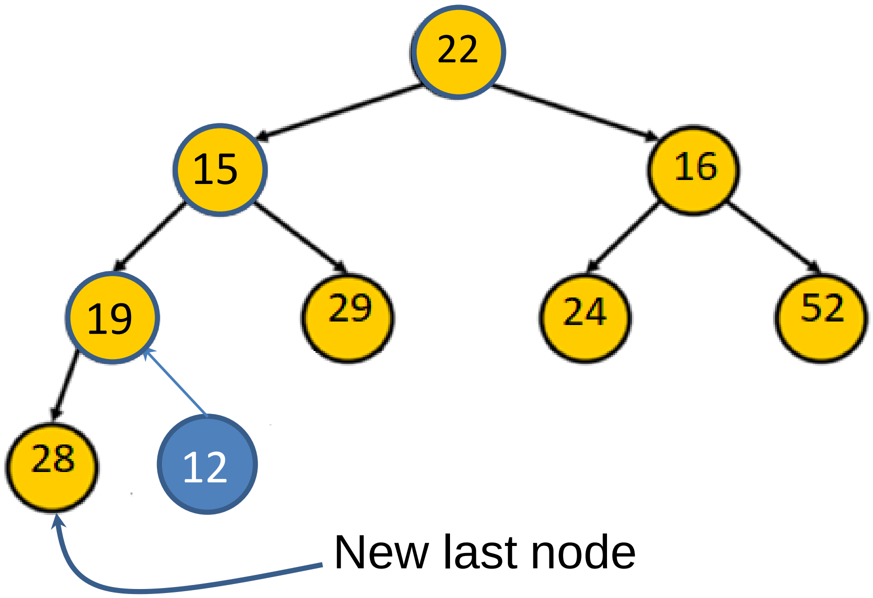
    i $\leftarrow$ i/2;

Running time: O(log n)

PQ-Heap: removeMin

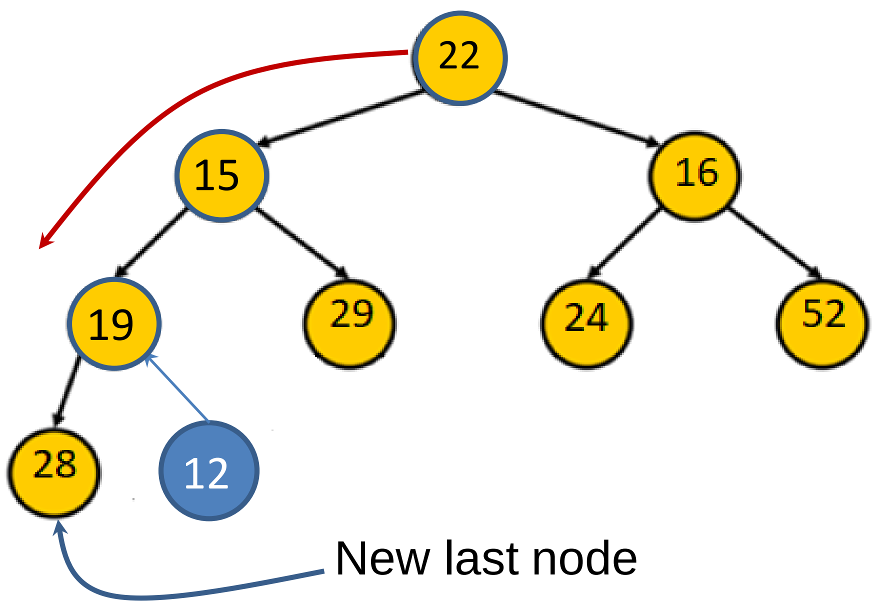
- Remove minimal node from the priority queue corresponds to the removal of the root node of PQ-Heap.
- The algorithm has three steps
  - Replace the root key with the key of the last node w
  - Remove w
  - Restore the heap-order property (down-heap)





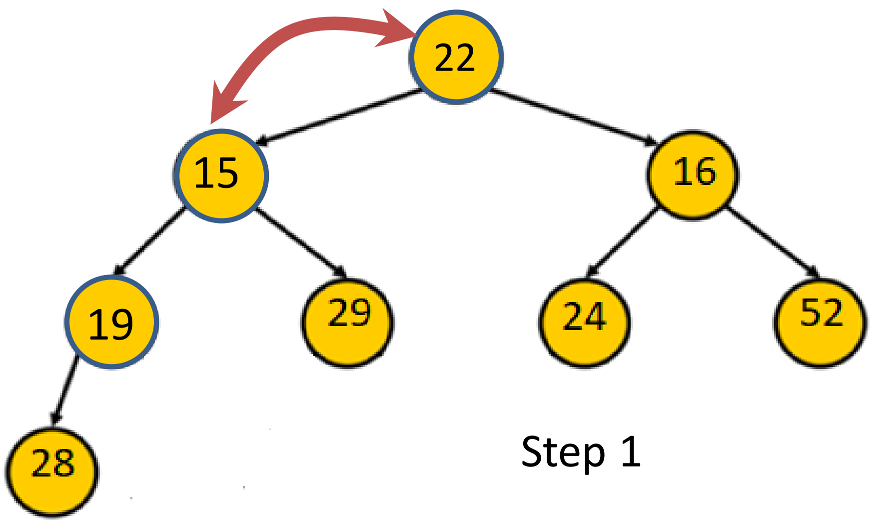
PQ-Heap: removeMin

- Down-heap algorithm – thuật toán vun xuống
  - Restores the heap-order property by swapping key k along a downward path from the root
  - Upheap terminates when key k reaches a leaf or a node whose children have keys greater than or equal to k

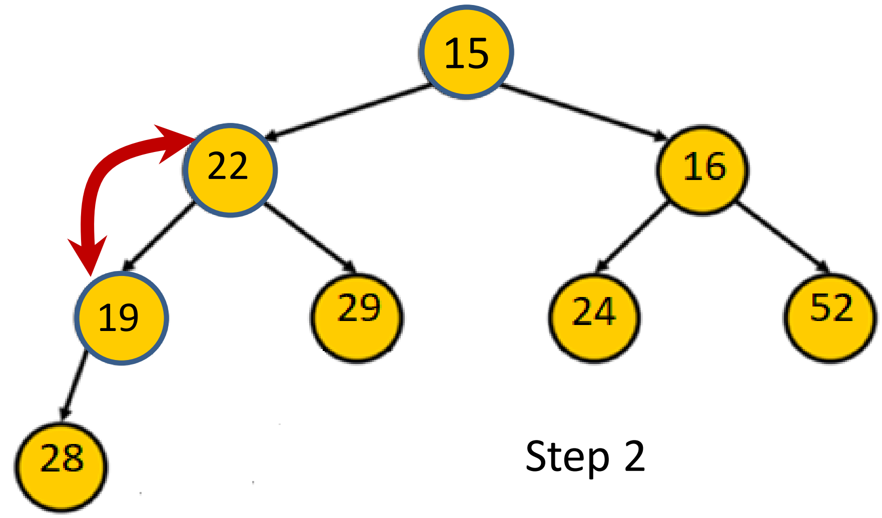


PQ-Heap: removeMin

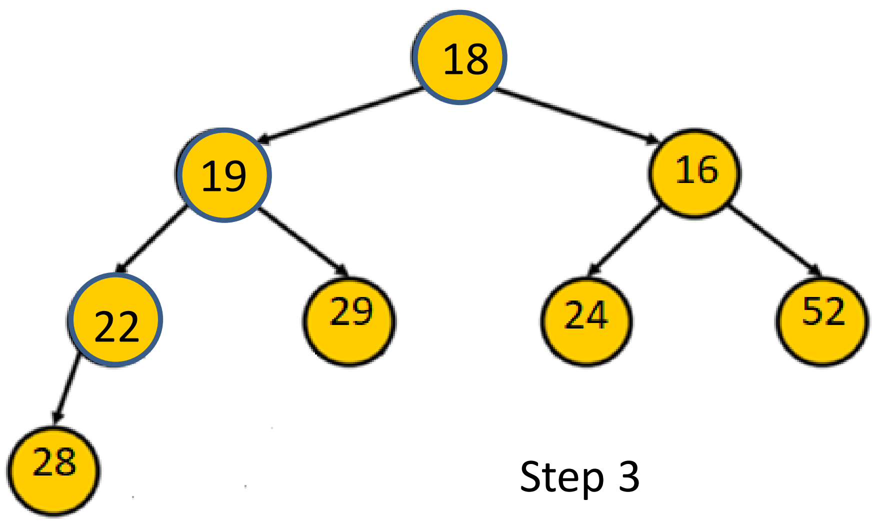
- Down-heap algorithm – thuật toán vun xuống



Step 1



Step 2



18

Step 3

PQ-Heap: removeMin

- Remove minimal node from PQ-heap algorithm

```
Algorithm heapRemoveMin():
    Input: None
    Output: An update of the array, T, of n elements, for a PQ-heap, to
    remove and return an item with smallest key
    temp ← T[1]; T[1] ← T[n]; n ← n – 1;
    i ← 1;
    while (i < n)
        if (2i + 1 < n) // this node has two internal children
            if (T[i] < T[2i] and T[i] < T[2i + 1])
                return temp; // we have restored the heap-order property
            else
                Let j be the index of the smaller of T[2i] and T[2i + 1];
                Swap (T[i], T[j]) ;
                i ← j;
        else // this node has zero or one child
            if (2i < n) // this node has one child (the last node)
                if (T[i] > T[2i])
                    Swap (T[i], T[2i]);
            return temp; // we have restored the heap-order property
        return temp; //reached the last node or an external node
```

PQ-Heap: removeMin

- Remove minimal node from PQ-heap algorithm

```
Algorithm heapRemoveMin():
        ...
        i ← 1;
        while (i < n)
                ...
                        Let j be the index of the smaller of T[2i] and T[2i + 1]
                        Swap (T[i], T[j])
                        i ← j        ⟶        chỉ cần log n lần là tới nút lá
                ...
```

Running time: O(log n)

PQ-Heap Application

Heap Sort

Heap-sort: Idea

- One application of PQ-Heap is sorting, where we rearrange a sequence of elements in increasing/nodecreasing order.

- The algorithm for sorting a sequence A with a PQ-Heap T is quite simple and consists of the following two phases:
  - First: insert the elements of A as keys into T by n insert operations, one for each.
  - Second: extract the elements from T in nondecreasing order by n removeMin operations, putting back into A in that order.

- The resulting algorithm is called heap-sort

Heap-sort: Algorithm

- Heap-sort algorithm

```
Algorithm heapSort(A):
        Input: Array A of elements with 2 properties (k,e) here k-key, e-value
        Output: Sorted array A

        T = new int[A.length]     //Array for stroring

        for (i=0; i < A.length; i++)  //insert all elements of A as keys into T
                heapInsert(A[i].k, A[i].e);

        for (i=0; i < A.length; i++)  //extract the elements from T  to A and A is sorted
                A[i] = heapRemoveMin();
```

- Running time
  - n*O(log n) + n*O(log n) = O(n log n)

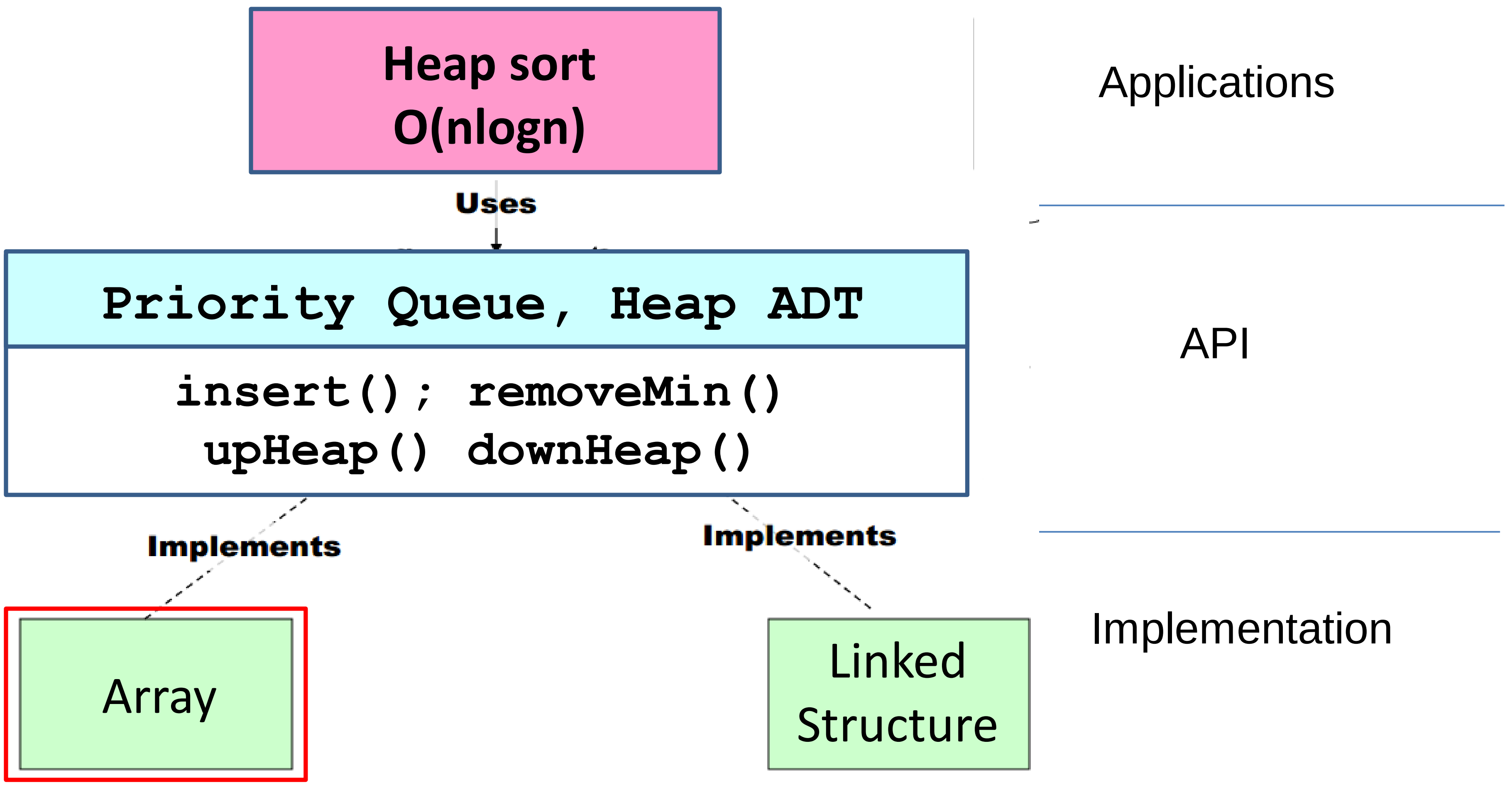
- Does the algorithm stable and in-place???

Heap-sort: Algorithm

- More tricks
  - Make heap-sort in-place: don't need to have a spare array T; work within A only.
  - Improve algorithms build the heap in time O(N) rather than O(N*log N) by building it from bottom up rather than top down.
- How to do? – Challenge!

  [M.Goodrich, p388]

Summary


Heap sort
O(nlogn)
Uses
Priority Queue, Heap ADT
insert(); removeMin()
upHeap() downHeap()
Implements
Implements
Array
Linked Structure
Applications
API
Implementation