Min-Max Heaps

Ch. 9

Part 2

Doubly Ended Heap (deap)

A complete binary tree empty or satisfying these properties:

1. The root contains no element.
2. The left subtree is a min heap.
3. The right subtree is a max heap.
4. If right subtree is not empty, let *i* be any node in the left subtree and *j* be the corresponding node in the right subtree. *i.key* ≤ *j.key*.

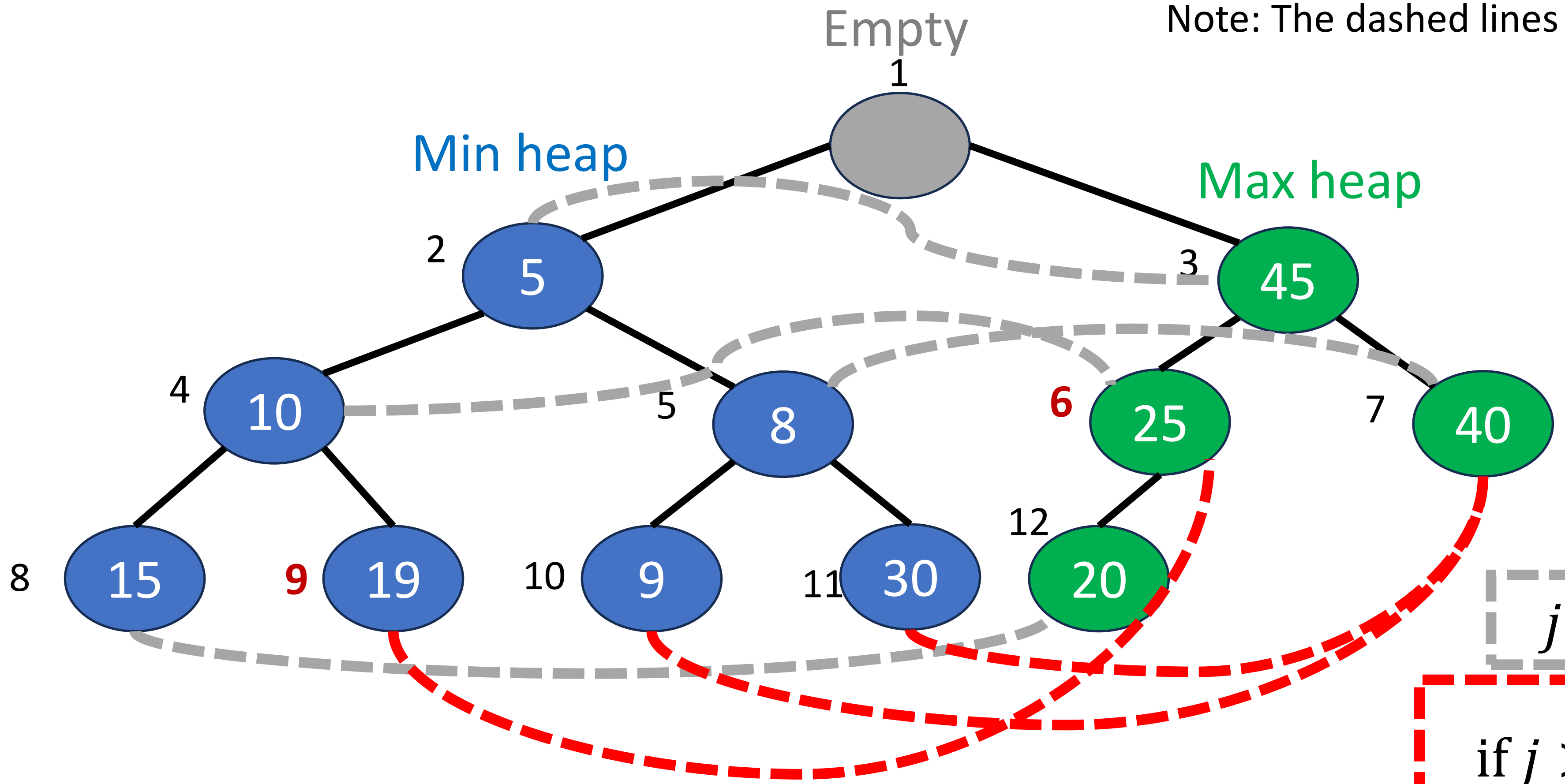
$$j = i + 2^{\lfloor \log_2 i \rfloor - 1}$$

- If *j* does not exist, the corresponding node of *i* is the corresponding node of *i*'s parent.

$$\text{if } j > n \text{ then } j = \text{floor}(\frac{j}{2})$$

Example of deap

Note: The dashed lines are only for visualization.



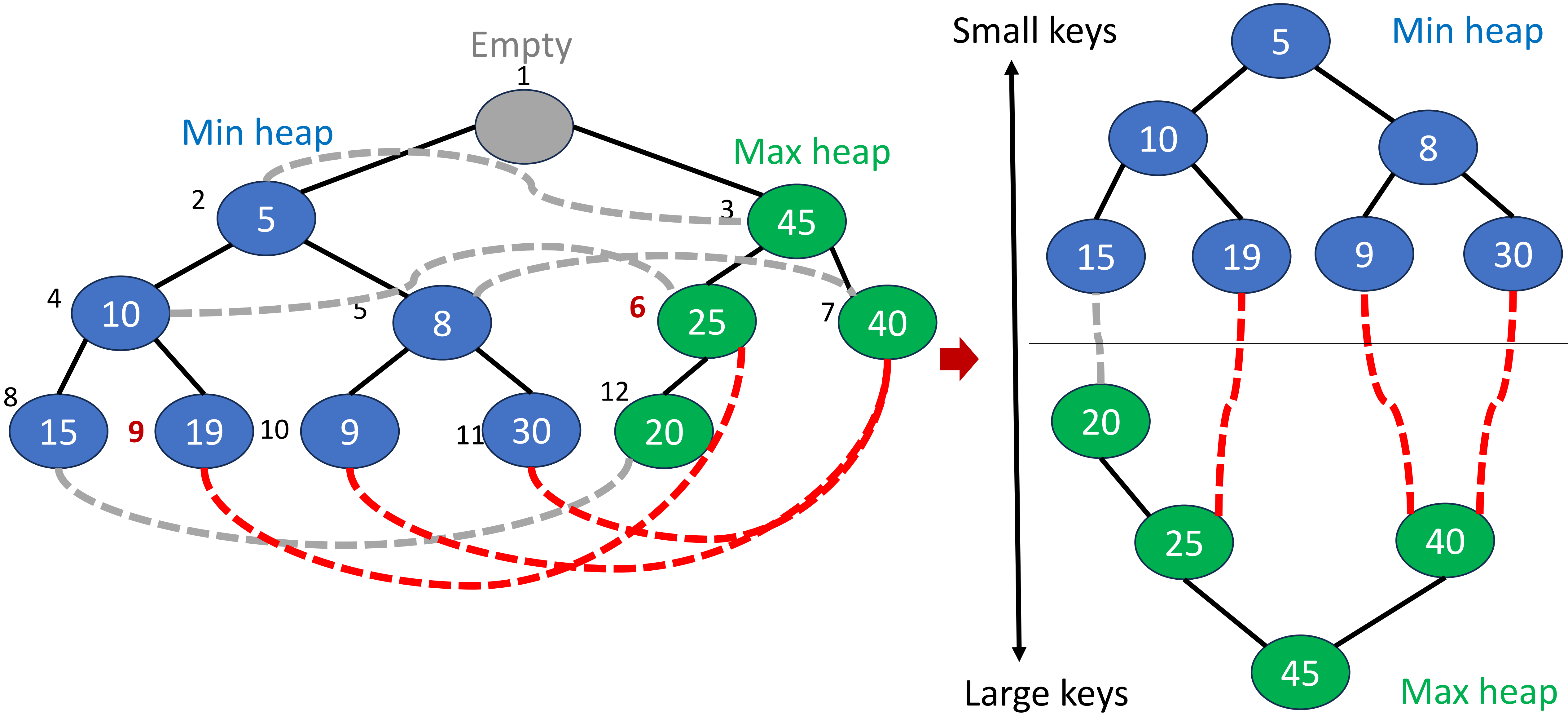
$$j = i + 2^{\lfloor \log_2 i \rfloor - 1}$$

$$\text{if } j > n \text{ then } j = \text{floor}\left(\frac{j}{2}\right)$$

- Min element: root of the min heap
- Max element: root of the max heap.

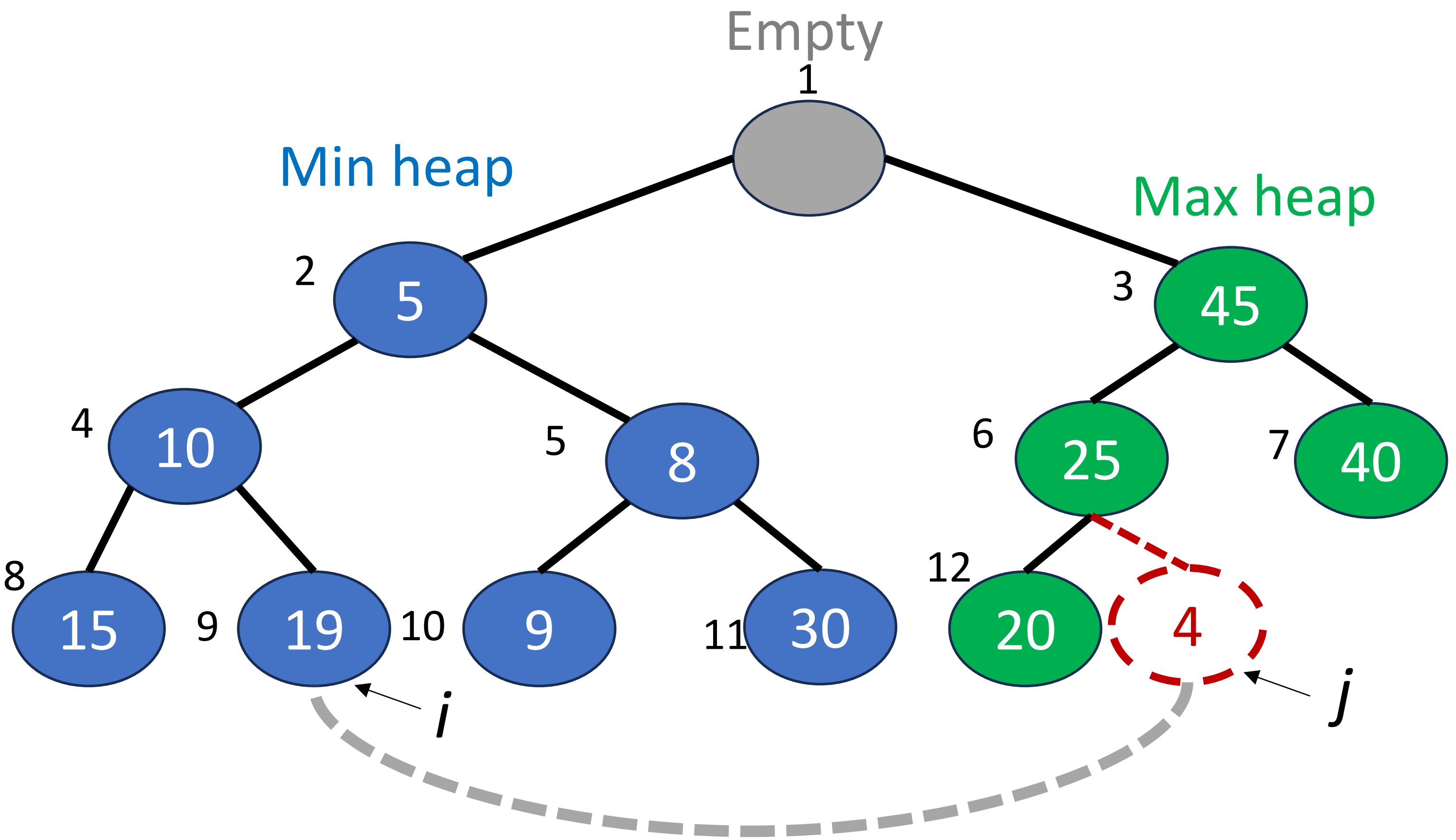
Another view

Note: The dashed lines are only for visualization.



Operation: Insertion

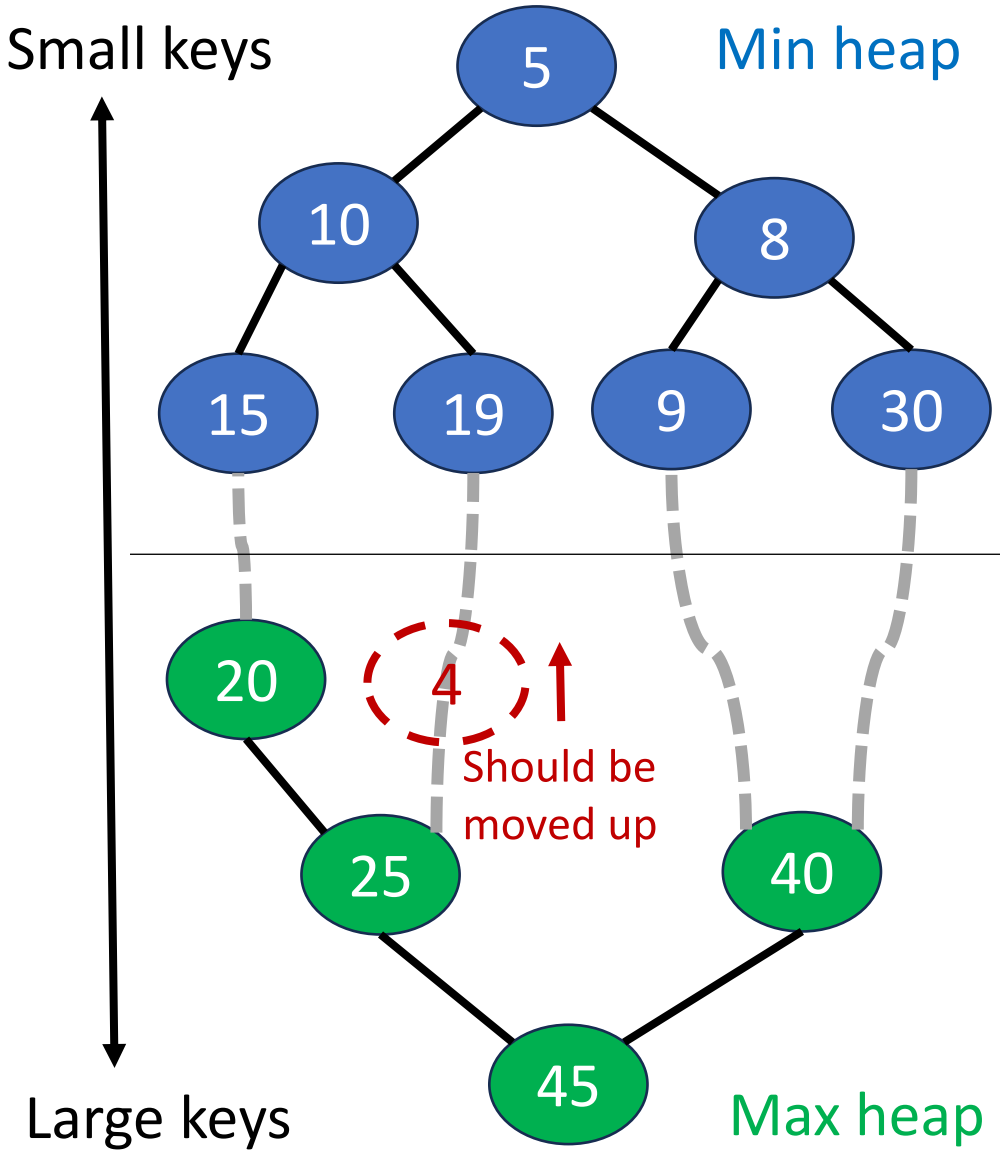
- Example: Insert 4



Step 1: Compare with the corresponding node.

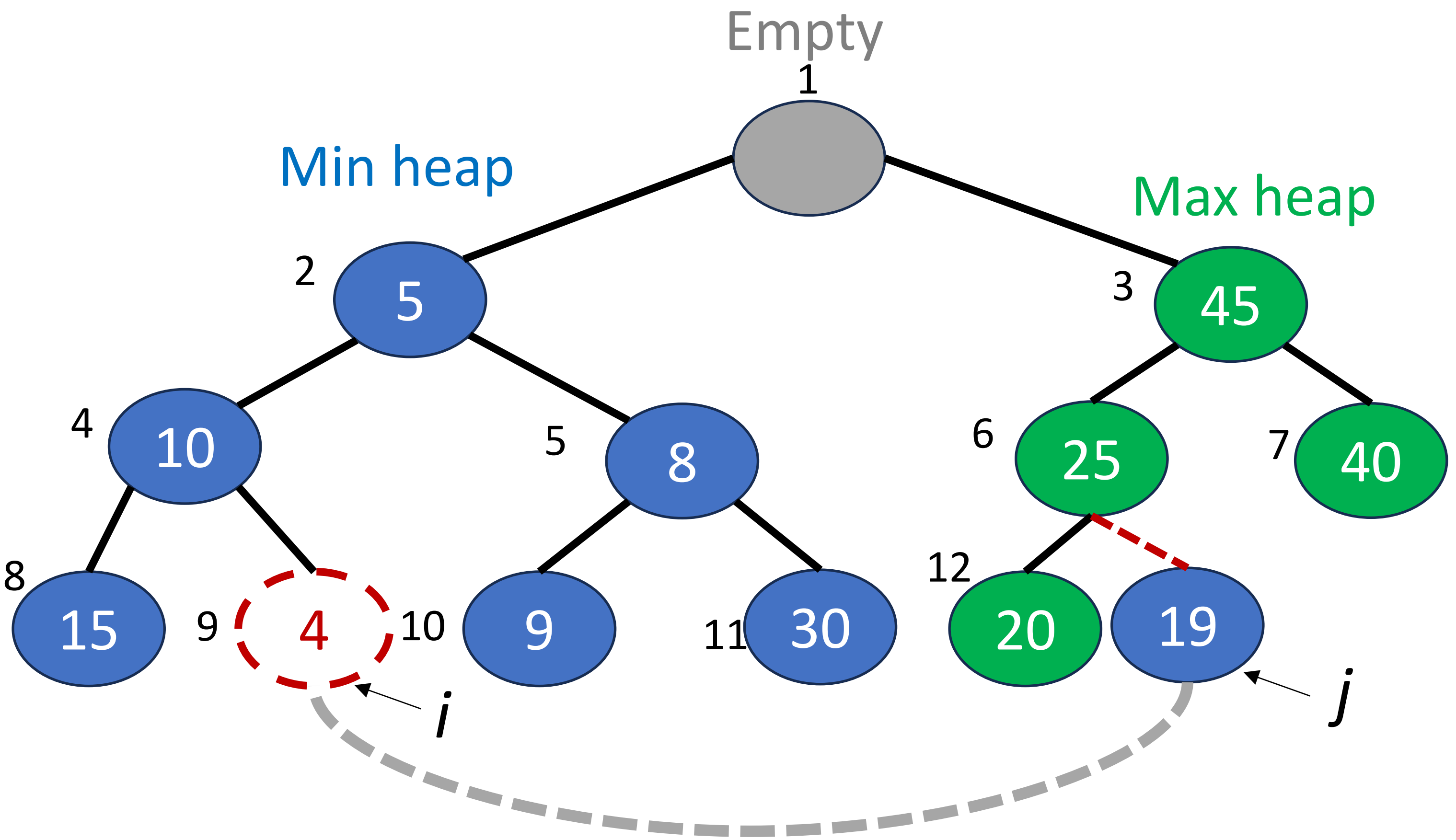
Step 2: Max in the right subtree; Min in the left subtree.

Step 3: Reorganize using min (max) heap insertion.



Operation: Insertion

- Example: Insert 4



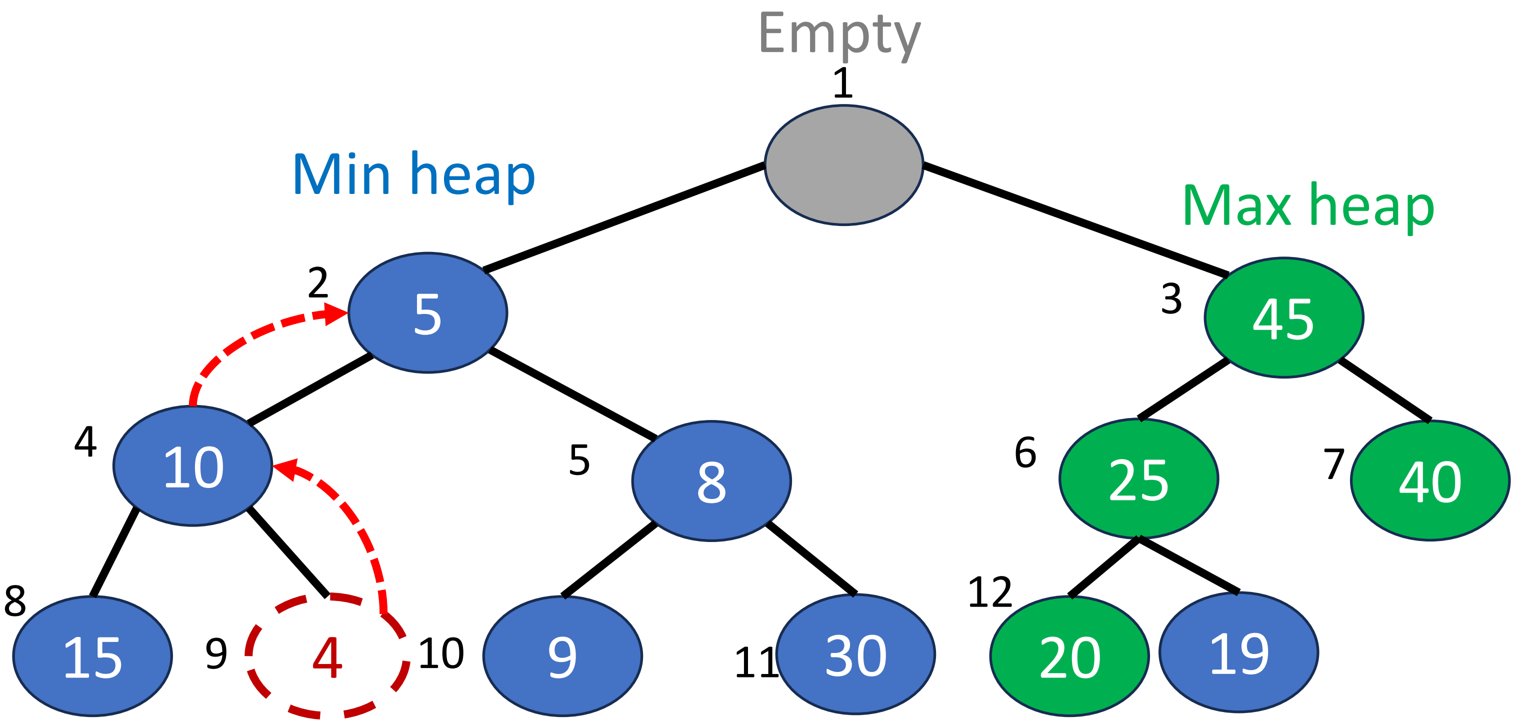
Step 1: Compare with the corresponding node.

Step 2: Max in the right subtree; Min in the left subtree.

Step 3: Reorganize using min (max) heap insertion.

Operation: Insertion

- Example: Insert 4



Step 1: Compare with the corresponding node.

Step 2: Max in the right subtree; Min in the left subtree.

Step 3: Reorganize using min (max) heap insertion.

Insertion algorithm

- Time is linear in the height of the tree.
- Time complexity: O(log *n*)

```
procedure DeapInsert (var d : deap ; var n : integer ; x : element);
{Insert x into the deap d of size n-1.}
var i : integer;
begin
  if n =MaxElements then DeapFull
  else begin
     n := n+1;
     if n = 2 then d[2] := x {insertion into an initially empty deap}
     else case MaxHeap(n) of
        true : begin {n is a position in the max heap}
                 i := MinPartner(n);
                 if x.key < d[i].key
                 then begin
                        d[n] := d[i];
                        MinInsert(d, i, x);
                      end
                 else MaxInsert(d, n, x);
        false : begin {n is a position in the min heap}
                 i := MaxPartner(n);
                 if x.key > d[i].key
                 then begin
                        d[n] := d[i];
                        MaxInsert(d, i, x);
                      end
                 else MinInsert(d, n, x);
     end; {of case and if n = 2}
  end; {of if n = MaxElements}
end; {of DeapInsert}
```

Goal: Insert the element *x*

Insert the element *x* to the last position

Location of *x* is in the max heap

Find the corresponding node *i* in min heap

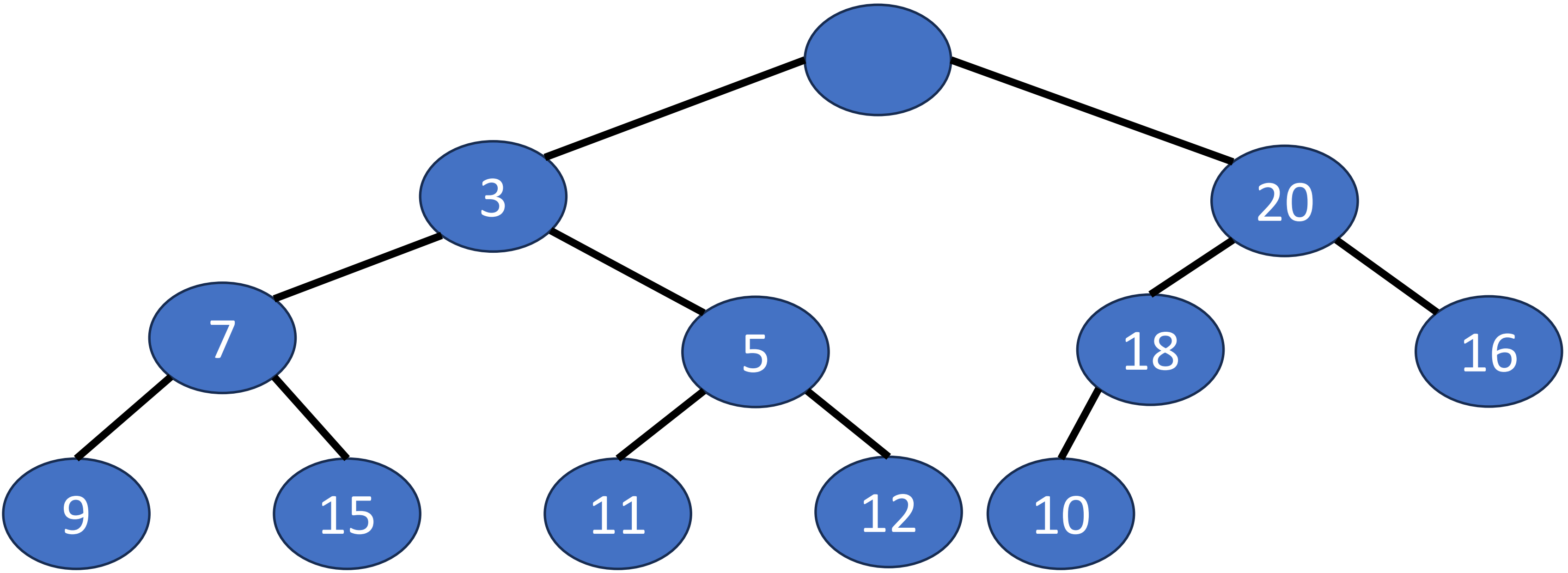
If x.key < i.key, swap them and do min heap insertion. Otherwise, do max heap insertion.

Location of *x* is in the min heap

Find the corresponding node *i* in max heap

Exercise

- Q1: Insert 2 into the following deap. Where will be the location of 2?



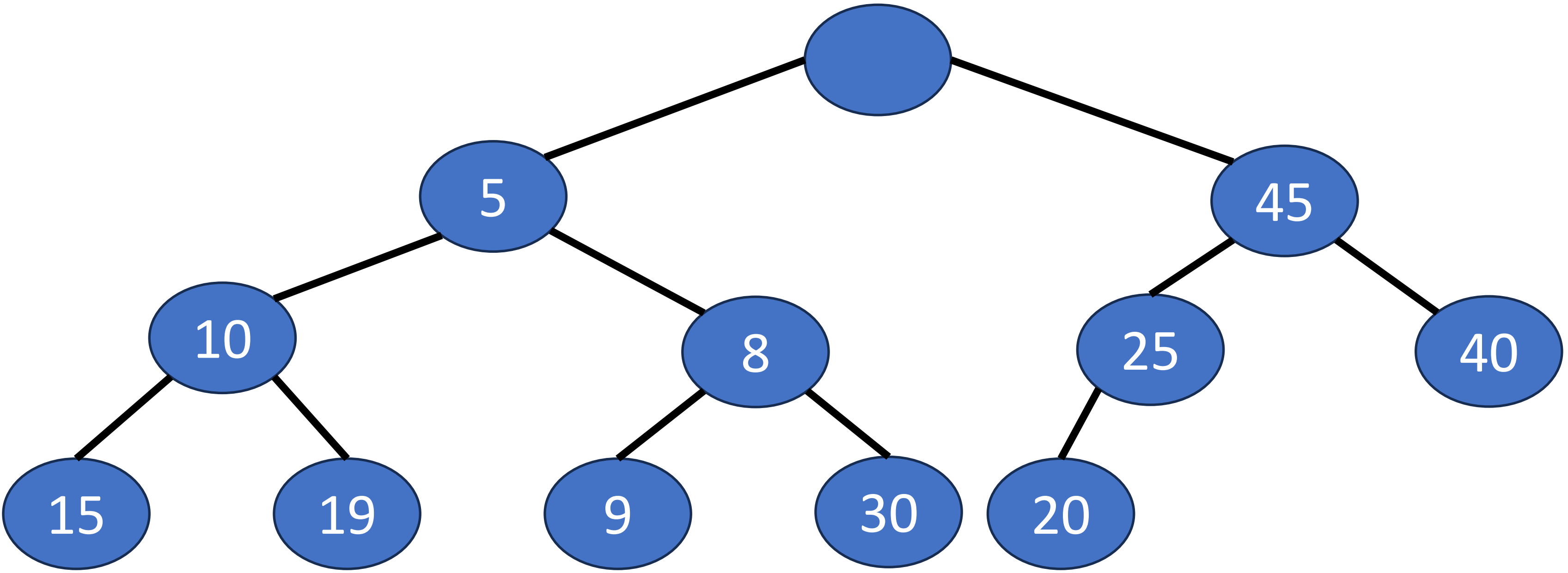
Please reply your answers of Q1 via the following link:

https://forms.gle/aWUwuR7JjTtyMCMv7

Group members: 2~4 people

Exercise

- Q2: Insert 30 into the following deap. Where will be the location of 30?



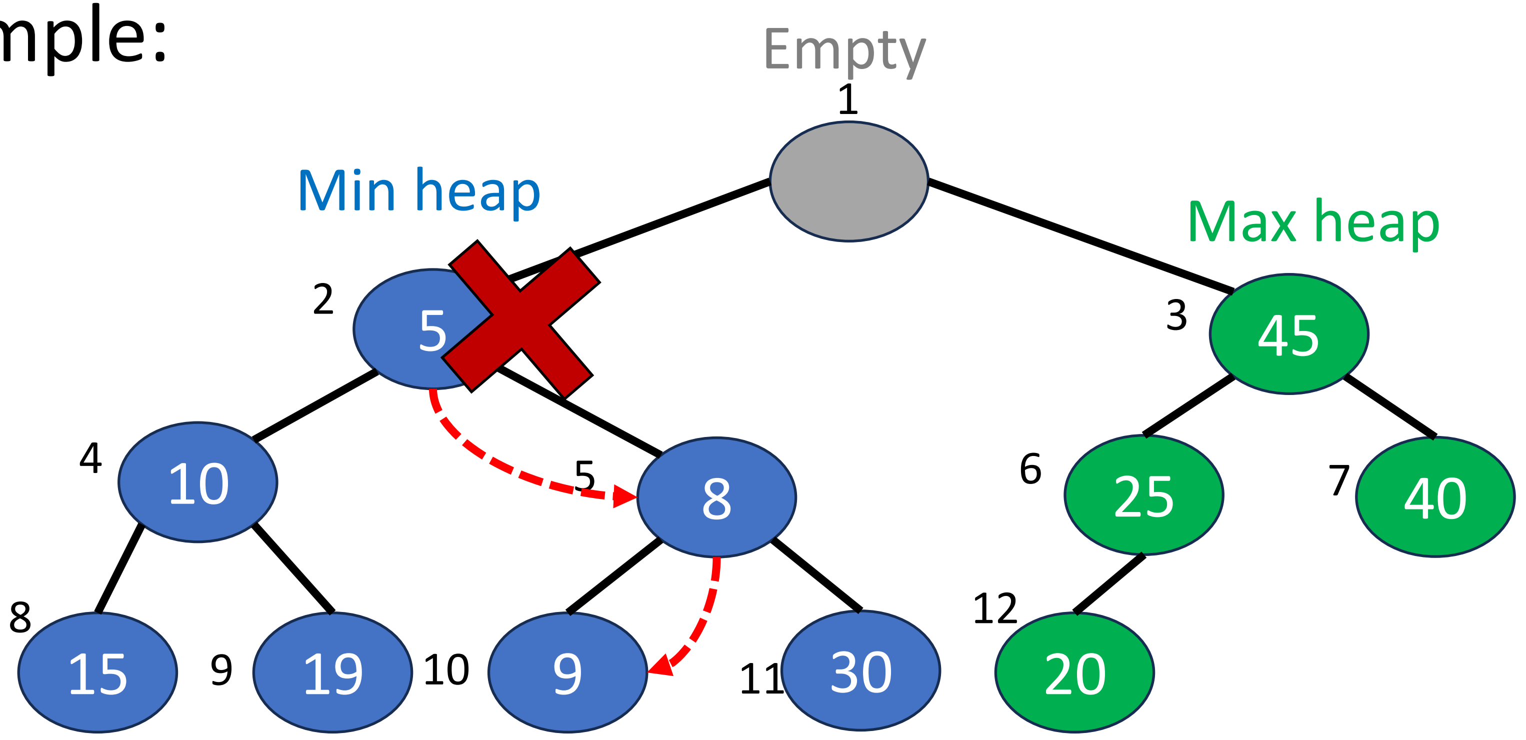
Please reply your answers of Q2 via the following link:

https://forms.gle/aWUwuR7JjTtyMCMv7

Group members: 2~4 people

Operation: DeleteMin

- Example:



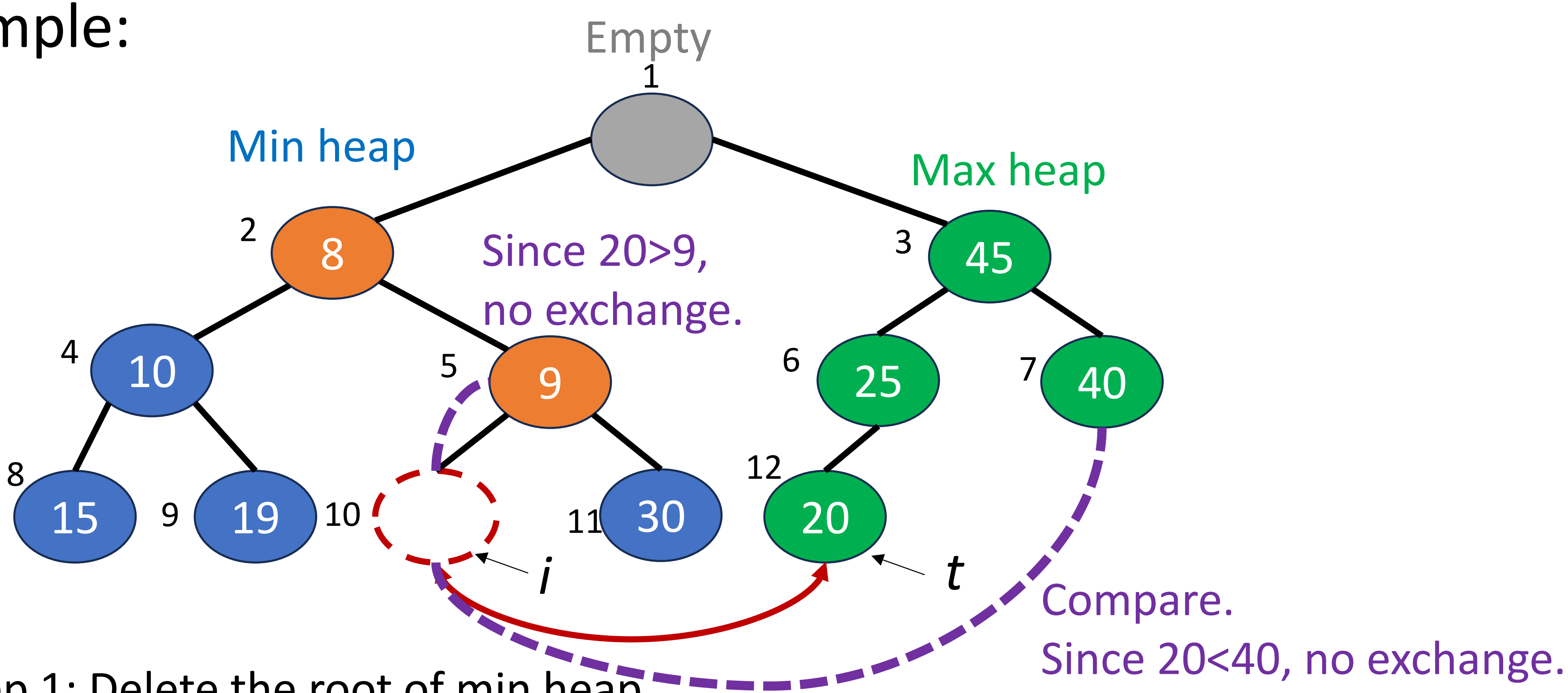
Step 1: Delete the root of min heap.

Step 2: Shifting the empty node from root to leaf position *i*.

Interchange with the child with min key value.

Operation: DeleteMin

- Example:



Step 1: Delete the root of min heap.

Step 2: Shifting the empty node from root to leaf position *i*.

Step 3: Insert the last element *t* into *i* using deap insertion.

Compare with the corresponding node and then do min (max) heap insertion.

DeleteMin algorithm

- Depend on the height of a heap.
- Time complexity: O(log n)

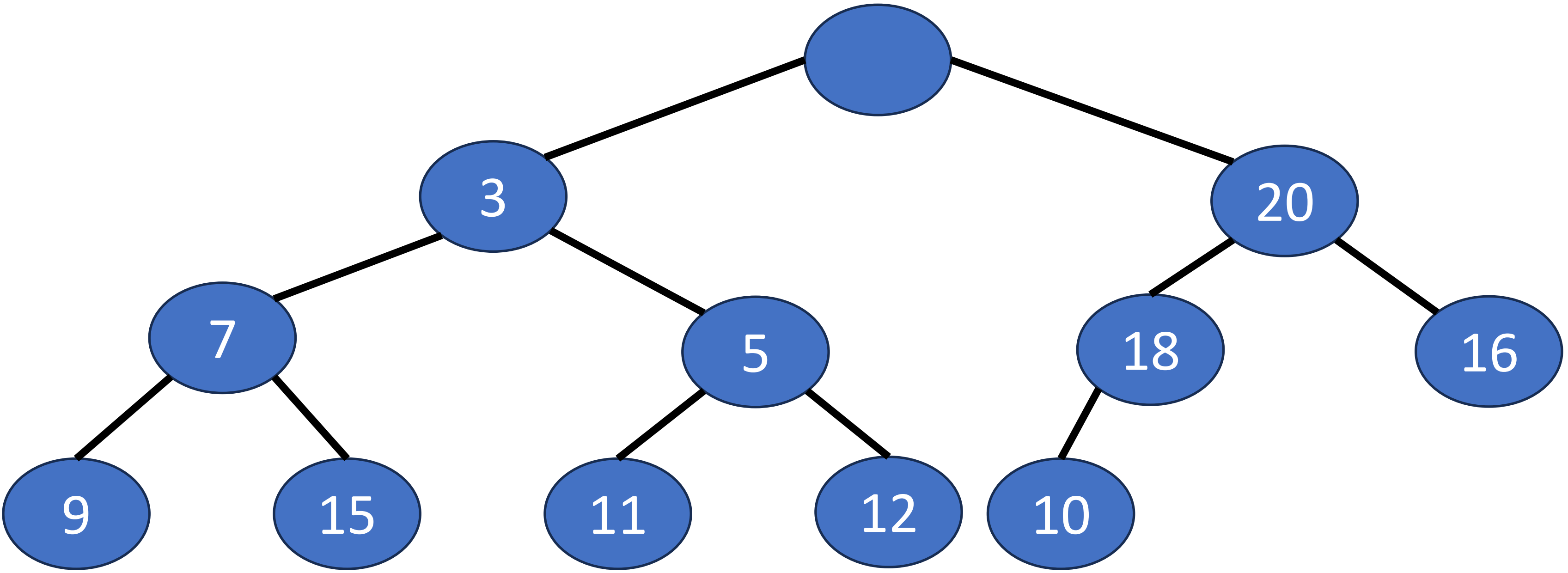
```
procedure DeapDeleteMin (var d : deap ; var n : integer ; var x : element);
{Delete the min element from the deap d.  The deleted element is returned in x.}
var i : integer;
    t : element;
begin
  if n < 2 then DeapEmpty
  else begin
        x := d[2];                Min element x
        t := d[n]; n := n−1;
                                  The last element
        i := 2;
        while i has a child do
        begin
          Let j be the child with smaller key;
          d[i] := d[j];
          i := j;
        end;
        Do a deap insertion of t at position i;
     end; {of if n < 2}
end; {of DeapDeleteMin}
```

Move empty node in the root (i=2) downward to a leaf

Insert the last element to the empty node.

Exercise

- Q3: Delete the min element from the following deap. Where will be the location of 10?



Please reply your answers of Q3 via the following link:

https://forms.gle/aWUwuR7JjTtyMCMv7

Group members: 2~4 people