Estructuras de Datos

Heap Data Structure

In This Session

- **Priority Queue Data Structure**
  - Max Priority Queues
    - Heaps
    - An Application

Priority Queue Data Structure

- **Priority Queues:** FIFO structure where elements are deleted in increasing (decreasing) order of priority rather than in the order in which they arrived in the queue.

- **Max Priority Queues**: The Find/Delete operations apply to the element of maximum priority.

- **Min Priority Queues**: The Find/Delete operations apply to the element of minimum priority.

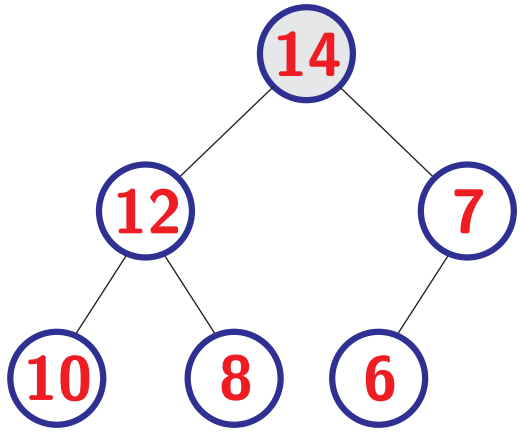
Representation of a Max Priority Queue

- Linear list.
- Heap.

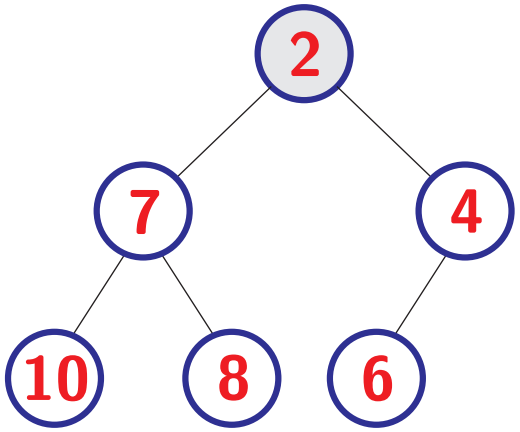
Heaps

- **Max tree (min tree):** is a tree in which the value in a each node is greater (less) than or equal to those in its children.

- **Max heap (min heap):** is a max (min) tree that is also a complete binary tree.



(a) MaxHeap



(b) MinHeap

Representation of a Heap

- Formula-based, since it is a complete Binary tree.
- Use of property P5.
- A heap with n elements has height $\lceil \log_2(n+1) \rceil$

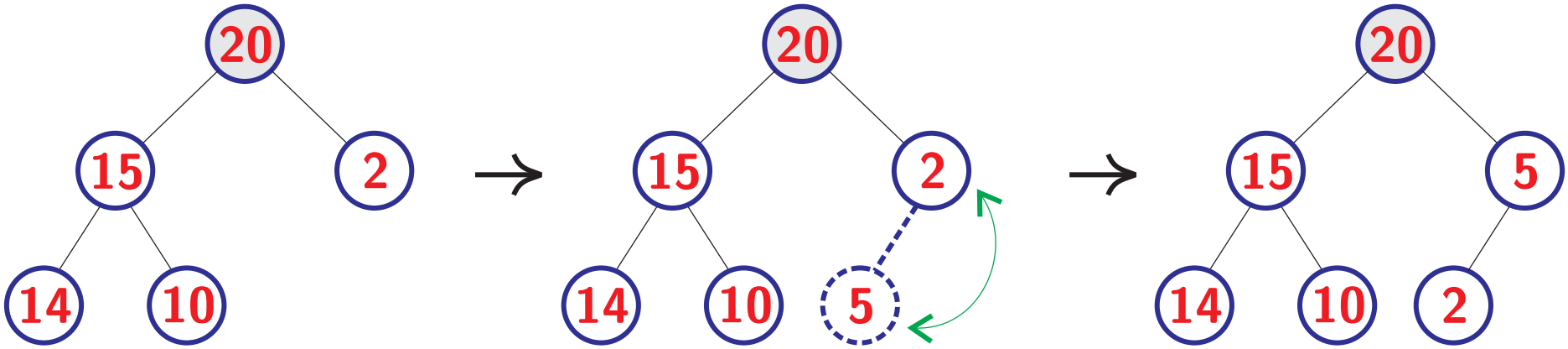
Insertion

- Insert a new element as a leaf of the heap.

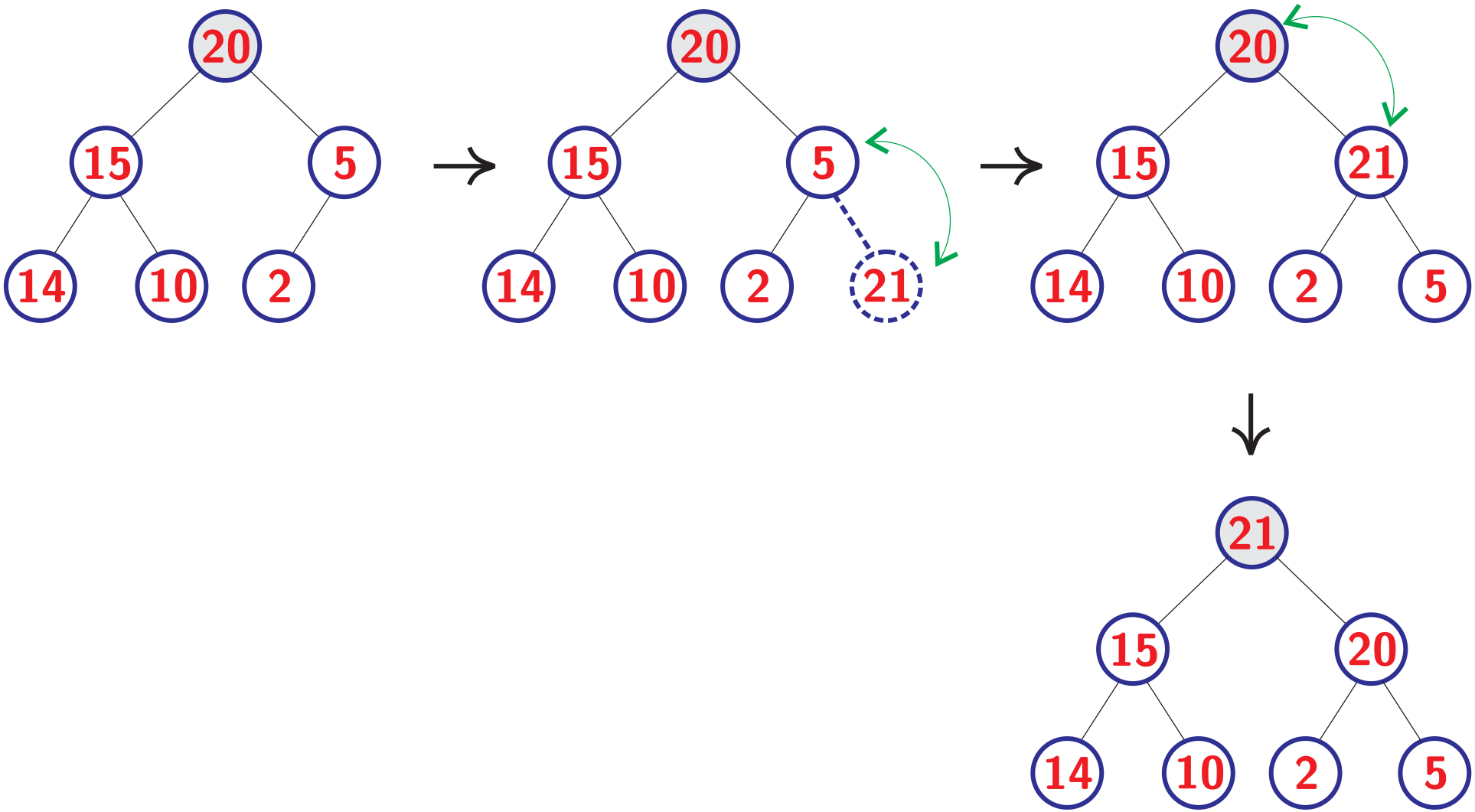
- Walk up to the root to restore the heap properties.

Time Complexity: $O(\log n)$

Example: Insert 5.



Example: Insert 21.



Deletion

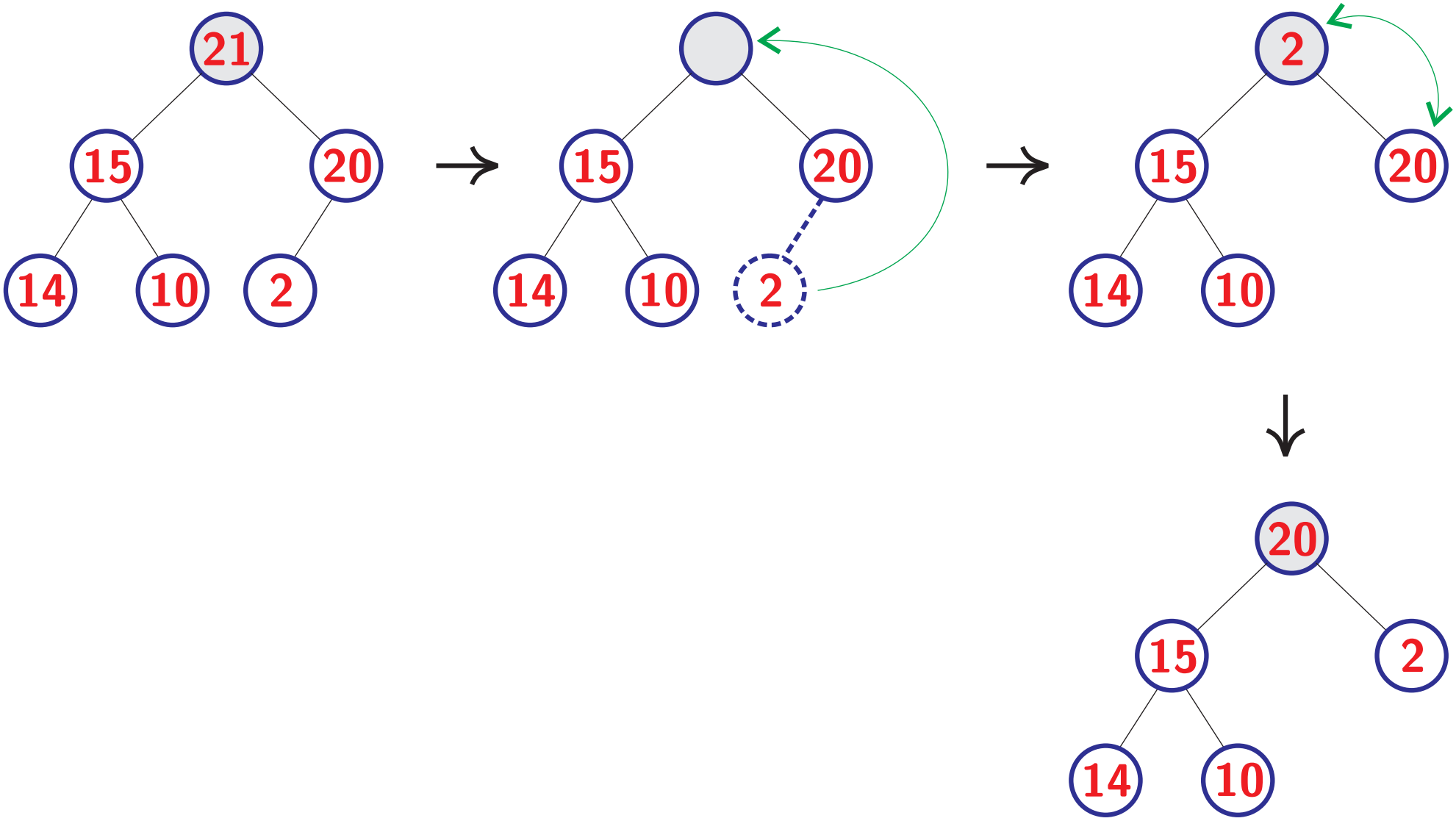
- Delete the root element.

- Delete the rightmost leaf at the highest level and put it in the root.

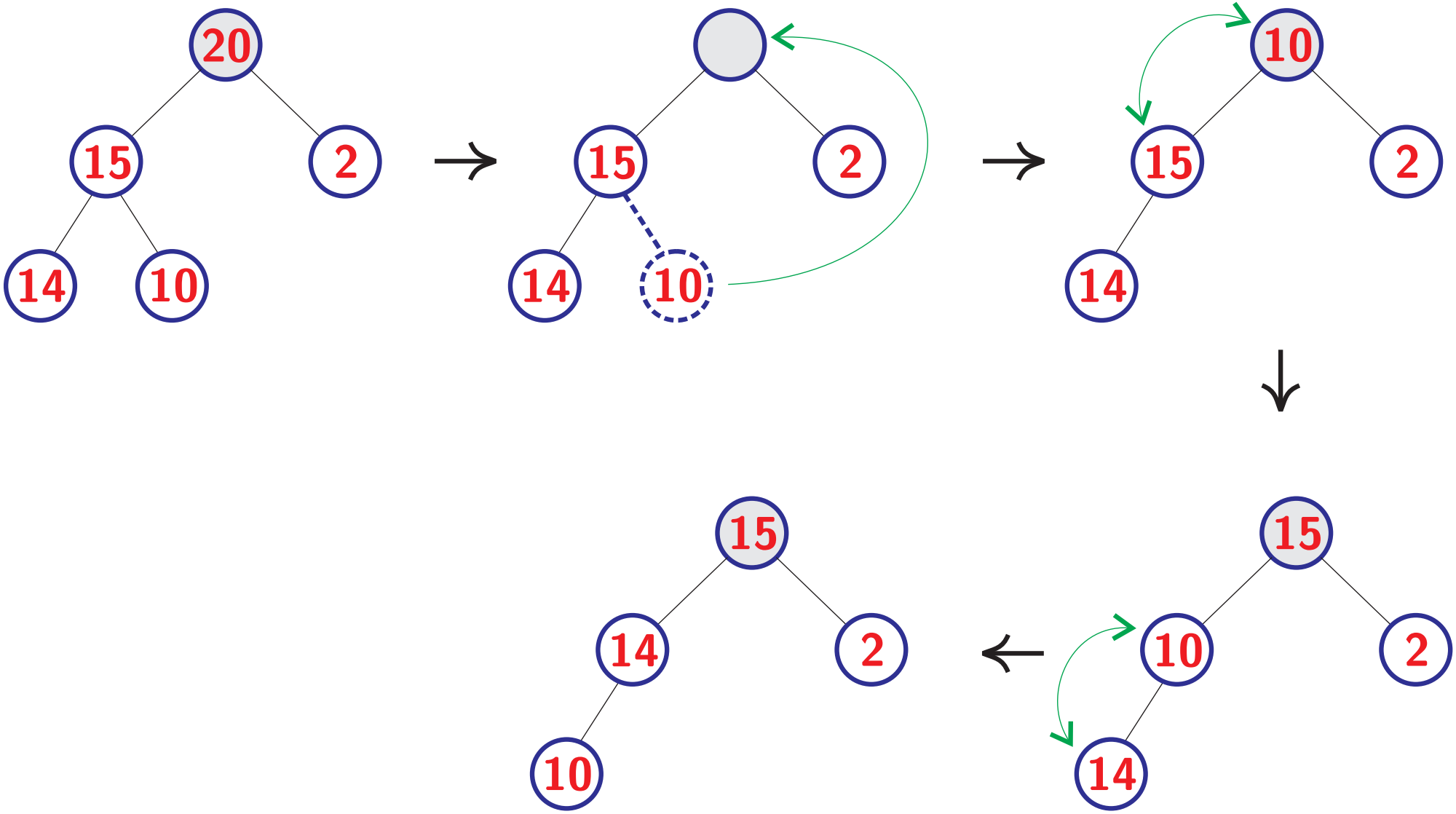
- Restore the heap properties by walking down from root to a leaf by following the path determined by the child having the largest value.

Time Complexity: $O(\log n)$

Example: Delete 21.



Example: Delete 20.



Initializing a Max Head

- n insert operations. Time $O(n \log n)$.

- Playing a tournament. Time $O(n)$.

maxheap.h

```
#ifndef MAXHEAP
#define MAXHEAP

#include <stdio.h>
#include <stdlib.h>
#define CAP 100

typedef struct Heap Heap;

struct Heap{
   int * a;
   int n;
   void (*put) ( Heap *, int );
   void (*delete) ( Heap * );
   int (*top) ( Heap * );
   int (*initialize) ( Heap *, int *, int );
   void (*display) ( Heap * );
   int (*empty) ( Heap * );
};

void put( Heap * x, int e ){ ... }
void delete( Heap * x ){ ... }
int top( Heap * x ){ ... }
int initialize( Heap * x, int * b, int size ){ ... }
void display( Heap * x ){ ... }
int empty( Heap * x ){ ... }
Heap createHeap( ){ ... }

#endif
```

createHeap

```
Heap createHeap( ){
   Heap h;
   h.n = 0;
   h.a = malloc( sizeof( int ) * CAP + 1 );
   h.put = &put;
   h.delete = &delete;
   h.top = &top;
   h.initialize = &initialize;
   h.display = &display;
   h.empty = &empty;
   return h;
}
```

put

```
void put( Heap * x, int e ){
   if( x->n == CAP ){
      fprintf( stderr, "Error: Heap is full\n" );
      return;
   }
   int p = ++x->n;
   while( p != 1 && x->a[ p / 2 ] < e ){
      x->a[ p ] = x->a[ p / 2 ];
      p /= 2;
   }
   x->a[ p ] = e;
}
```

delete

```
void delete( Heap * x ){
   if( empty( x ) ){
      fprintf( stderr, "Error: Heap is empty\n" );
      return;
   }
   int e = x->a[ x->n-- ];
   int p = 1, child = 2;
   while( child <= x->n ){
      if( child < x->n && x->a[ child ] < x->a[ child + 1 ] )
         child++;
      if( e >= x->a[ child ] )
         break;
      x->a[ p ] = x->a[ child ];
      p = child;
      child *= 2;
   }
   x->a[ p ] = e;
}
```

top, display, empty

```
int top( Heap * x ){
   if( empty( x ) ){
      fprintf( stderr, "Error: Heap is empty\n" );
      exit( 1 );
   }
   return x->a[ 1 ];
}


void display( Heap * x ){
   int i;
   printf( "Heap: " );
   for( i = 1; i <= x->n; i++ )
      printf( "%d ", x->a[ i ] );
   printf( "\n" );
   return;
}


int empty( Heap * x ){
   return !x->n;
}
```

initialize

```
int initialize( Heap * x, int * b, int size ){
   int root, rootElement, child;
   free( x->a );
   x->a = b;
   x->n = size;
   for( root = x->n / 2; root >= 1; root-- ){
      rootElement = x->a[ root ];
      child = 2 * root;
      while( child <= x->n ){
         if( child < x->n && x->a[ child ] < x->a[ child + 1 ] )
            child++;
         if( rootElement >= x->a[ child ] )
            break;
         x->a[ child / 2 ] = x->a[ child ];
         child *= 2;
      }
      x->a[ child / 2 ] = rootElement;
   }
}
```

testheap.c

```
#include <stdio.h>
#include "maxheap.h"

int main(){
   int array[] = { 0, 8, 23, 89, 100, 35, 11, 52, 77, 44, 60 };
   Heap maria = createHeap( );
   maria.put( &maria, 100 );
   maria.put( &maria, 80 );
   maria.put( &maria, 15 );
   maria.put( &maria, 24 );
   maria.put( &maria, 135 );
   maria.display( &maria );
   printf( "Top element is %d\n", maria.top( &maria ) );
   maria.delete( &maria );
   maria.display( &maria );
   maria.initialize( &maria, array, 10 );
   maria.display( &maria );
   while( !maria.empty( &maria ) ){
      printf( "Deleting max element %d\n", maria.top( &maria ) );
      maria.delete( &maria );
   }
   return 0;
}
```

Compilation and Running

```
C:\Users\Yoan\Desktop\code\progs>gcc testheap.c

C:\Users\Yoan\Desktop\code\progs>a
Heap: 135 100 15 24 80
Top element is 135
Heap: 100 80 15 24
Heap: 100 77 89 44 60 11 52 23 8 35
Deleting max element 100
Deleting max element 89
Deleting max element 77
Deleting max element 60
Deleting max element 52
Deleting max element 44
Deleting max element 35
Deleting max element 23
Deleting max element 11
Deleting max element 8

C:\Users\Yoan\Desktop\code\progs>
```

Priority Queue Application

– Heap Sort –

A heap can be used to sort n elements in $O(n \log n)$ time.

1) Initialize a max heap with n elements (time $O(n)$)

2) Extract (i.e. delete) elements from the heap one at a time. Each deletion takes $O(\log n)$ time, so the total time is $O(n \log n)$