CSE100: Design and Analysis of Algorithms
Lecture 09 – Maximum Subarray & Matrix Multiplication (wrap up), Heaps

Feb 15th 2022

More divide and conquer, Strassen's algorithm, Heaps, Heapsort and Priority Queues

Matrix multiplication

- How to multiply two matrices?

$$\begin{bmatrix} -3 & 3 \\ 3 & -2 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 \\ -2 & -1 \end{bmatrix} = \begin{bmatrix} -9 & -3 \\ 7 & 2 \\ 2 & 1 \end{bmatrix}$$

- Given matrix A_{nn} and $B_{nn}, C_{nn} = AB$
- $c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$

Time Complexity?

- For each c_{ij}, we need $\Theta(n)$
- There are $n^2 c_{ij}$, so $T(n) = n^2 \Theta(n)$=$\Theta(n^3)$

Matrix multiplication divide-and-conquer algorithm

- $C = A \times B$
- $\begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \times \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$
- $C_{11} = A_{11} \times B_{11} + A_{12} \times B_{21}$
- $C_{12} = A_{11} \times B_{12} + A_{12} \times B_{22}$
- $C_{21} = A_{21} \times B_{11} + A_{22} \times B_{21}$
- $C_{22} = A_{21} \times B_{12} + A_{22} \times B_{22}$
- Recurrence equation?
- $T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$

Matrix multiplication divide-and-conquer algorithm

- $T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$
- What is the time complexity?
- From Master method we know it is $\Theta(n^3)$

Matrix multiplication Strassen's algorithm

- $T(n) = 8T\left(\frac{n}{2}\right) + \Theta(n^2)$

- $T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$

Matrix multiplication
Strassen's algorithm

- Strassen's algorithm:

1. Perform 10 times matrix addition or subtraction to make S_1 to S_{10} from A_{ij} and B_{ij}
2. Perform 7 times **matrix multiplication** to make P_1 to P_7 from A_{ij} , B_{ij} and S_i
3. Perform matrix addition or matrix subtraction to obtain C_{11}, C_{12}, C_{21} and C_{22}

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2) = \Theta(n^{log_2 7})$$

Strassen's algorithm (1)

- Discovered a way to compute the C_{ij}'s using 7 multiplications and 18 additions or subtractions
- It is a bit complicated!
- Some notation first (the essence of the algo):

$$P = (A_{11} + A_{22})(B_{11} + B_{22})$$
$$Q = (A_{21} + A_{22})B_{11}$$
$$R = A_{11}(B_{12} - B_{22})$$
$$S = A_{22}(B_{21} - B_{11})$$
$$T = (A_{11} + A_{12})B_{22}$$
$$U = (A_{21} - A_{11})(B_{11} + B_{12})$$

$$V = (A_{12} - A_{22})(B_{21} + B_{22})$$
$$C_{11} = P + S - T + V$$
$$C_{12} = R + T$$
$$C_{21} = Q + S$$
$$C_{22} = P + R - Q + U$$

Strassen's algorithm (2)

procedure Strassen (n, A, B, C) // n is size, A,B the input matrices, C output matrix

begin

if $n = 2$, ← Stopping Condition In the recursion

$$\begin{cases} C_{11} = a_{11} \cdot b_{11} + a_{12} \cdot b_{21}; \\ C_{12} = a_{11} \cdot b_{12} + a_{12} \cdot b_{22}; \\ C_{21} = a_{21} \cdot b_{11} + a_{22} \cdot b_{21}; \\ C_{22} = a_{21} \cdot b_{12} + a_{22} \cdot b_{22}; \end{cases}$$

else

(cont.)

Strassen's algorithm (3)

else

Partition A into 4 submatrices: $A_{11}, A_{12}, A_{21}, A_{22}$;

Partition B into 4 submatrices: $B_{11}, B_{12}, B_{21}, B_{22}$;

call Strassen ($\frac{n}{2}, A_{11} + A_{22}, B_{11} + B_{22}, P$);

call Strassen ($\frac{n}{2}, A_{21} + A_{22}, B_{11}, Q$);

call Strassen ($\frac{n}{2}, A_{11}, B_{12} - B_{22}, R$);

call Strassen ($\frac{n}{2}, A_{22}, B_{21} - B_{11}, S$);

call Strassen ($\frac{n}{2}, A_{11} + A_{12}, B_{22}, T$);

call Strassen ($\frac{n}{2}, A_{21} - A_{11}, B_{11} + B_{12}, U$);

call Strassen ($\frac{n}{2}, A_{12} - A_{22}, B_{21} + B_{22}, V$);

Strassen's algorithm (4)

(cont)

$$C_{11} = P + S - T + V;$$
$$C_{12} = R + T;$$
$$C_{21} = Q + S;$$
$$C_{22} = P + R - Q + U;$$

end;

Ufff… that was long!

Time Complexity

$$T(n) = 7T\left(\frac{n}{2}\right) + \Theta(n^2)$$

- Remember the Master Theorem
- Suppose $T(n) = a \cdot T\left(\frac{n}{b}\right) + O\left(n^d\right)$. Then

$$T(n) = \begin{cases} O(n^d \log(n)) & \text{if } a = b^d \\ O(n^d) & \text{if } a < b^d \\ O(n^{\log_b(a)}) & \text{if } a > b^d \end{cases}$$

- Our recurrence formula has the appropriate format!

  a : number of subproblems

  b : factor by which input size shrinks

  d : need to do n^d work to create all the subproblems and combine their solutions.

  - a=7, b=2, d=2 (bottom case)
- $T(n) = O(n^{\log\_2(7)}) = O(n^{2.81})$

Discussion of Strassen's Algorithm

- Not always practical
  - constant factor is larger than for naïve method
  - specially designed methods are better on sparse matrices
  - issues of numerical (in)stability
  - recursion uses lots of space

- Not the fastest known method
  - Fastest known is $O(n^{2.376})$
  - Best known lower bound is $\Omega(n^2)$

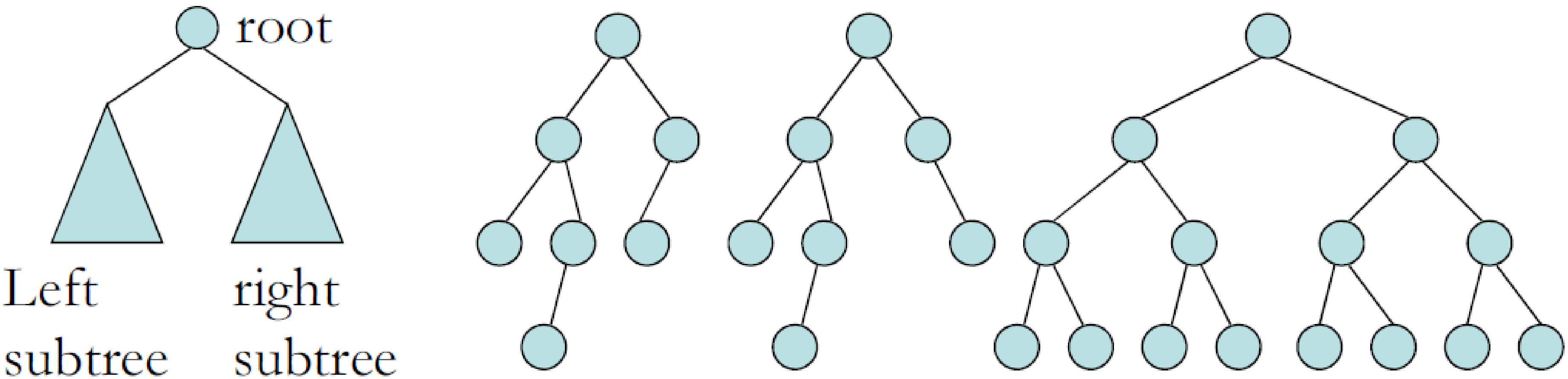
Recap (last and this lecture)

- Two more examples of divide and conquer strategies
- We saw a (pretty clever) algorithm to do find the maximum subarray in time O(n.logn)
  - Not the fastest (Kadane O(n))
- We also saw a (complicated) algorithm to perform matrix multiplication in time $O(n^{2.81})$
  - Not the fastest (best known is $O(n^{2.376})$)
- We'll now see some more sorting algorithms (Heapsort)

Trees, heaps, heapsort, priority queues.

- Basic tree and heap properties
- Heap operations:
  - Heapify
  - Build-Heap
  - Heapsort
- Running time of all the operations
- Priority Queues
- PQ operations:
  - Insert
  - Maximum
  - ExtractMax

Binary Tree



- A node without subtree is called a leaf
- In a full binary tree, each node has 2 or NO children
- A *complete binary tree* has all leaves with the same depth and all internal nodes have 2 children

Heap Data Structure

- Definition
  - (binary) heap data structure is an array object that we can view as a nearly complete binary tree
- A node of the tree corresponds to an element of the array A[1…n]
  - n: heap size
  - A[1]: root
  - floor[i/2]: parent of node i
  - 2i: left child of node i
  - 2i+1: right child of node i

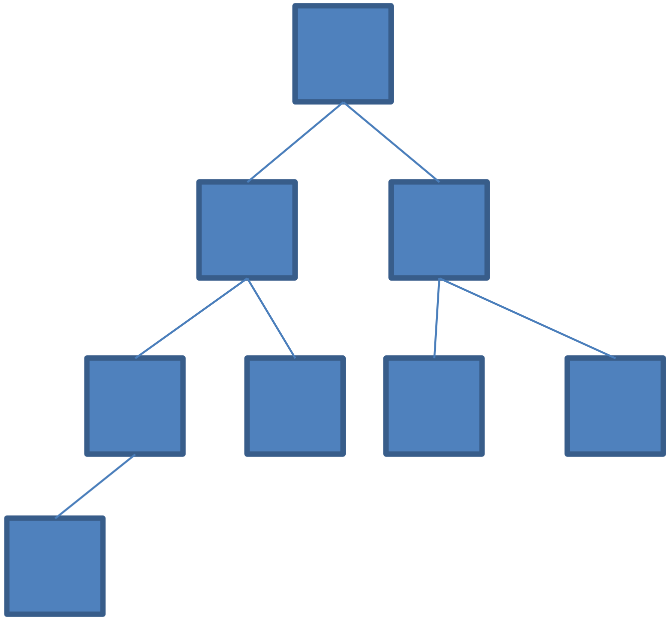
Properties of Heap

- Height of heap: Θ(logn)
  - Because the heap is a binary tree, the height of any node is at most Θ(logn)
- Max-Heap:
  - A[PARENT(*i*)] ≥ A[*i*]   for all nodes *i* except the root
  - Root stores the largest value
- Min-Heap:
  - A[PARENT(*i*)] ≤ A[*i*]   for all nodes *i* except the root
  - Root stores the smallest value
- From now on, we'll work with Max-Heaps, but the same rules apply for both

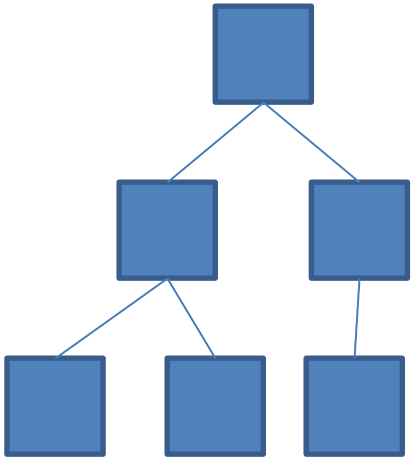
Max-Heap

- A nearly complete binary tree, and …

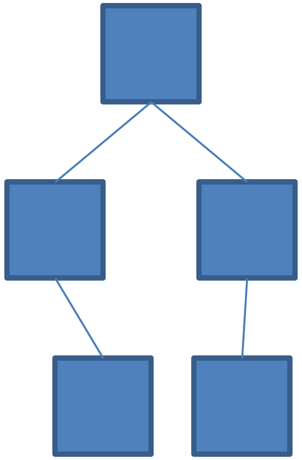
every level is completely filled, *except possibly the last*, which is filled from left to right



Yes



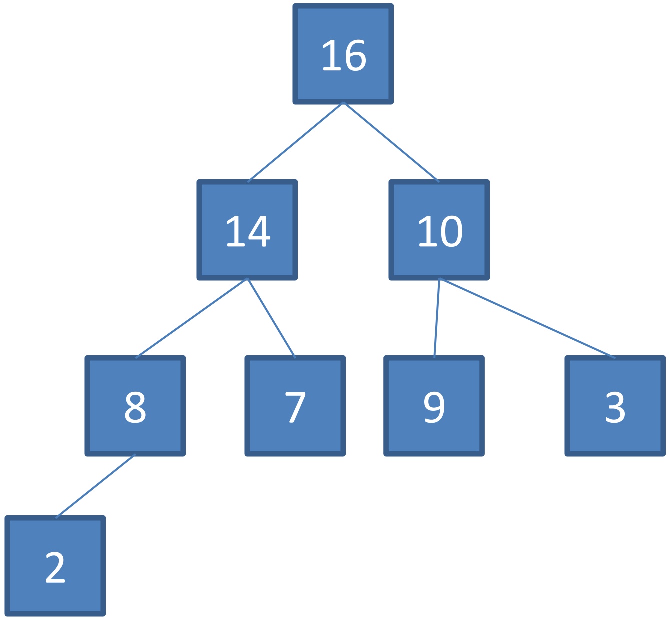
Yes



No

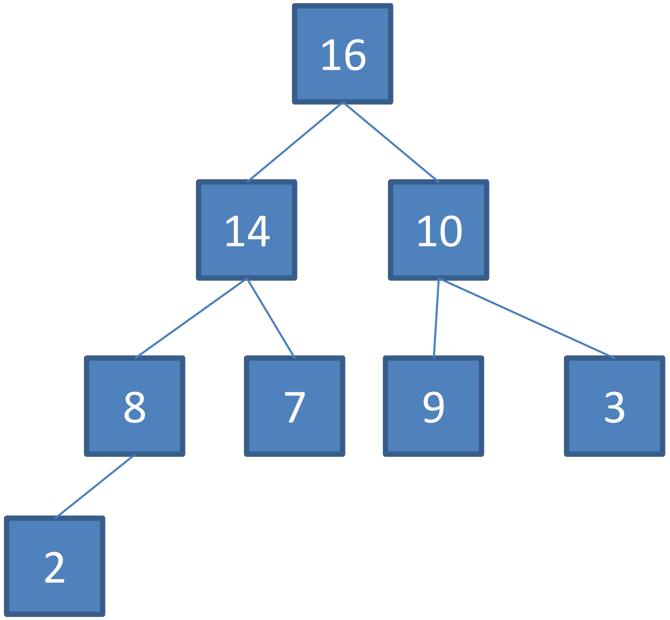
Max-Heap

- Satisfy max-heap property: parent >= children



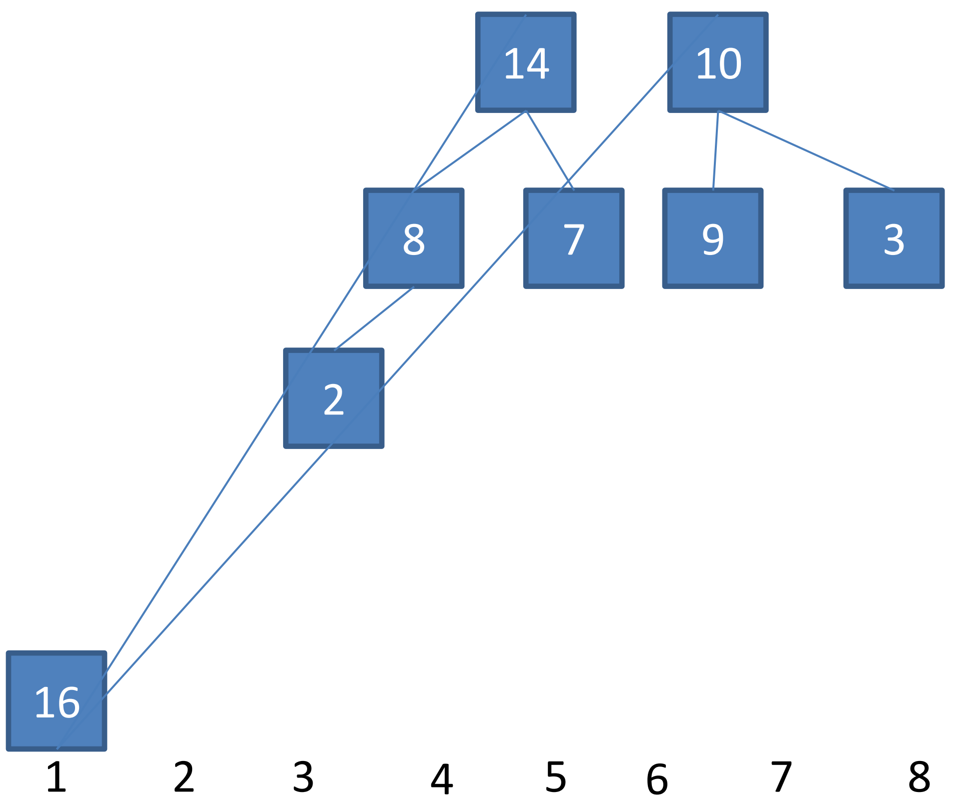
Since it is a complete tree, it **can be put into an array without lose its structure information**.

Max-Heap

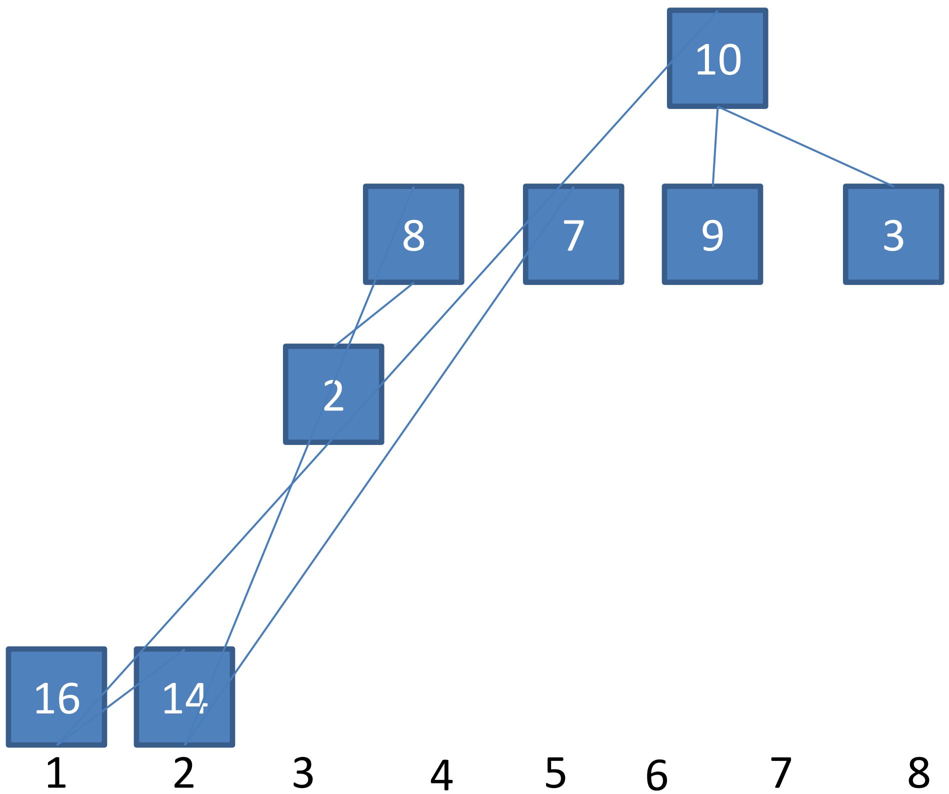


1 2 3 4 5 6 7 8

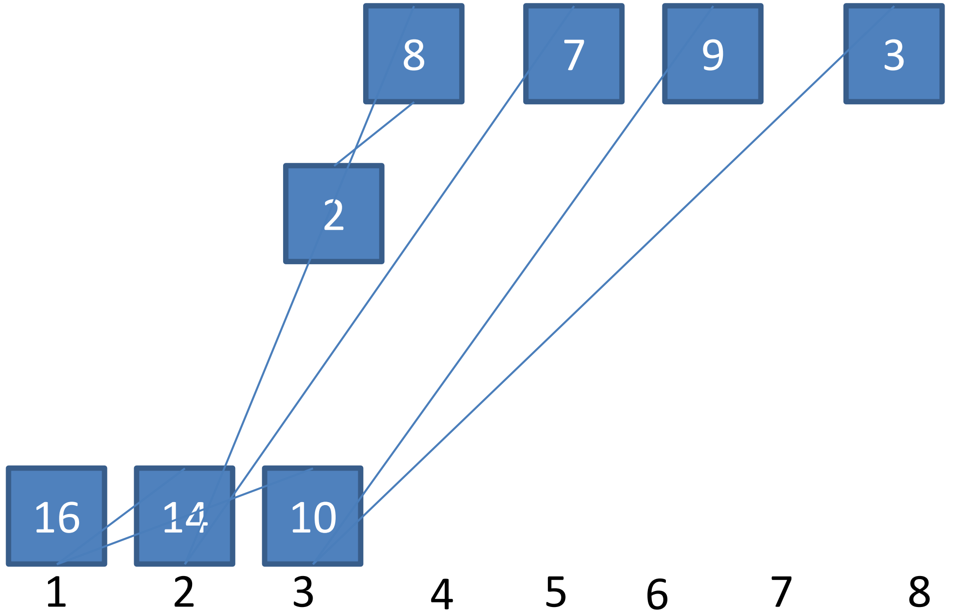
Max-Heap



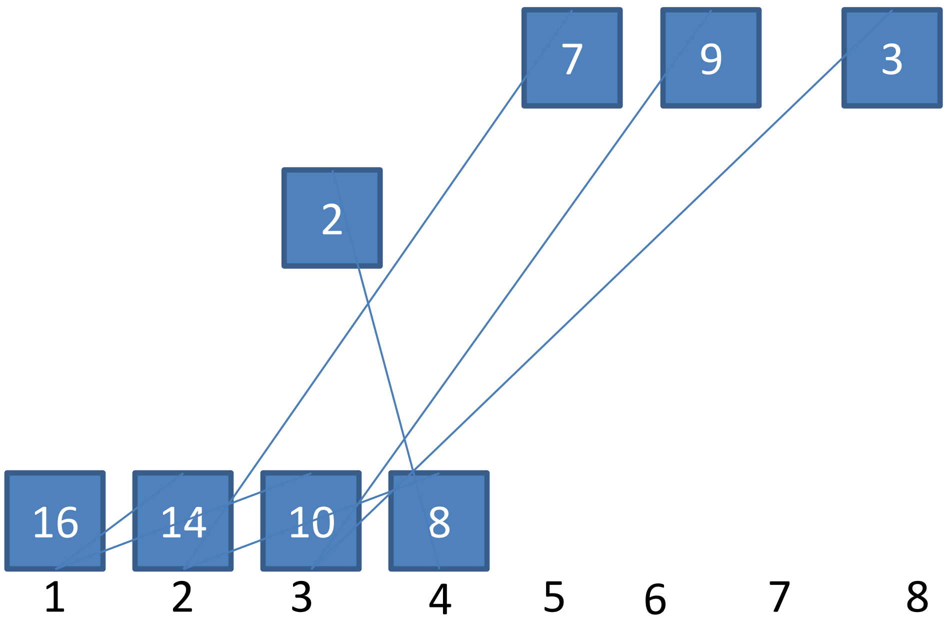
Max-Heap



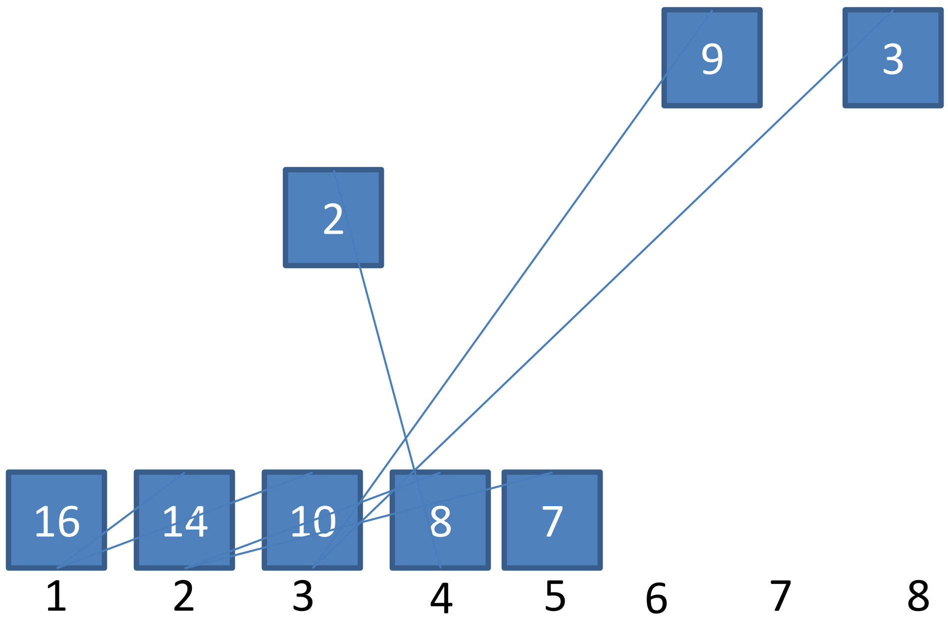
Max-Heap



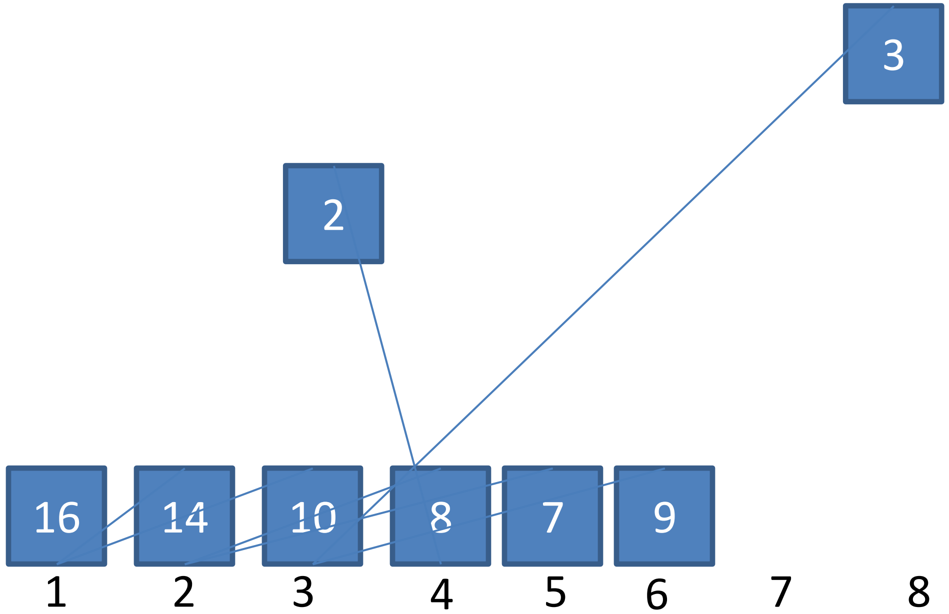
Max-Heap



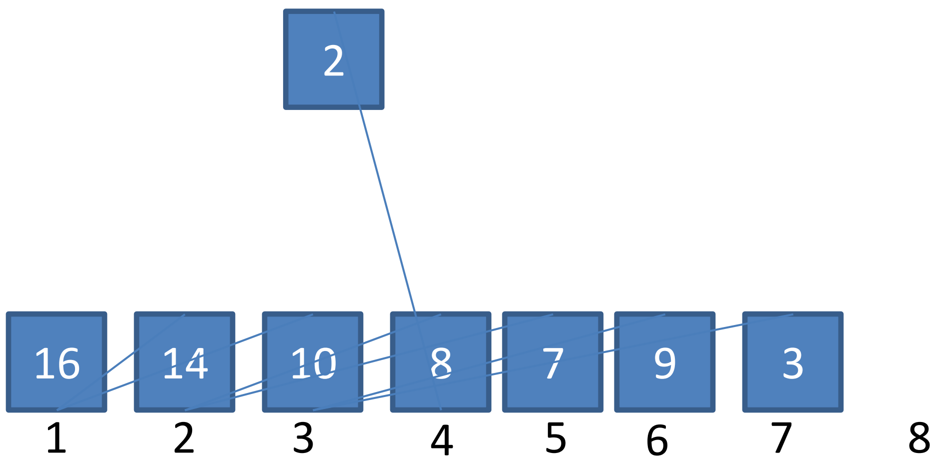
Max-Heap



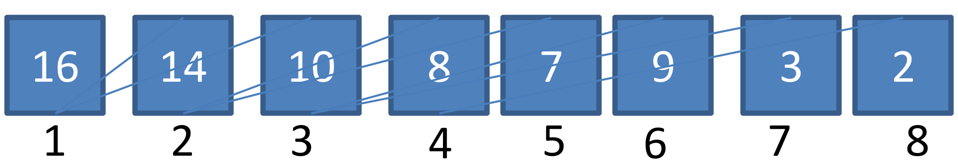
Max-Heap



Max-Heap

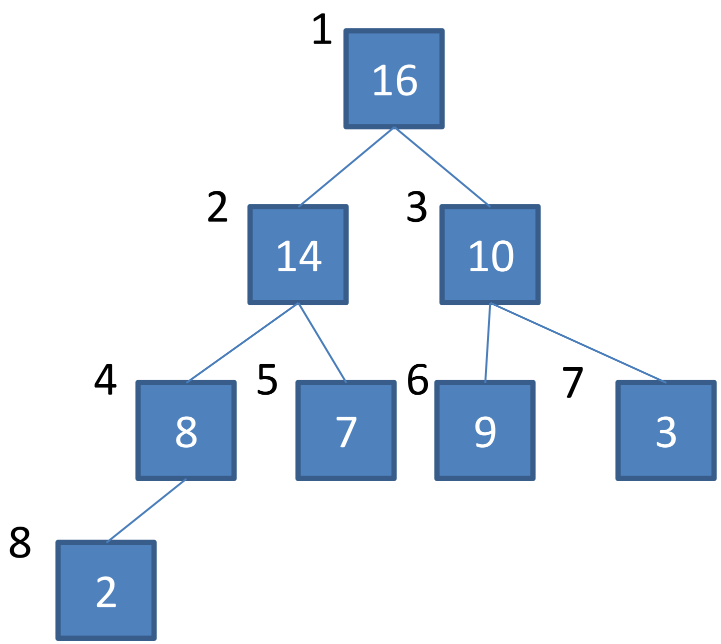


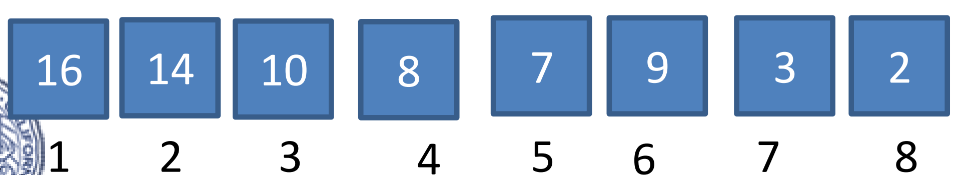
Max-Heap



Max-Heap

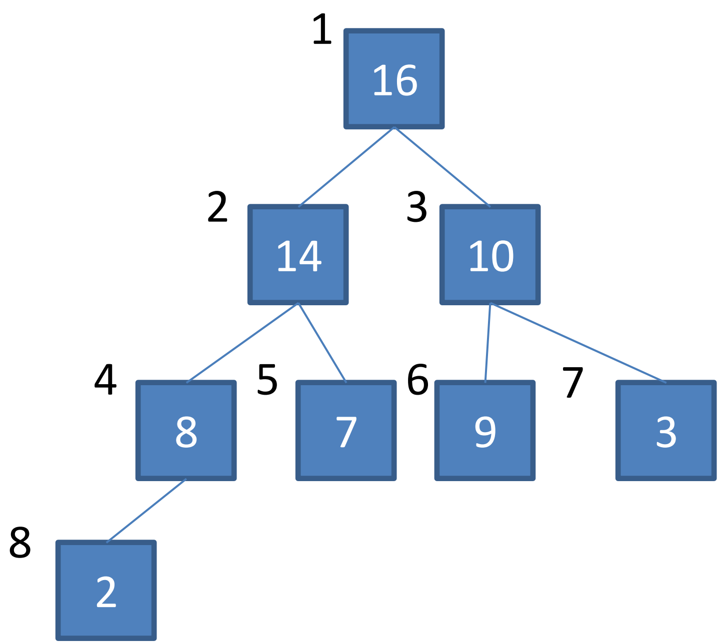
- Use an array as a heap





Max-Heap

- Use an array as a heap

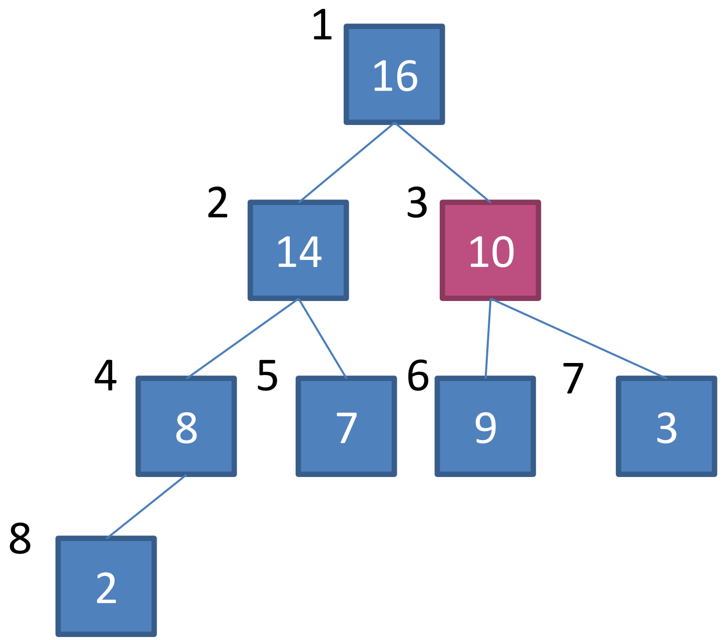


For element at i:
Parent index =parent(i)= floor(i/2);
Left child index = left(i)=2*i;
Right child index =right(i)=2*i +1
Last non-leaf node = floor(length/2)

16	14	10	8	7	9	3	2
1	2	3	4	5	6	7	8

Max-Heap

- Use an array as a heap



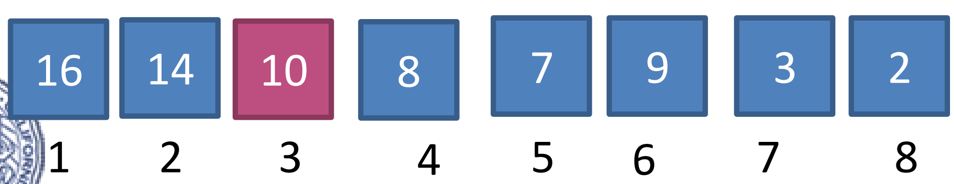
For element at i:

Parent index =parent(i)= floor(i/2);

Left child index = left(i)=2*i;

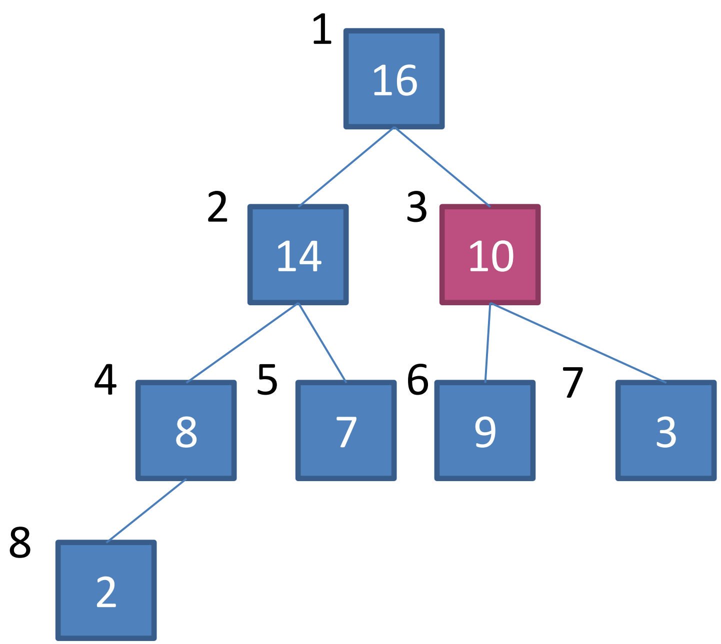
Right child index =right(i)=2*i +1

Last non-leaf node = floor(length/2)



Max-Heap

- Use an array as a heap



For element at *i*:

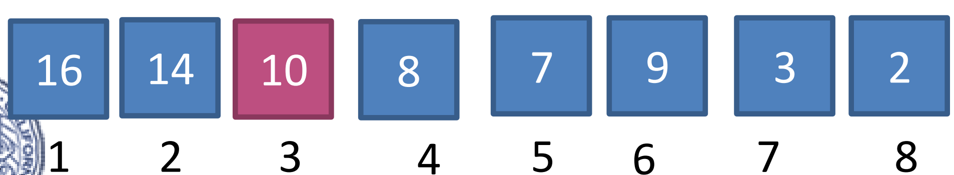
Parent index =parent(*i*)= floor(*i*/2);

Left child index = left(*i*)=2**i*;

Right child index =right(*i*)=2**i* +1

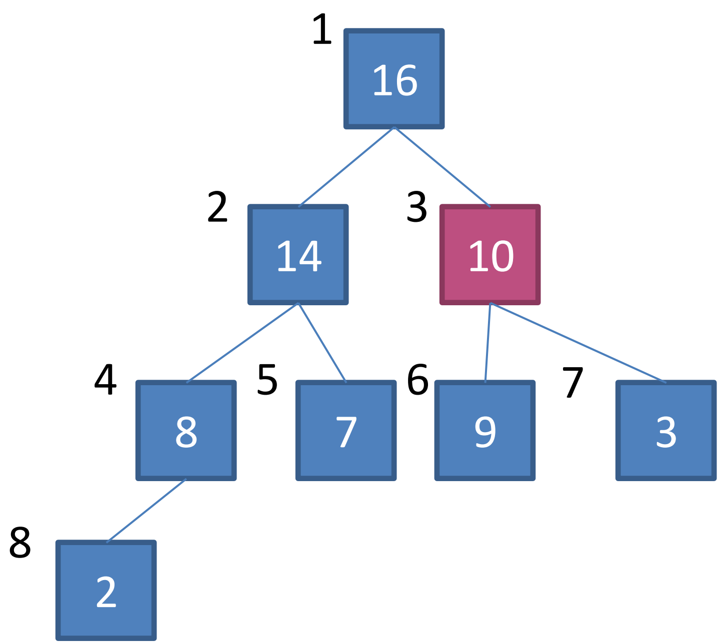
Last non-leaf node = floor(length/2)

i=3



Max-Heap

- Use an array as a heap



For element at *i*:

Parent index =parent(*i*)= floor(*i*/2);

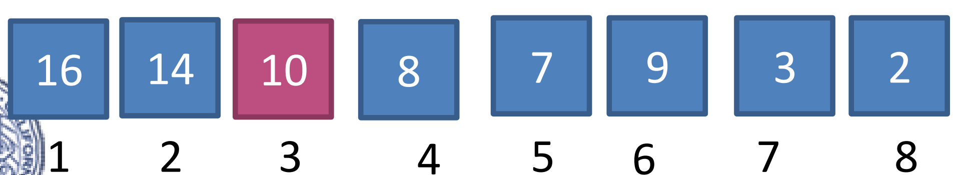
Left child index = left(*i*)=2**i*;

Right child index =right(*i*)=2**i* +1

Last non-leaf node = floor(length/2)

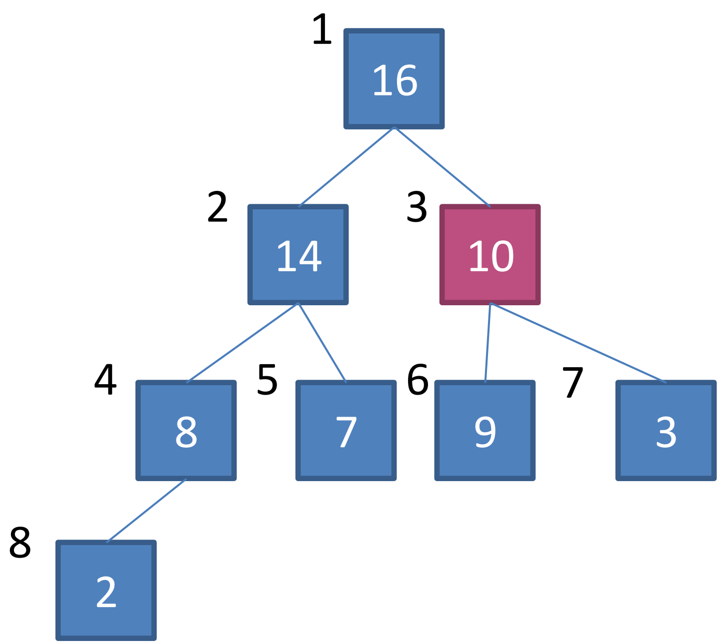
i=3

floor(*i*/2)=floor(1.5)=1



Max-Heap

- Use an array as a heap



For element at *i*:

Parent index =parent(*i*)= floor(*i*/2);

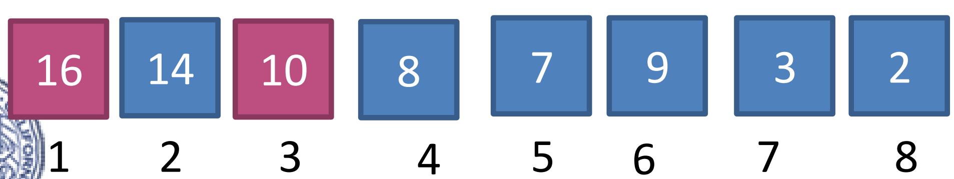
Left child index = left(*i*)=2**i*;

Right child index =right(*i*)=2**i* +1

Last non-leaf node = floor(length/2)

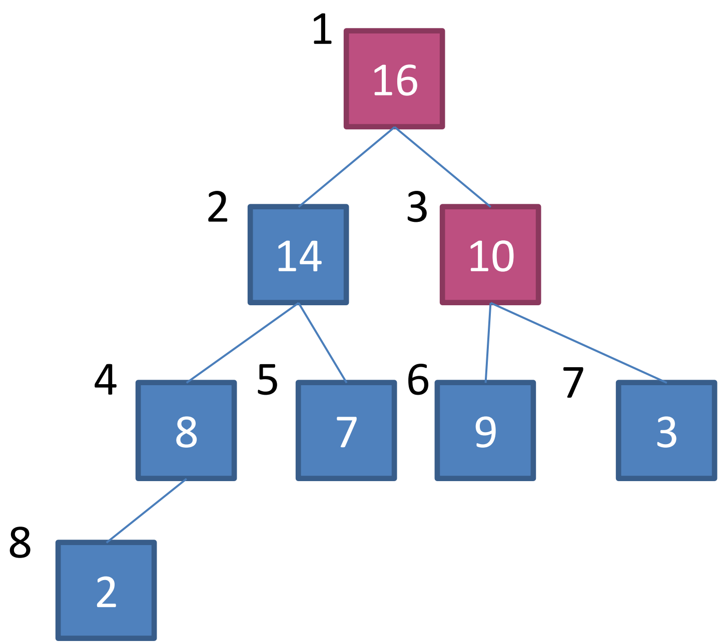
i=3

floor(*i*/2)=floor(1.5)=1



Max-Heap

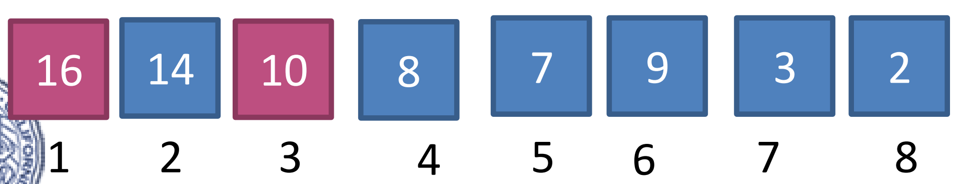
- Use an array as a heap



For element at i:
Parent index =parent(i)= floor(i/2);
Left child index = left(i)=2*i;
Right child index =right(i)=2*i +1
Last non-leaf node = floor(length/2)

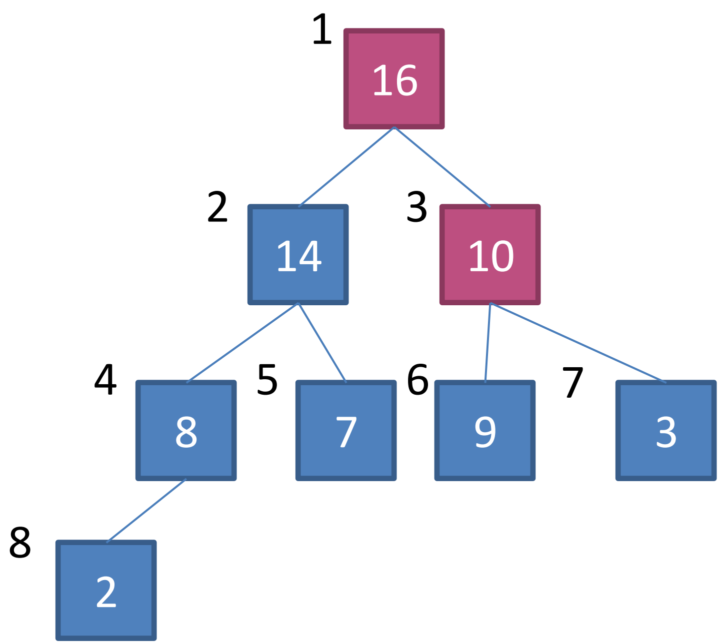
i=3

floor(i/2)=floor(1.5)=1



Max-Heap

- Use an array as a heap



For element at *i*:

Parent index =parent(*i*)= floor(*i*/2);

Left child index = left(*i*)=2**i*;

Right child index =right(*i*)=2**i* +1

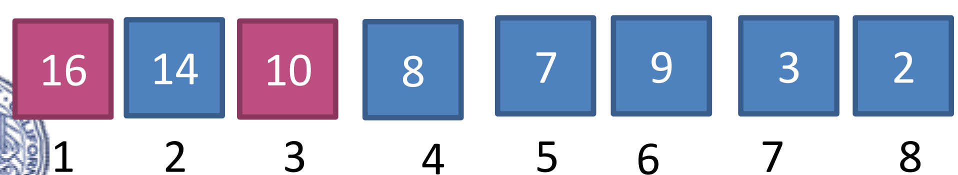
Last non-leaf node = floor(length/2)

i=3

floor(i/2)=floor(1.5)=1

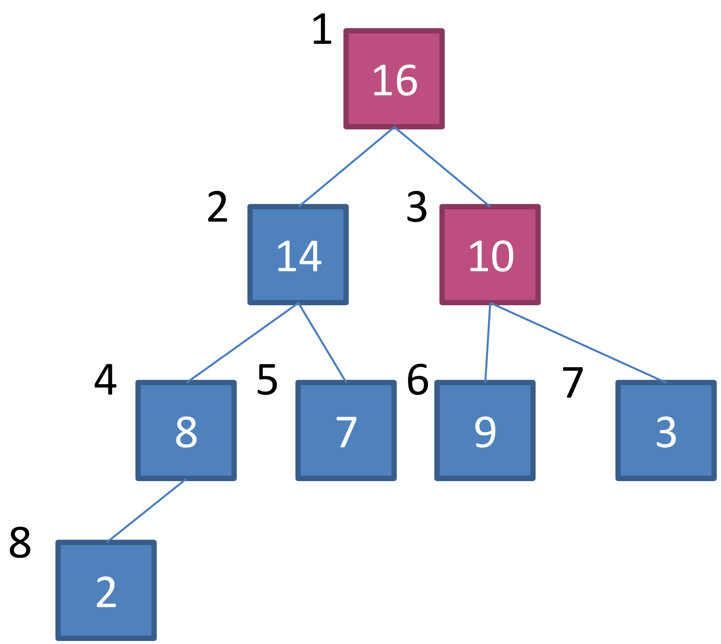
2*i = 6

2*i+1=7



Max-Heap

- Use an array as a heap



For element at i:

Parent index =parent(i)= floor(i/2);

Left child index = left(i)=2*i;

Right child index =right(i)=2*i +1

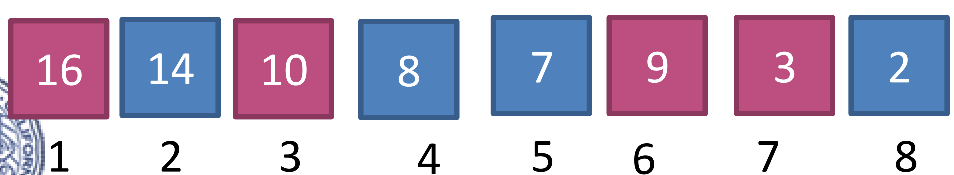
Last non-leaf node = floor(length/2)

i=3

floor(i/2)=floor(1.5)=1

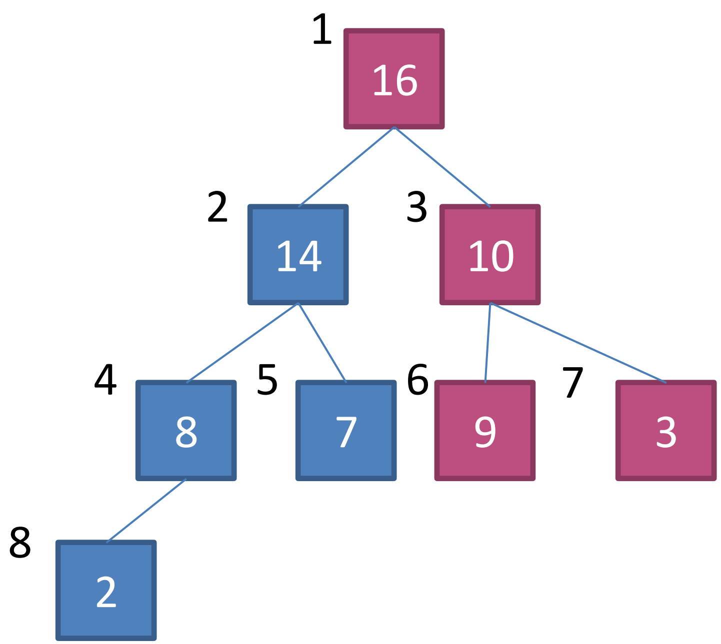
2*i = 6

2*i+1=7



Max-Heap

- Use an array as a heap



For element at i:

Parent index =parent(i)= floor(i/2);

Left child index = left(i)=2*i;

Right child index =right(i)=2*i +1

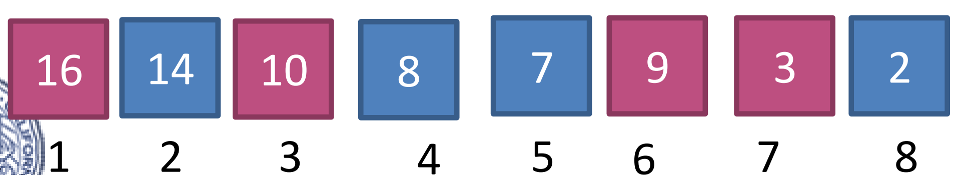
Last non-leaf node = floor(length/2)

i=3

floor(i/2)=floor(1.5)=1

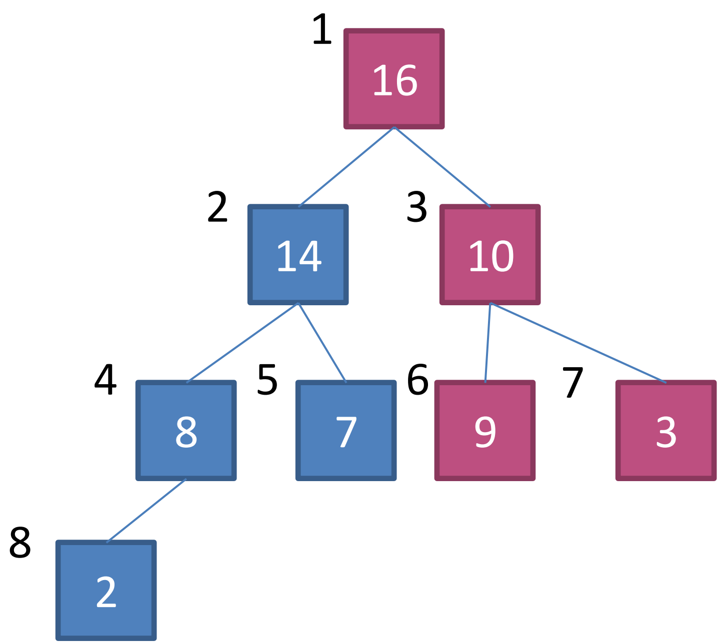
2*i = 6

2*i+1=7



Max-Heap

- Use an array as a heap



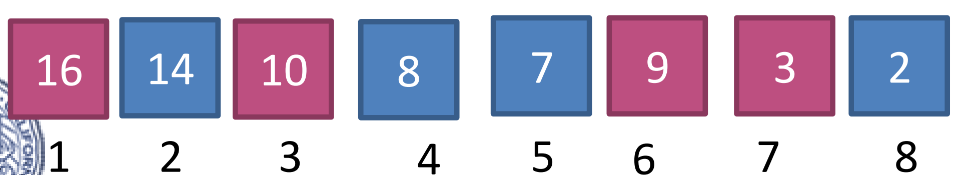
For element at *i*:
Parent index =parent(*i*)= floor(*i*/2);
Left child index = left(*i*)=2**i*;
Right child index =right(*i*)=2**i* +1
Last non-leaf node = floor(length/2)

i=3

floor(*i*/2)=floor(1.5)=1

2**i* = 6

2**i*+1=7



floor(length/2)=4

Max-Heapify

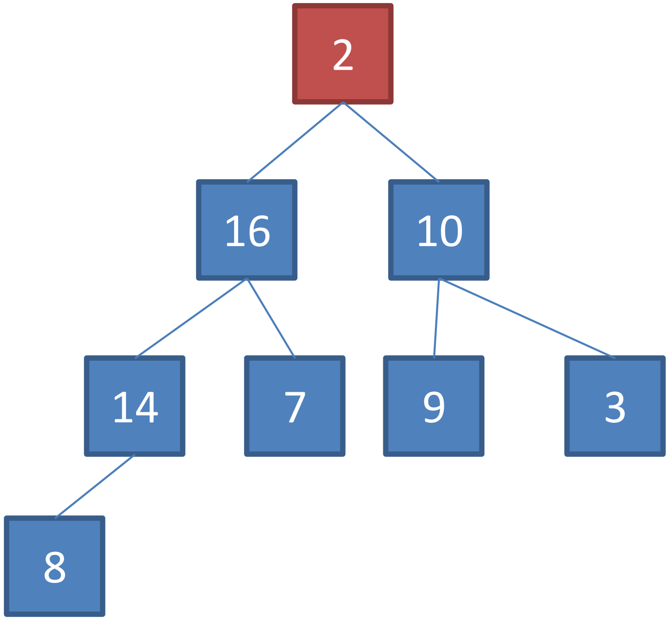
- <u>Input</u>: A complete binary tree A, rooted at i, ended at t, whose left and right sub trees are max-heaps; last node index
- <u>Output</u>: A max-heap rooted at i.
- Algorithm:

  MAX-HEAPIFY (A, i, t)

  1. if(right(i)>t and left(i)>t) return;
  2. Choose the largest node among node i, left(i), right(i) .
  3. if(the largest node is not i) {
     - m = the index of the larger node
     - Exchange i with the largest node
     - **MAX-HEAPIFY** (A, m, t)

     }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

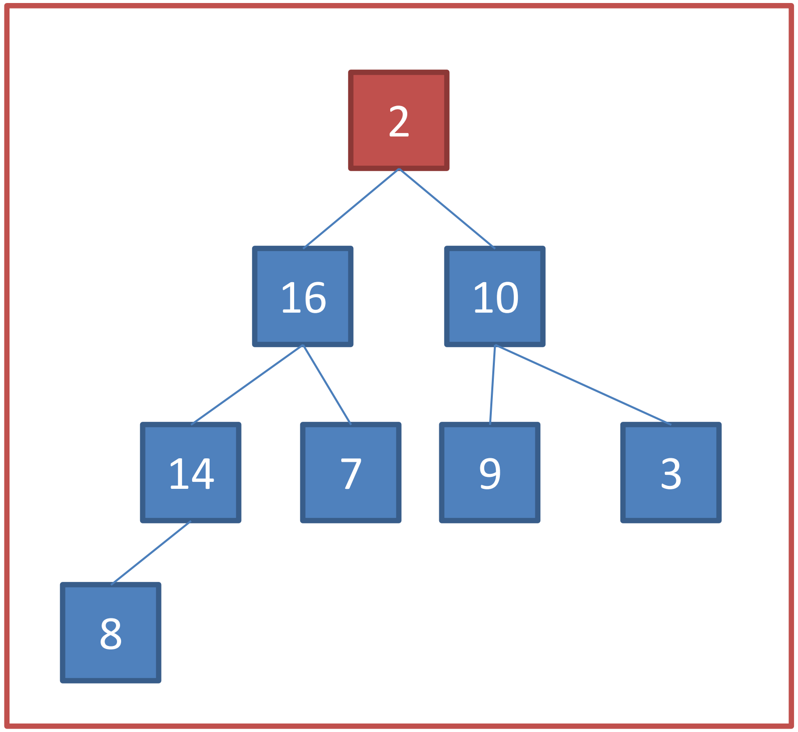
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

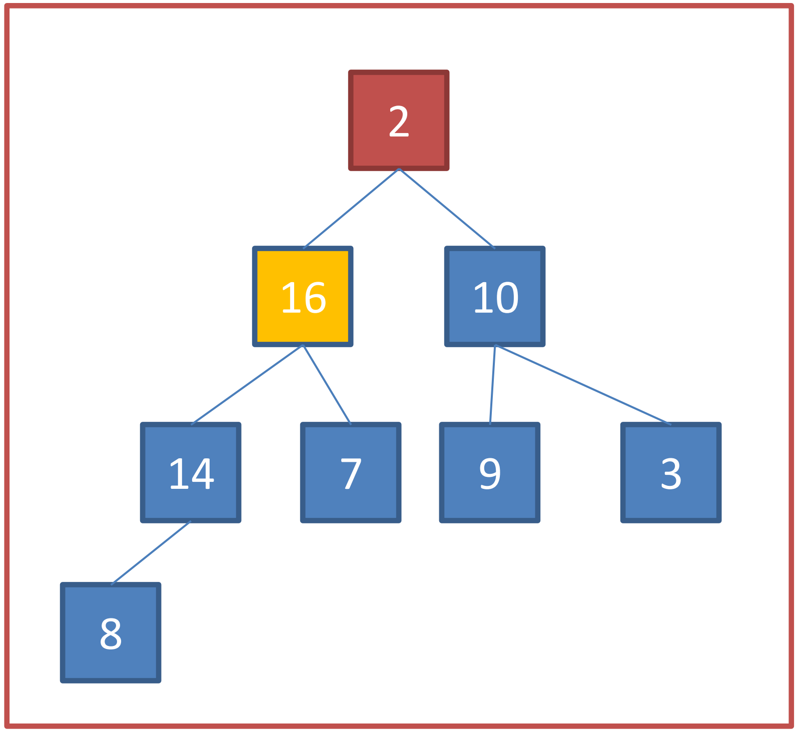
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

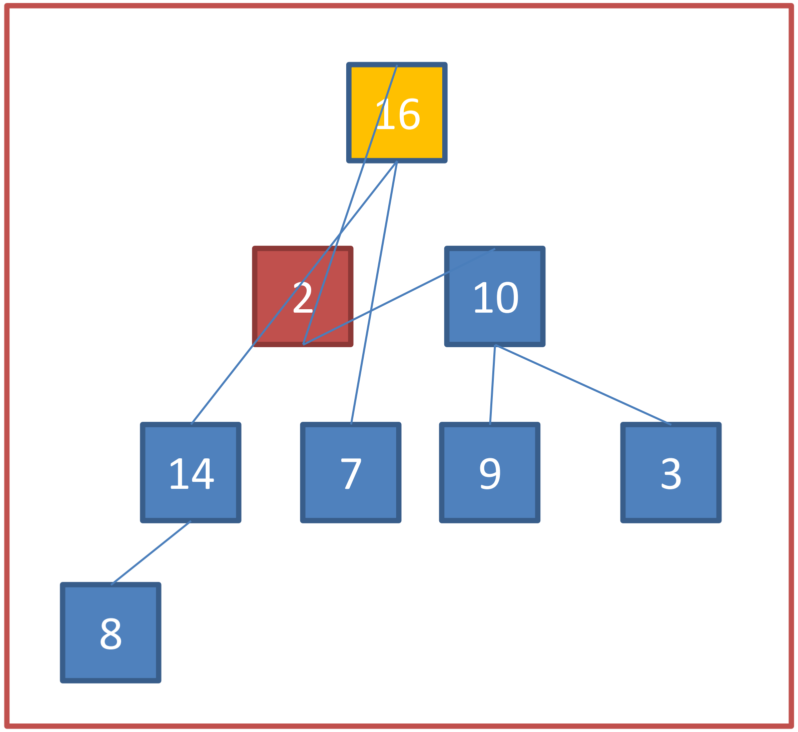
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

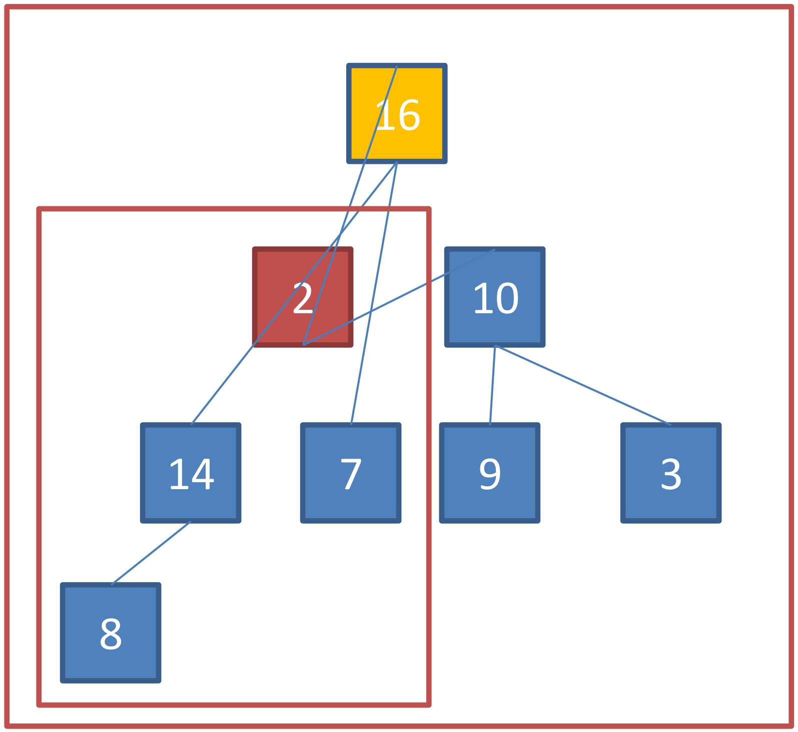
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

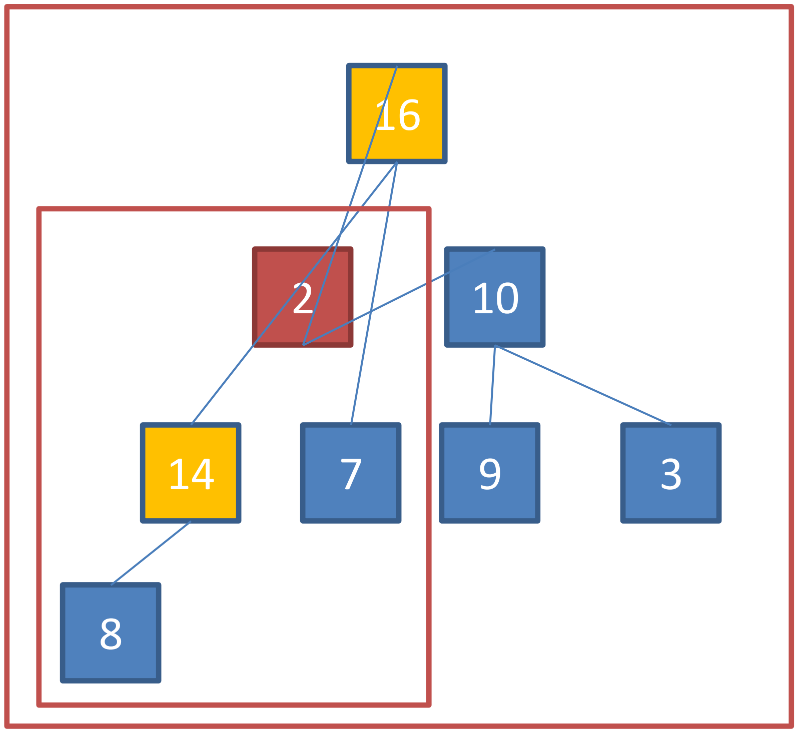
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

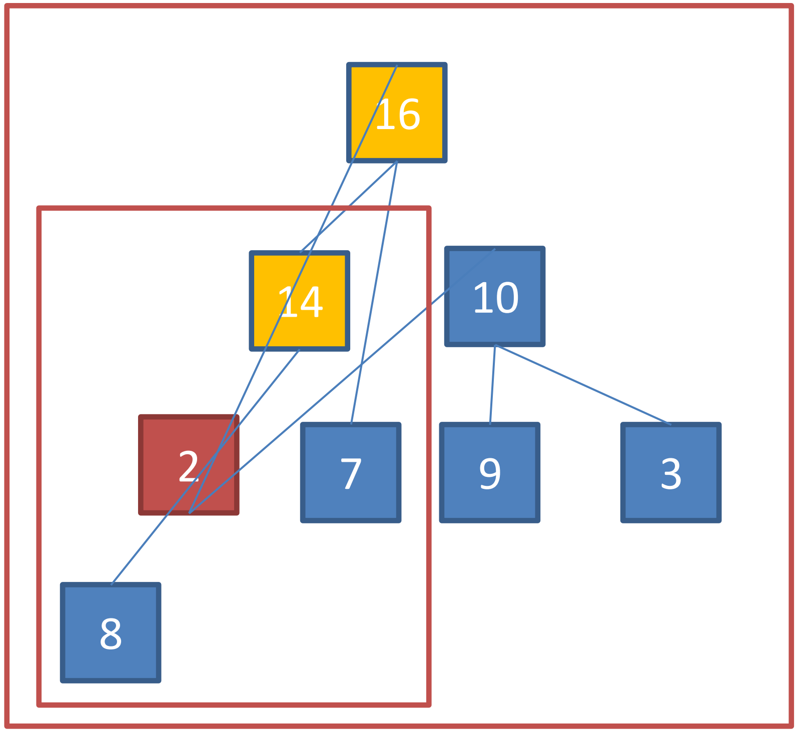
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

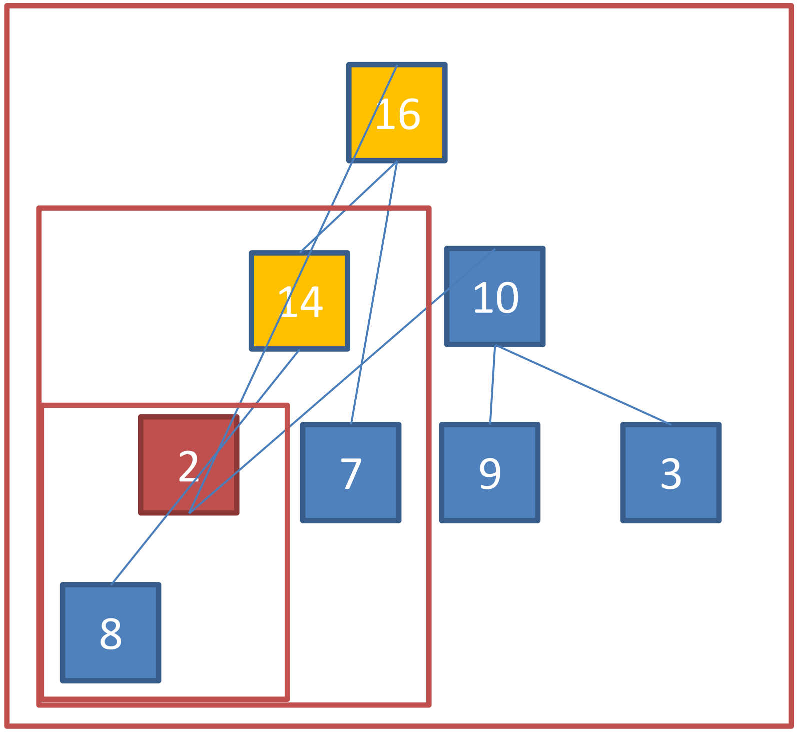
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

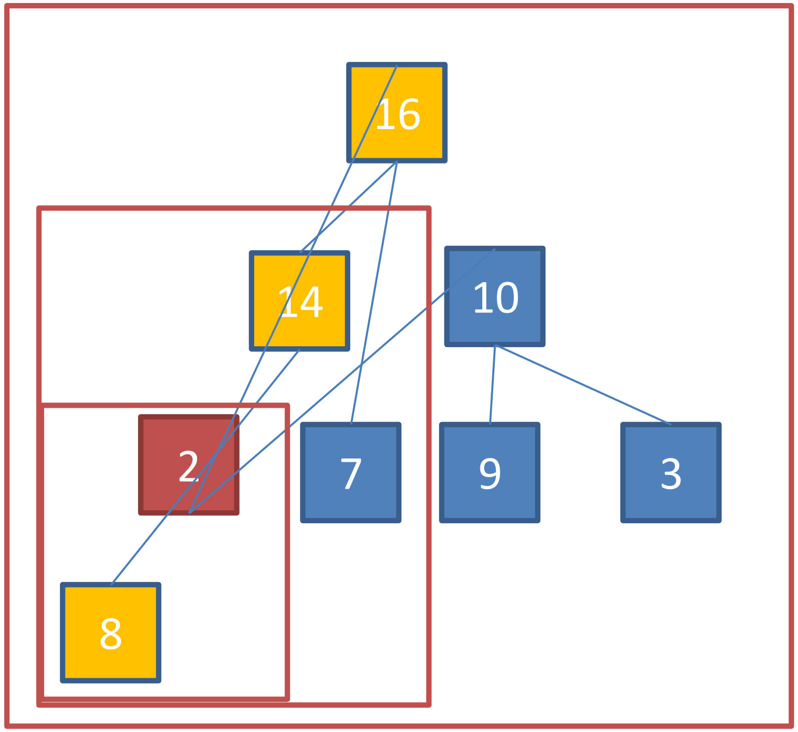
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

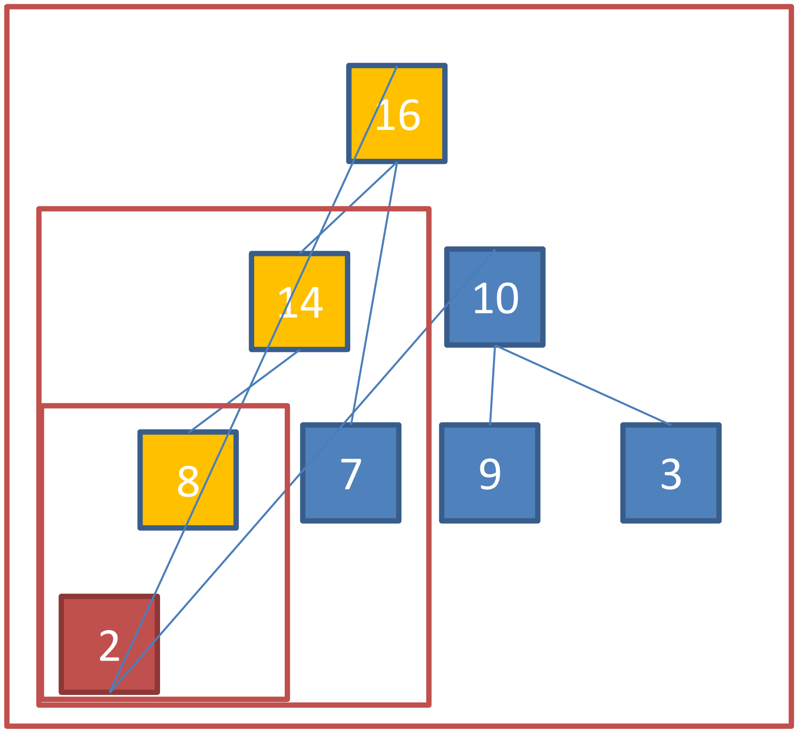
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

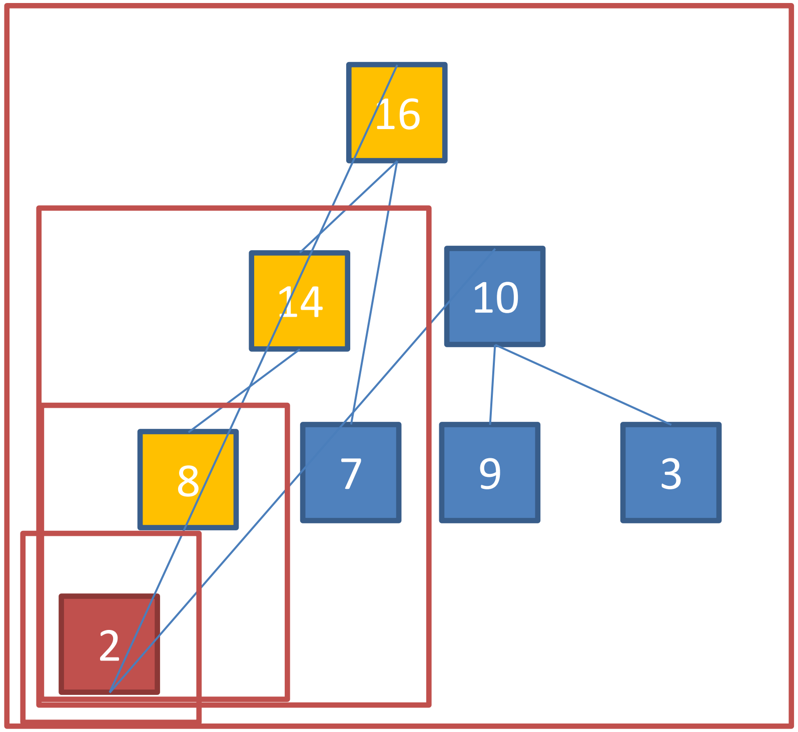
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

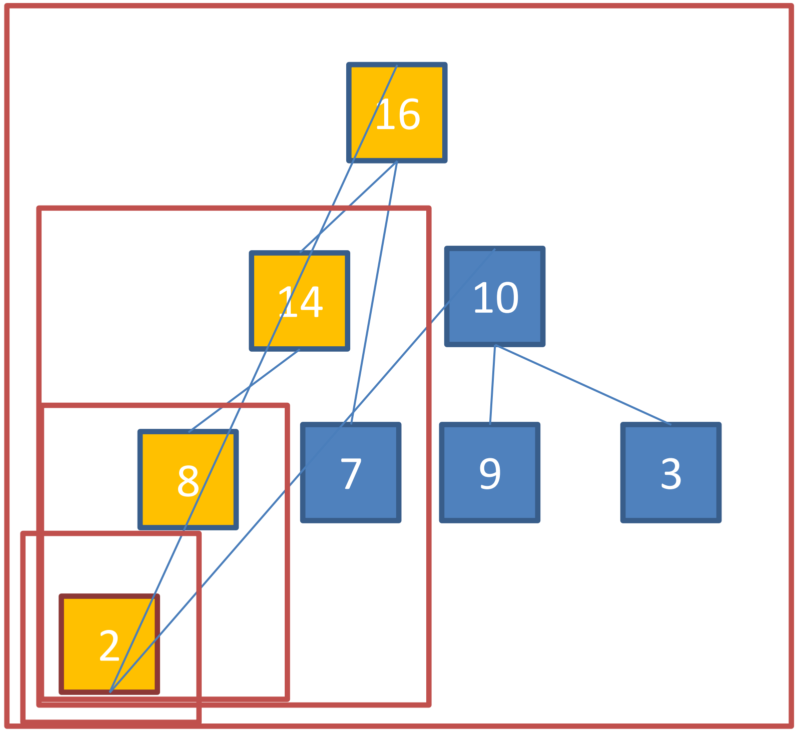
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

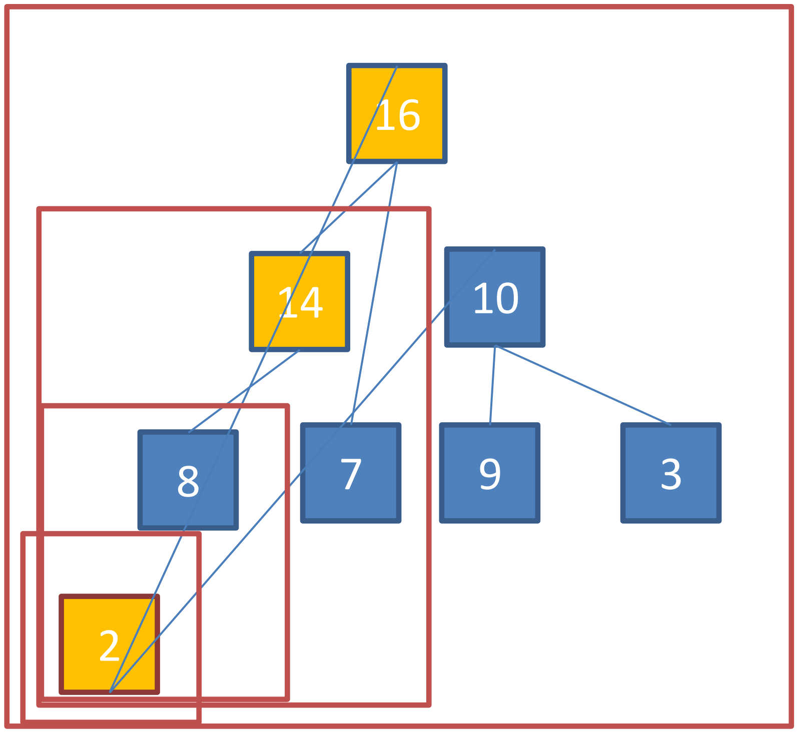
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

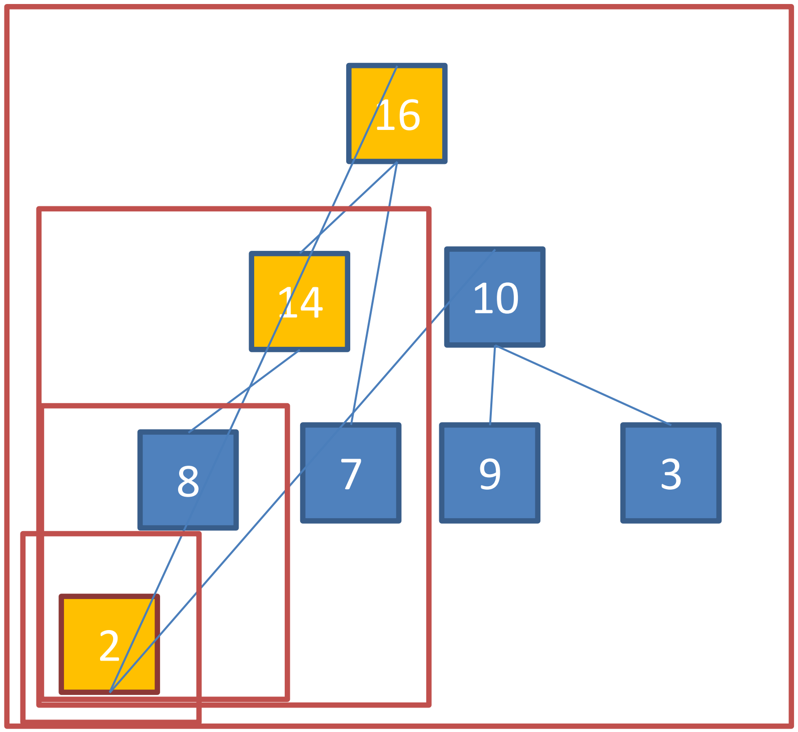
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Max-Heapify Example



MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

<u>Final result</u>: a Max-Heap rooted at i

Max-Heapify Running Time

- The running time of MAX-HEAPIFY on a subtree of size *n* rooted at a given node *i* is the O(1) time to fix up the relationships among the elements A[*i*], A[LEFT(*i*)], and A[RIGHT(*i*)]
- Plus the time to run MAX-HEAPIFY on a subtree rooted at one of the children of node *i* (assuming that the recursive call occurs)

Max-Heapify Running Time

- The children's subtrees each have size at most $2n/3$
- The worst case occurs when the bottom level of the tree is exactly half full
- Recurrence formula:
  - $T(n) = T(n/(3/2)) + O(1)$

For a more detailed explanation of this, see: https://hongyuhe.github.io/heap-sort/

- Using the Master Theorem we get:
  - a=1; b=3/2; d=0 → a=bd →
  - $T(n) = O(n^d \log(n)) = O(\log(n))$

Build-Max-Heap

- We can build a heap in a bottom-up manner by running Max-Heapify on successive subarrays
  - Walk backwards through the array from n/2 to 1, calling Max-Heapify() on each node
  - Order of processing guarantees that the children of node *i* are heaps when *i* is processed.

Array -> Max-Heap

- Input: an array A
- Output: a Max-Heap A
- Algorithm:

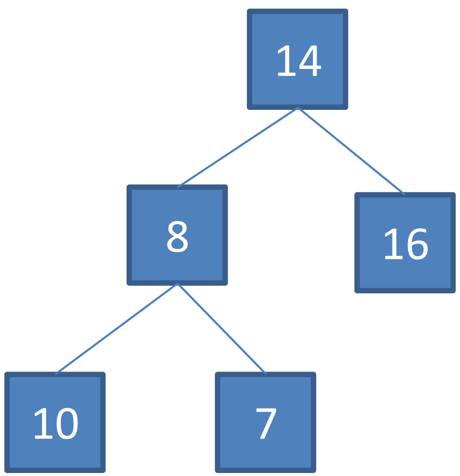
BUILD-MAX-HEAP(A):

Considering A as a complete binary tree, from the last non-leaf node to the first one i (in reverse) {

    MAX-HEAPIFY(A, i, A.lastIndex);

}

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

  MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

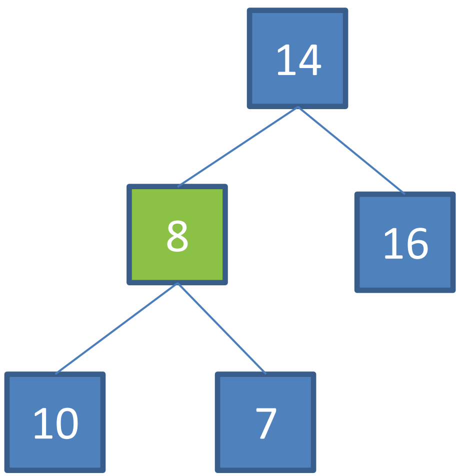
  m = the index of the larger node

  Exchange i with the largest node

  MAX-HEAPIFY (A, m, t)

 }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

    MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

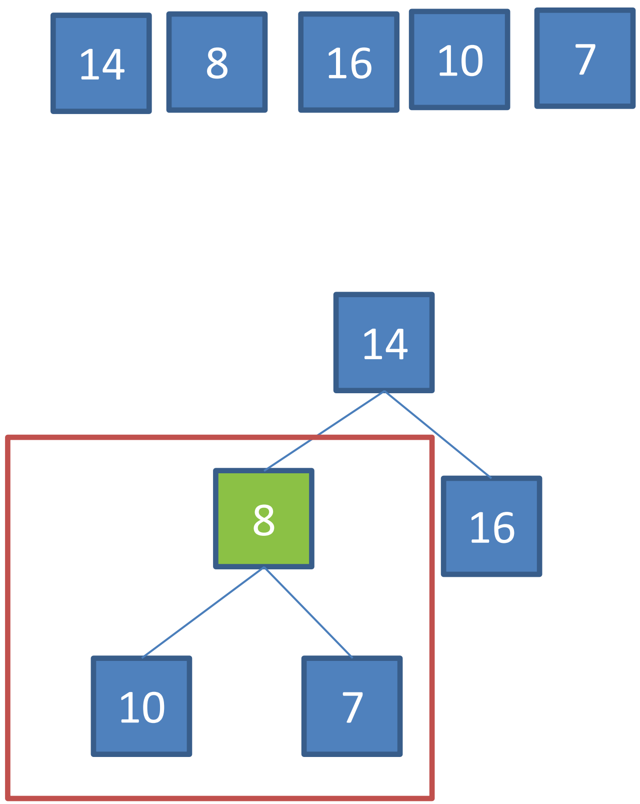
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

    MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

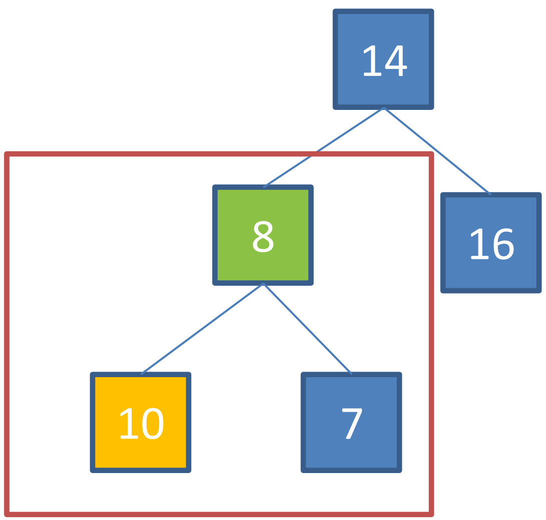
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

  MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

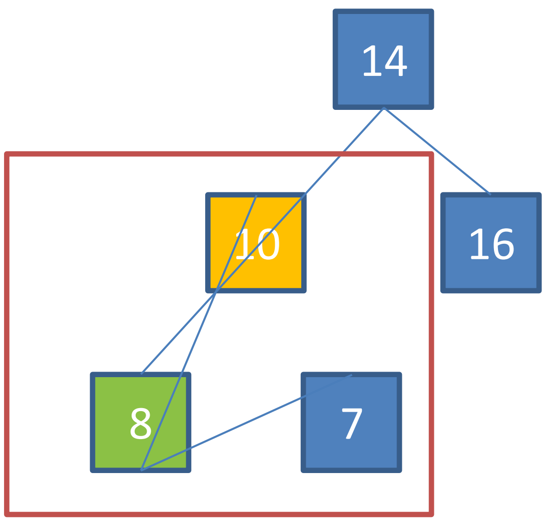
  m = the index of the larger node

  Exchange i with the largest node

  MAX-HEAPIFY (A, m, t)

 }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

 MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

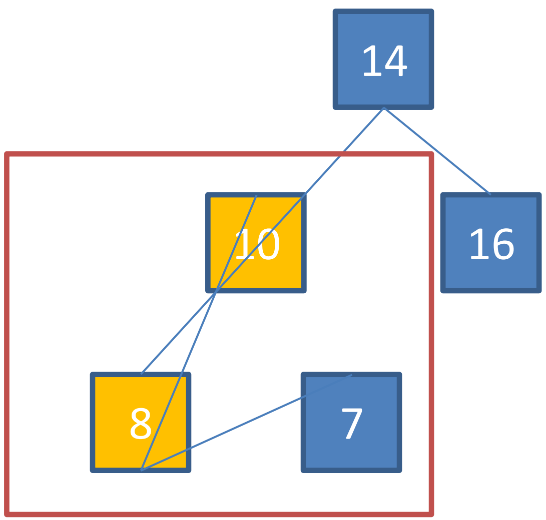
 m = the index of the larger node

 Exchange i with the largest node

 MAX-HEAPIFY (A, m, t)

}

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

  MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

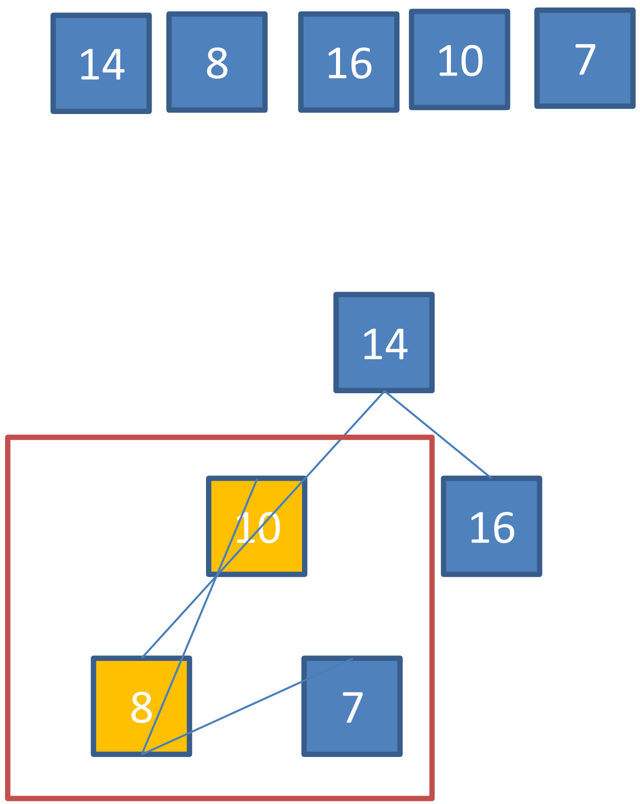
  m = the index of the larger node

  Exchange i with the largest node

  MAX-HEAPIFY (A, m, t)

 }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

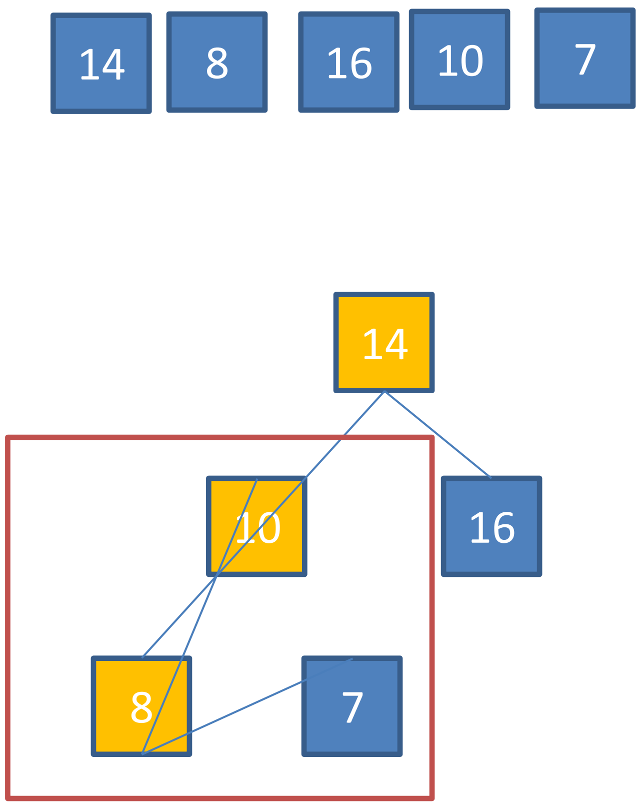
m = the index of the larger node

Exchange i with the largest node

MAX-HEAPIFY (A, m, t)

}

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

  MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

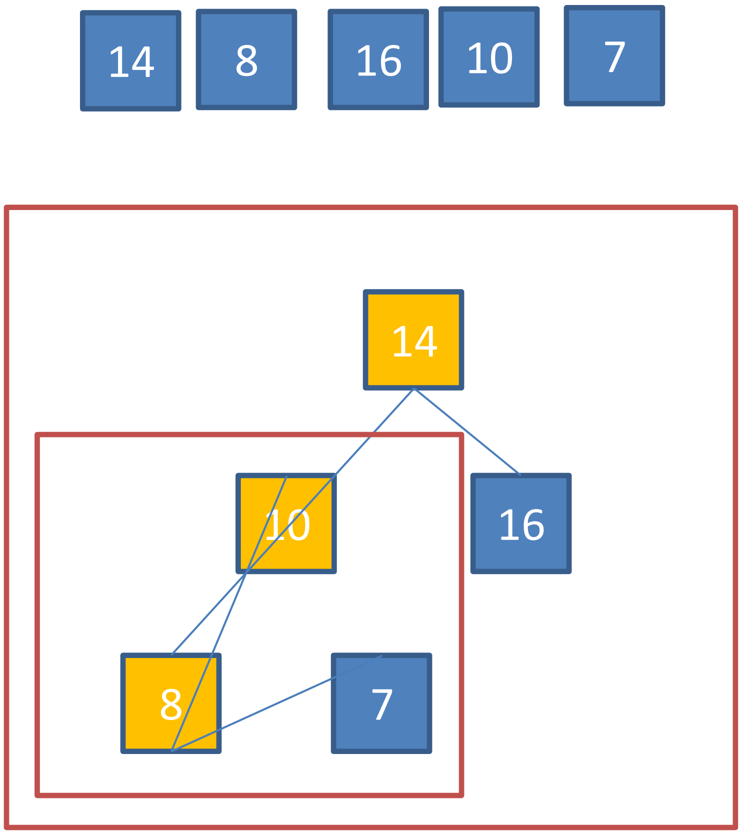
  m = the index of the larger node

  Exchange i with the largest node

  MAX-HEAPIFY (A, m, t)

  }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

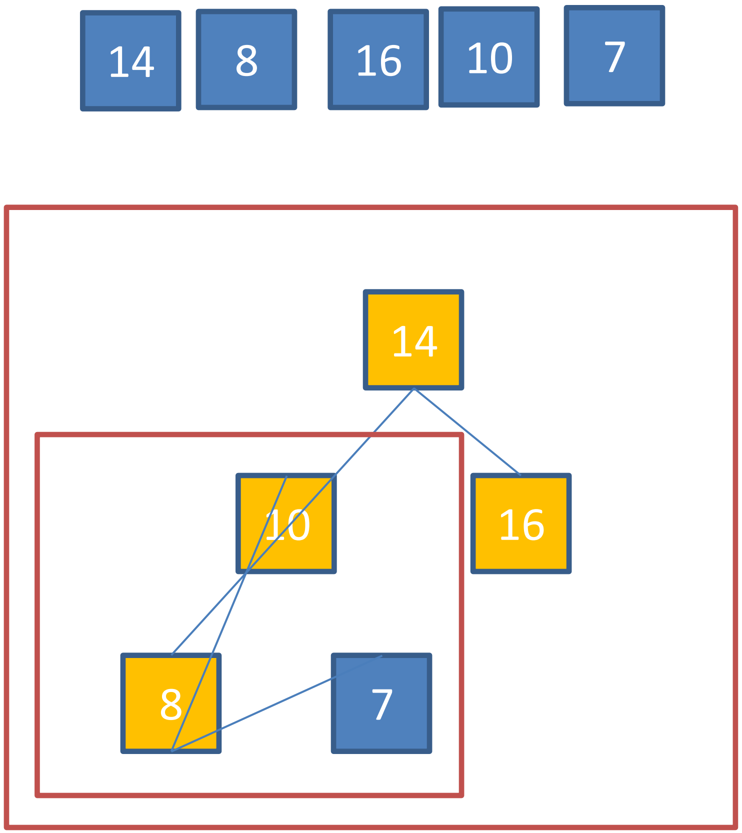
m = the index of the larger node

Exchange i with the largest node

MAX-HEAPIFY (A, m, t)

}

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

    MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

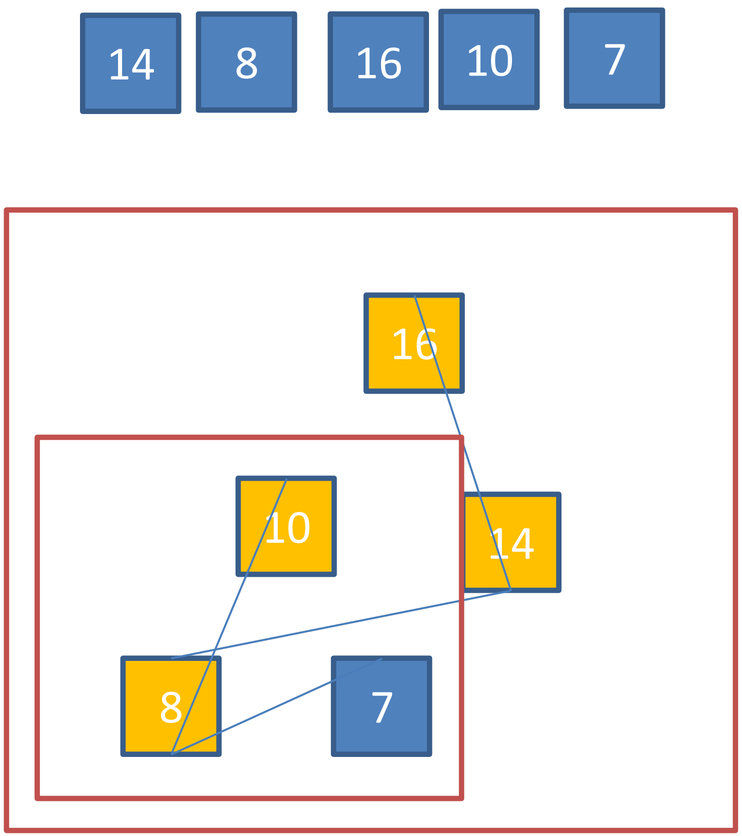
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

    MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

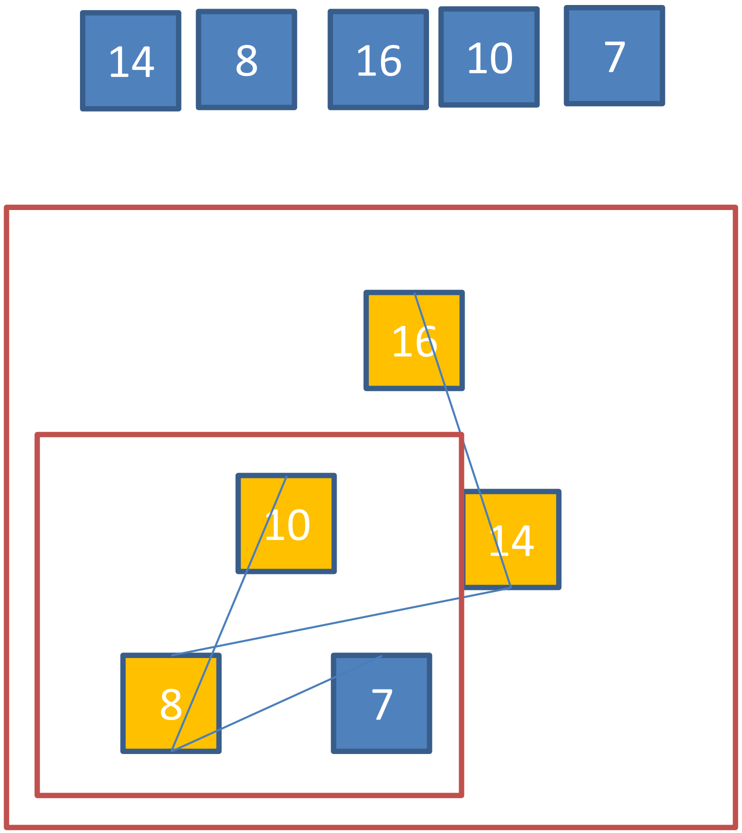
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

    MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

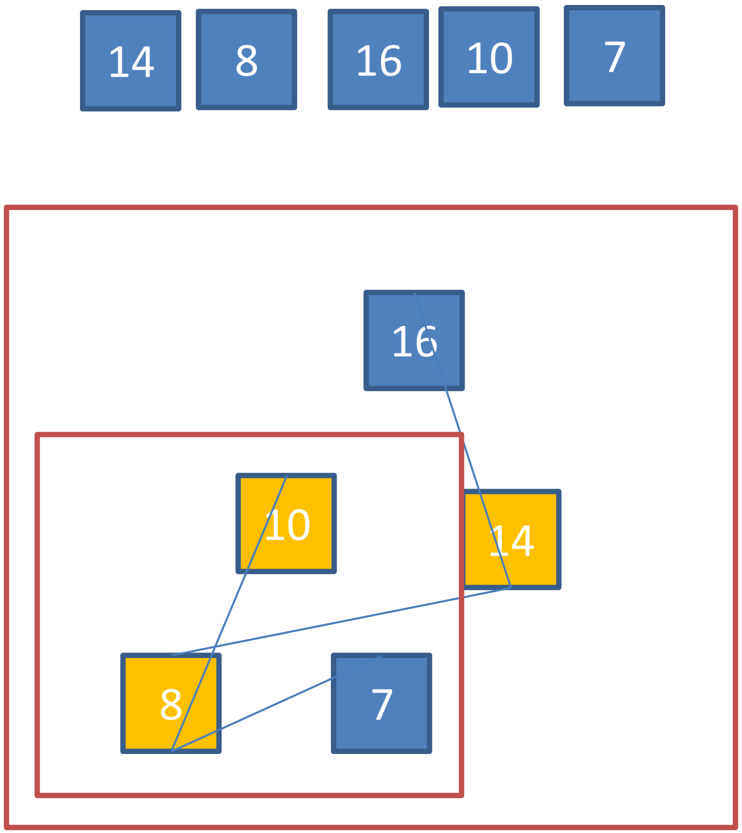
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

  MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

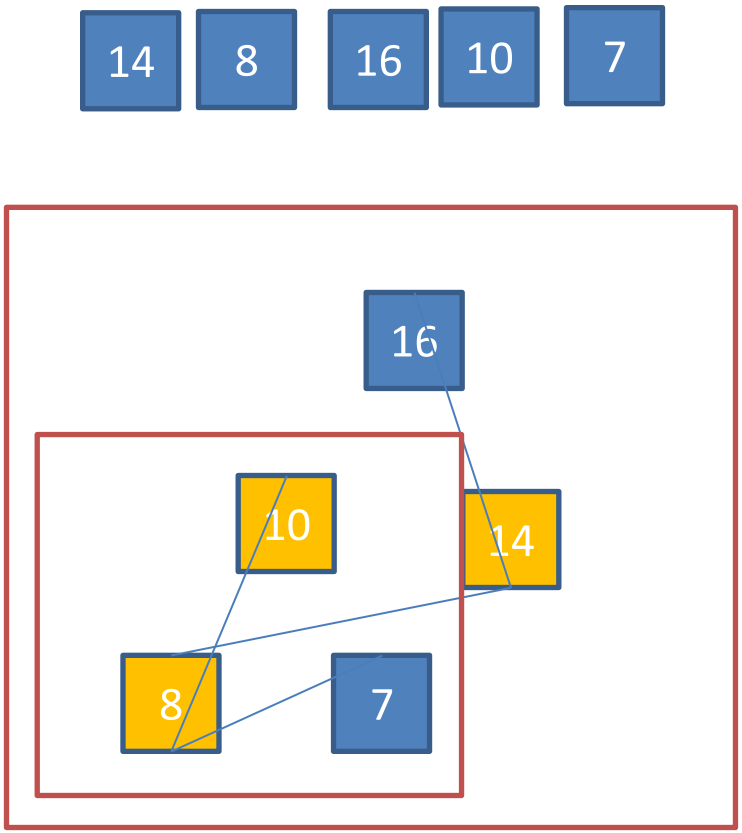
  m = the index of the larger node

  Exchange i with the largest node

  MAX-HEAPIFY (A, m, t)

}

Build Max-Heap Example



BUILD-MAX-HEAP(A):

Considering A as a complete binary tree,

from the last non-leaf node to the first one i {

    MAX-HEAPIFY(A, i, A.lastIndex);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

    m = the index of the larger node

    Exchange i with the largest node

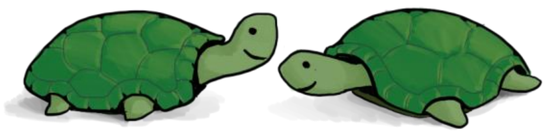
    MAX-HEAPIFY (A, m, t)

  }

Final result: a Max-Heap A

Build-Max-Heap Running Time

- Each call to Max-Heapify() takes O(log(n)) time
- There are O(n) such calls (specifically, floor[n/2])
- Thus the running time is O(n.log(n))
  - Is this a correct asymptotic upper bound?
  - Is this an asymptotically tight bound?
- A tighter bound is O(n)
  - How can this be? Is there a flaw in the above reasoning?



Think-Pair-Share Terrapins (in-class questions)

Read CLRS pages 157-159 for an answer.

Siggi the Studious Stork (recommended exercises)

Ollie the Over-achieving Ostrich (challenge questions)

Heapsort for a Heap

- Input: array A
- Output: a sorted array A
- Algorithm:

  HEAP-SORT (A):

  1. **BUILD-MAX-HEAP**(A)
  2. Last node index i = A's last node index
  3. From the last element to the second in A {

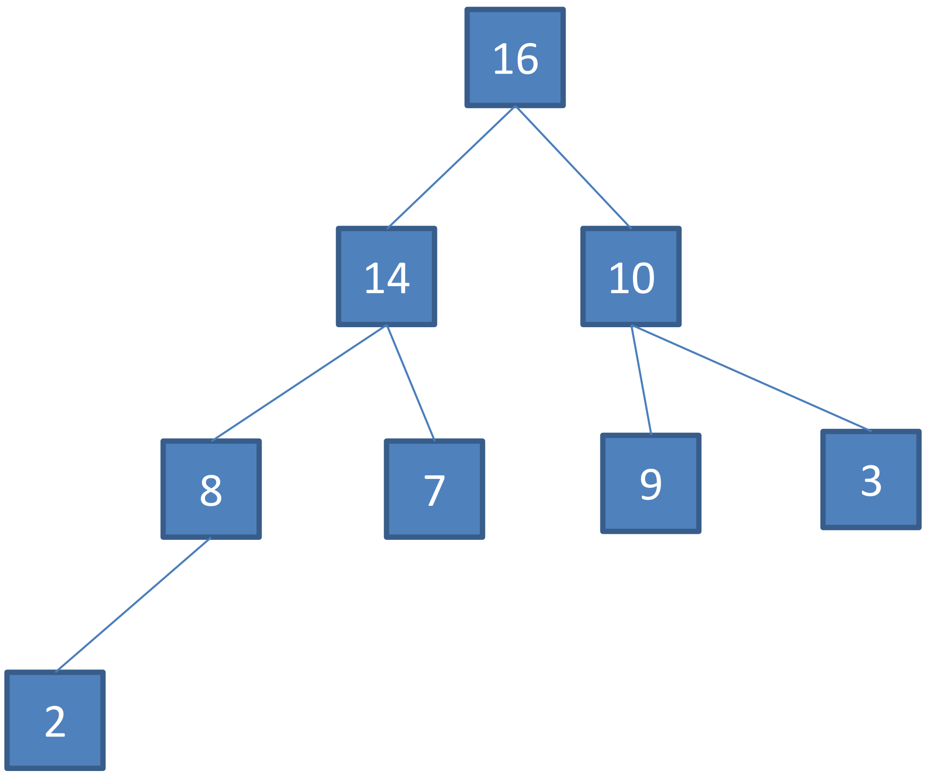
     exchange (i, root);

     i--;

     MAX-HEAPIFY(A, root, i);

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

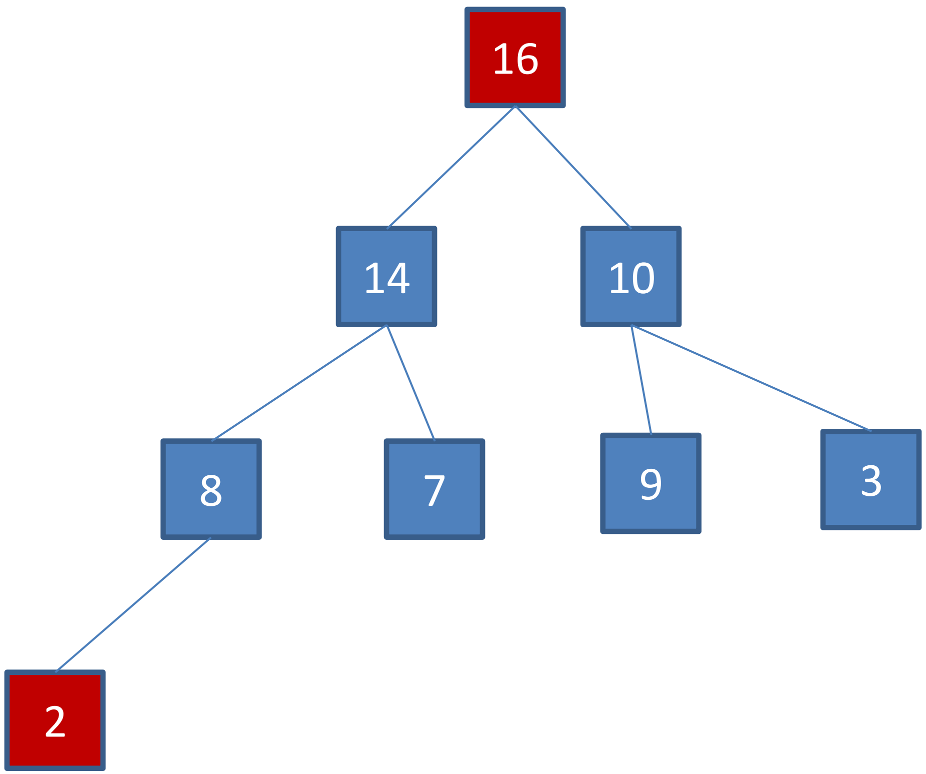
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

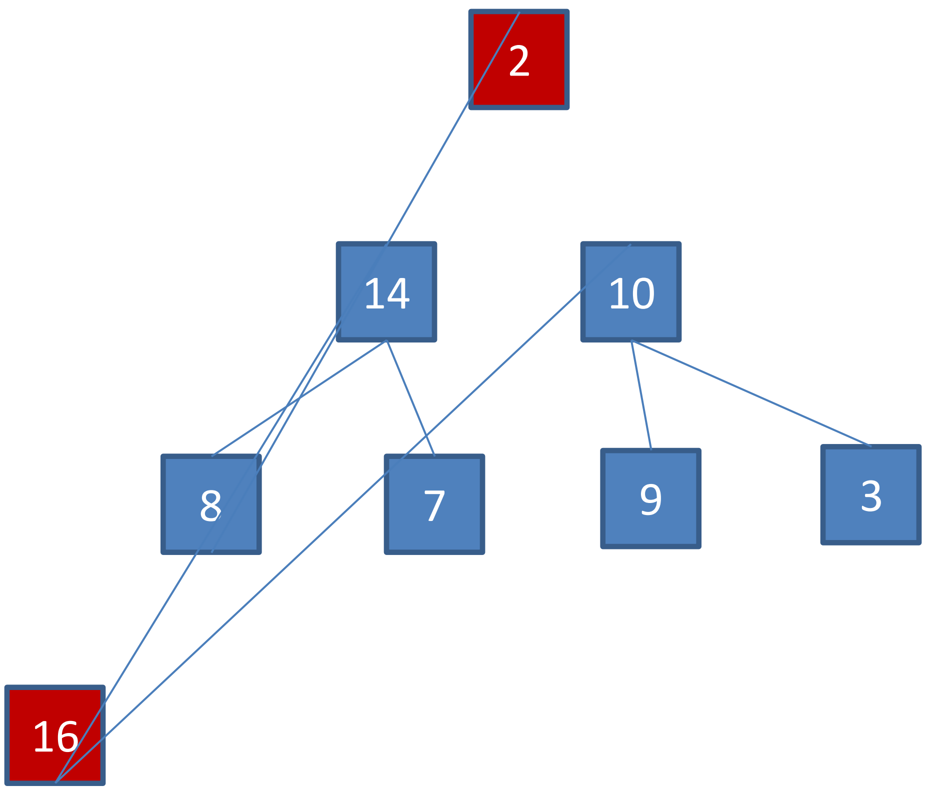
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

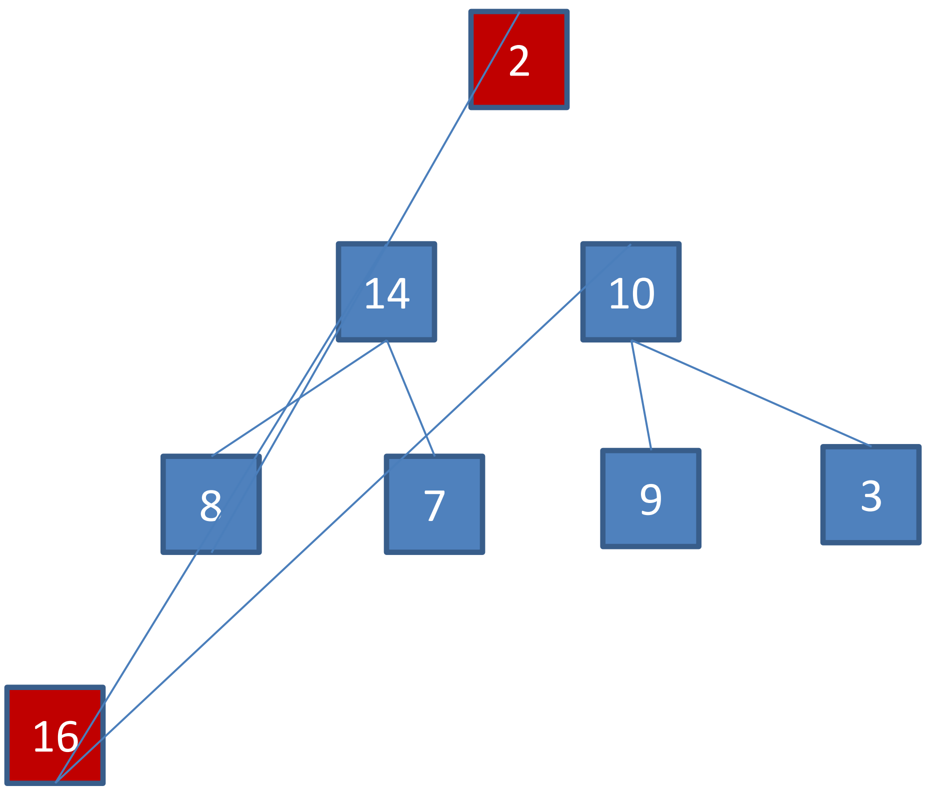
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

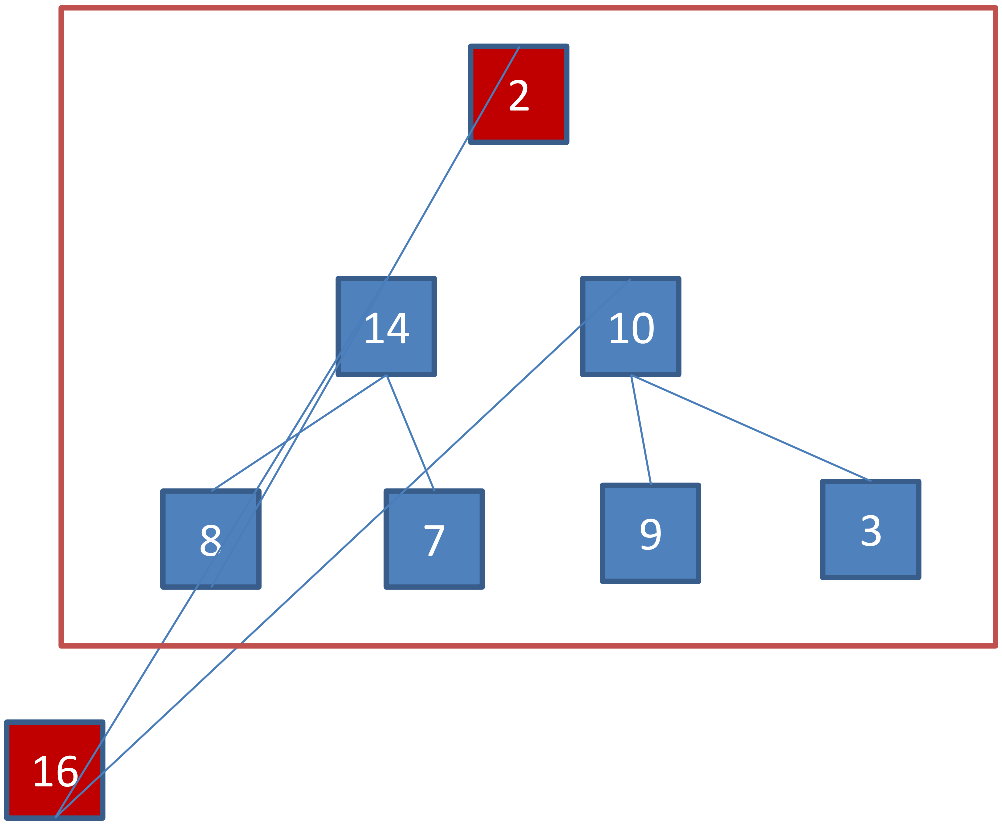
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

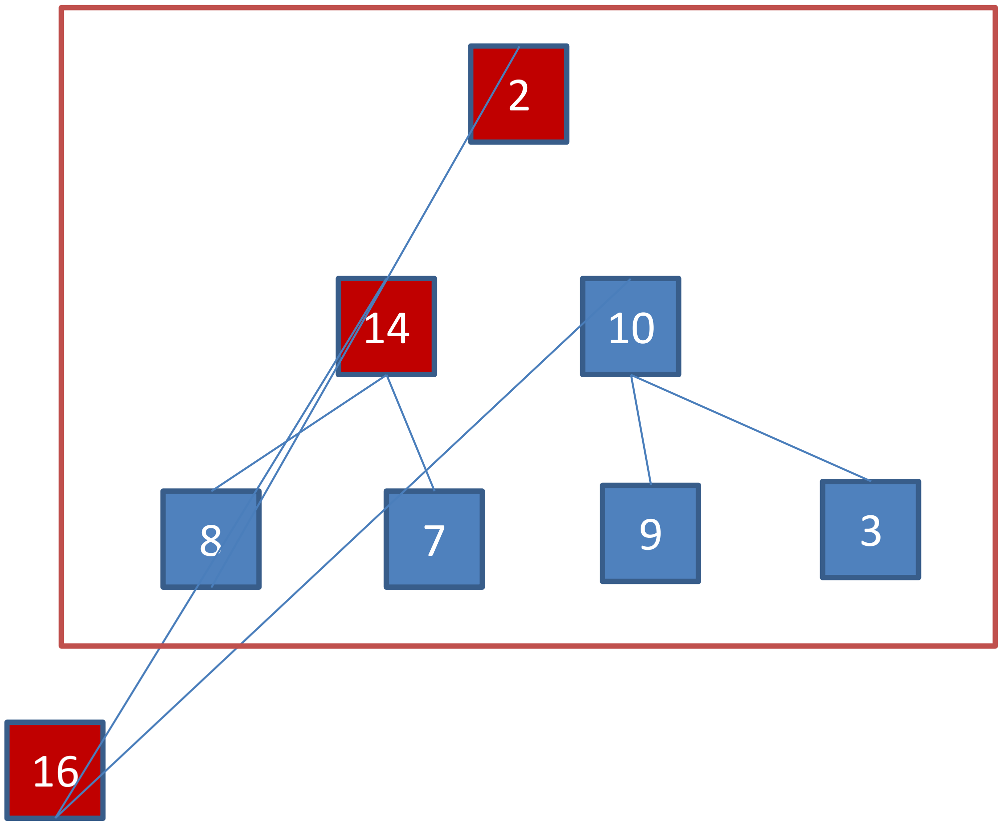
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

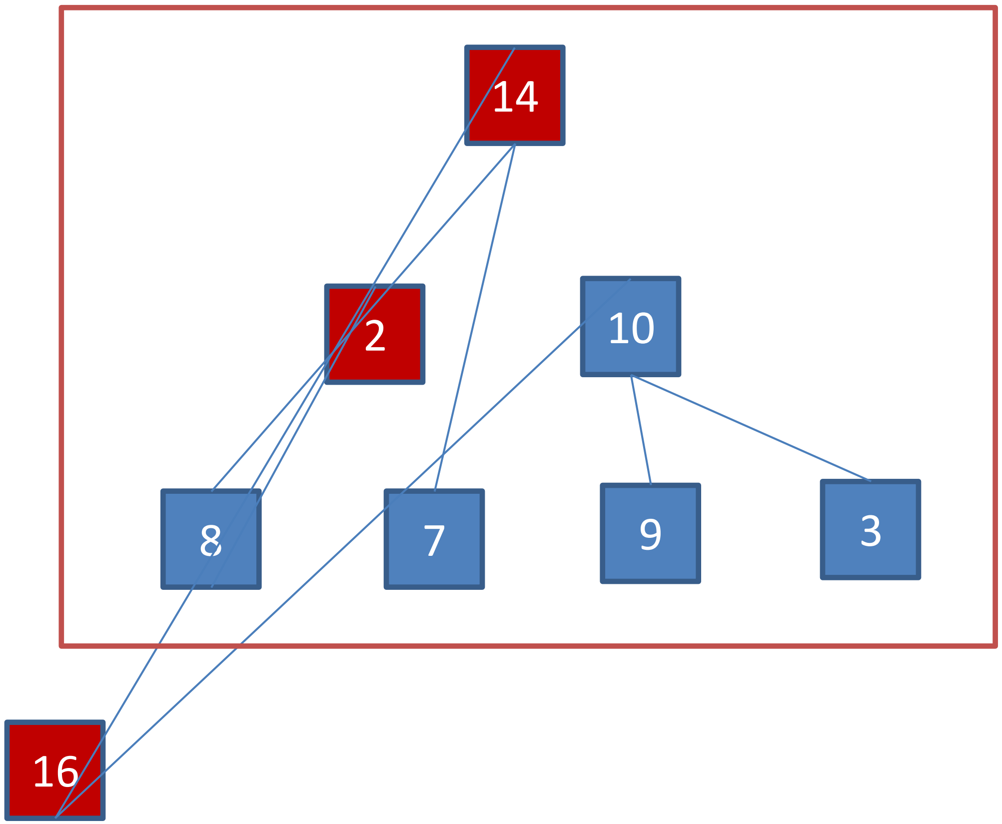
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

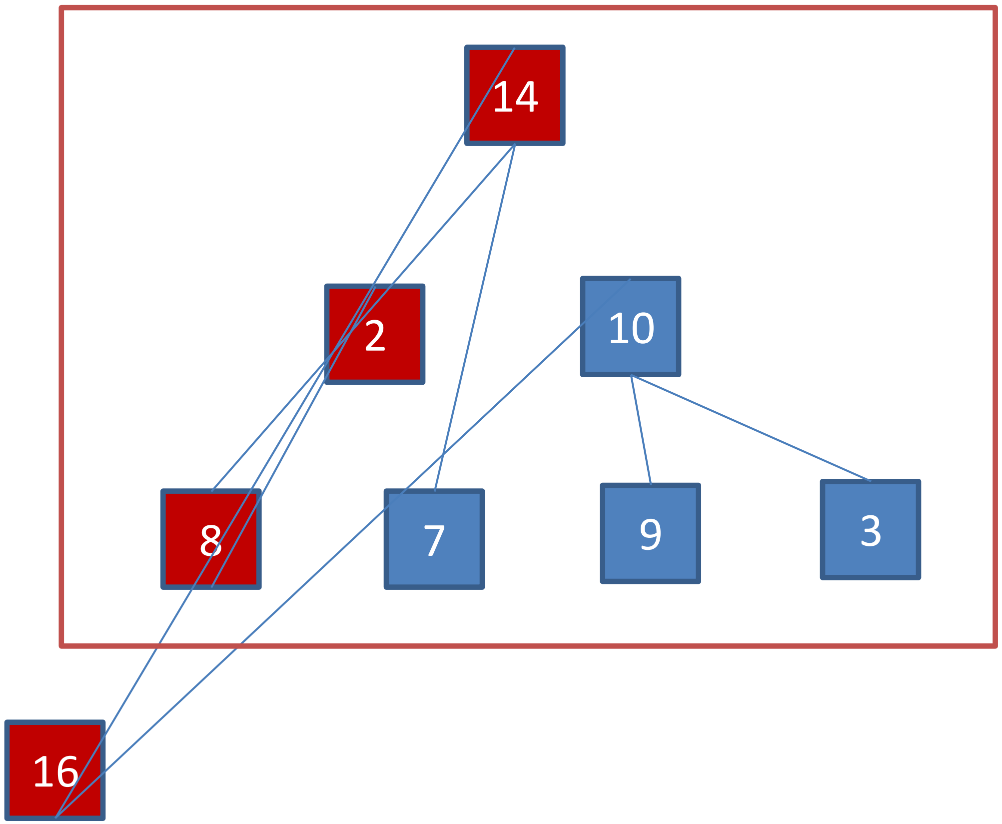
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

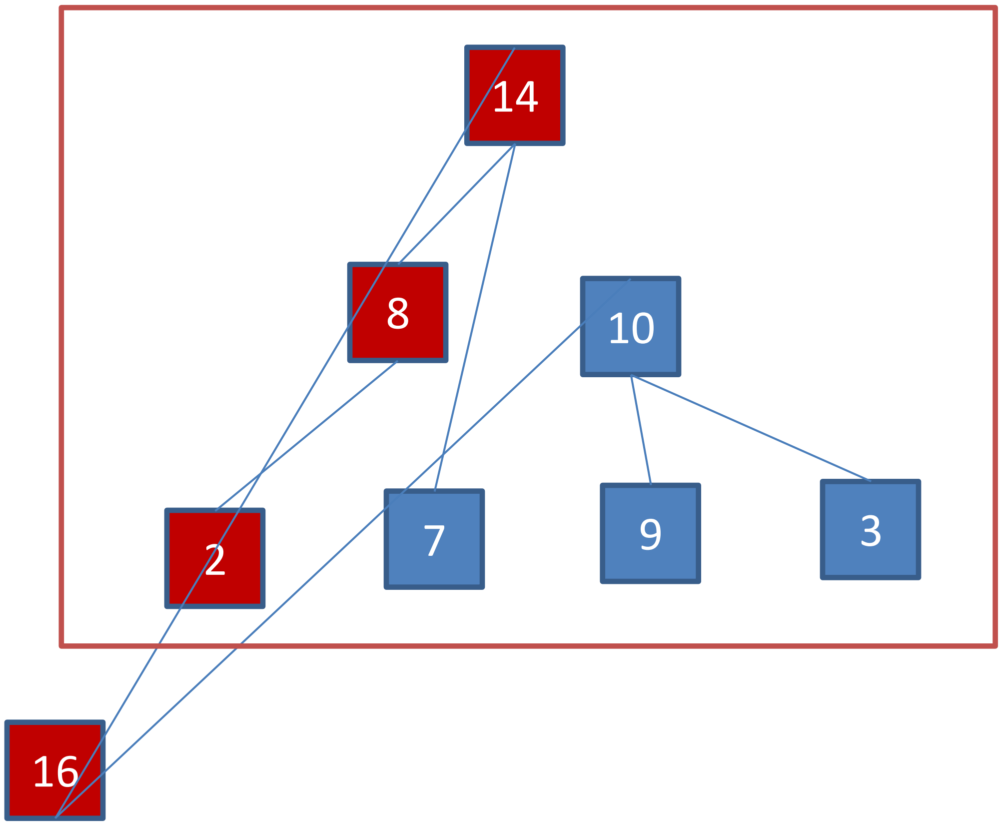
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

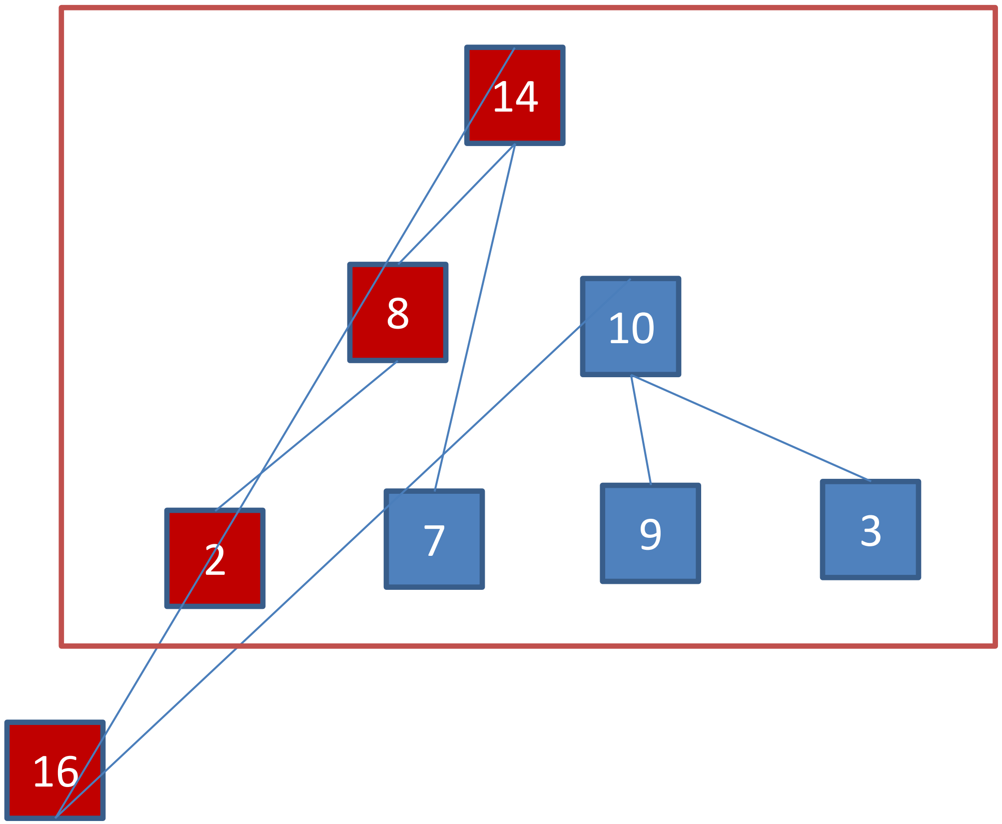
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

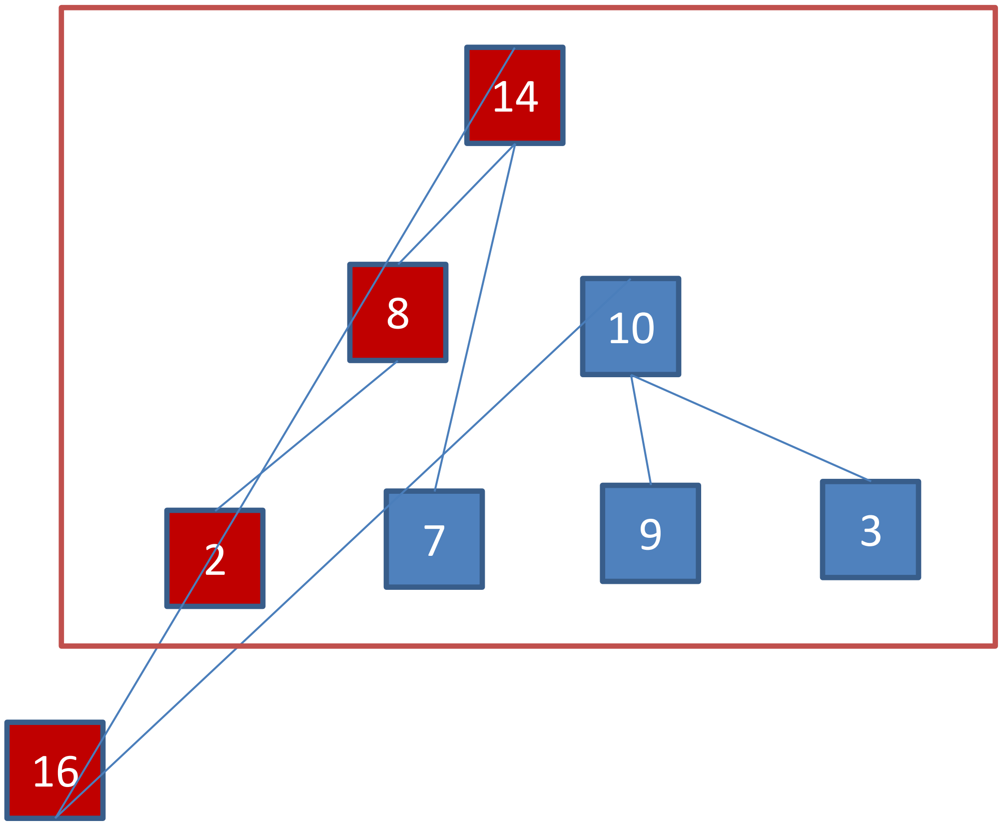
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

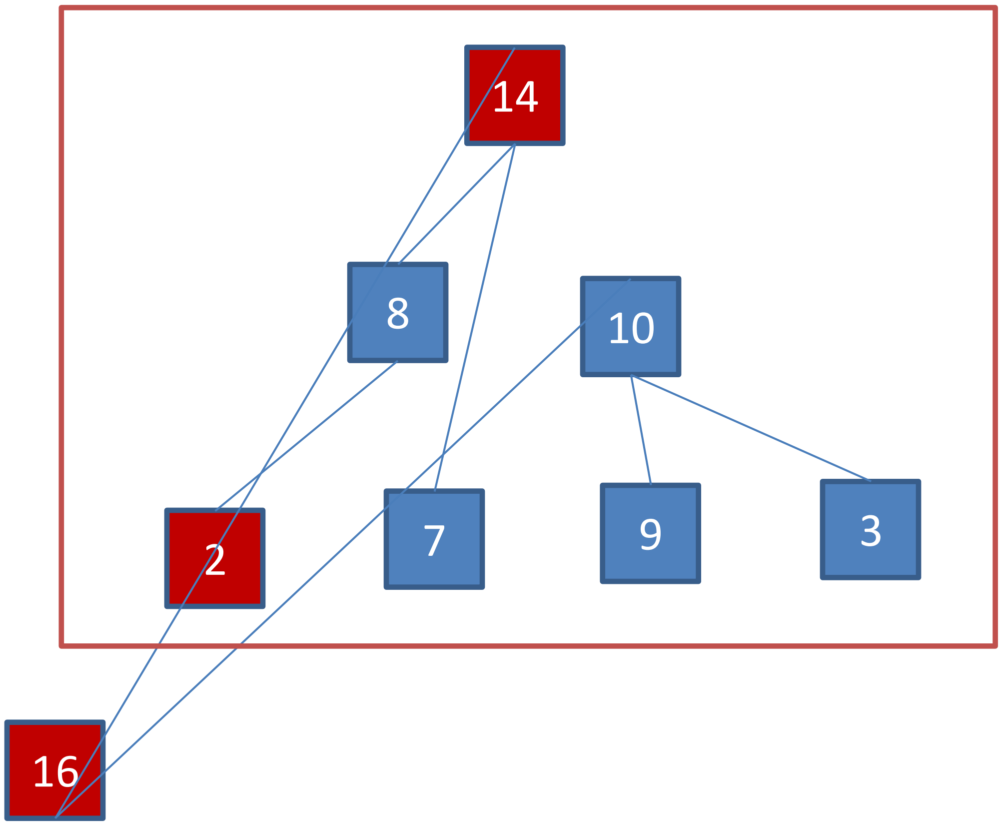
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

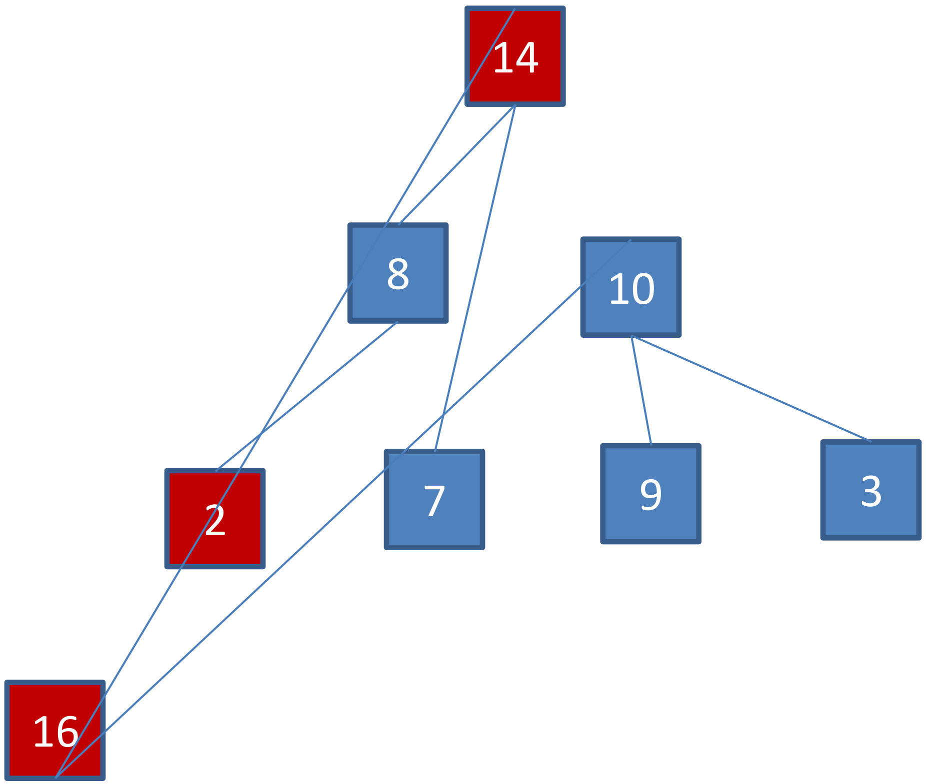
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

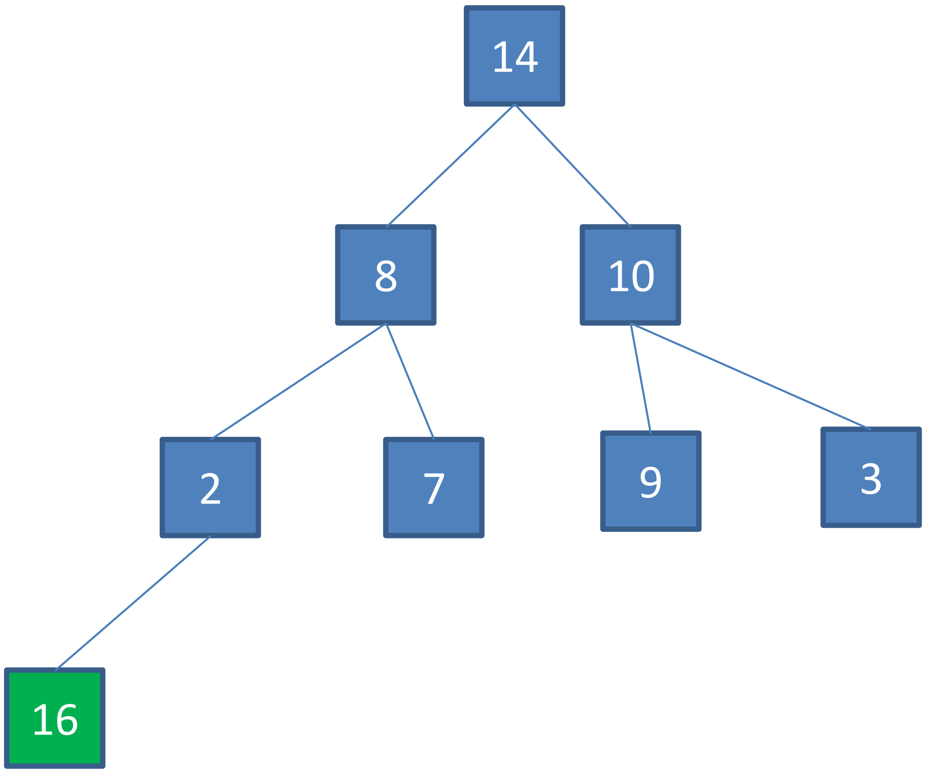
Heapsort Example



HEAP-SORT (A):
1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {
   exchange (i, root);
   i--;
   MAX-HEAPIFY(A, root, i);
   }

MAX-HEAPIFY (A, i, t)
1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)
   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

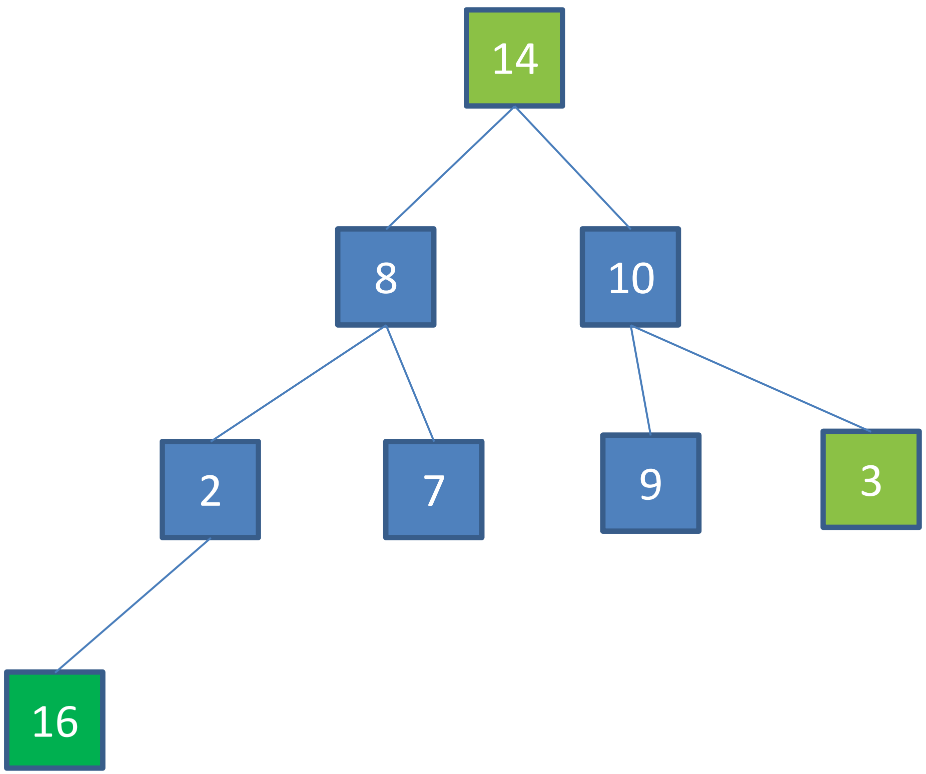
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A’s last node index
3. From last element to the second in A {

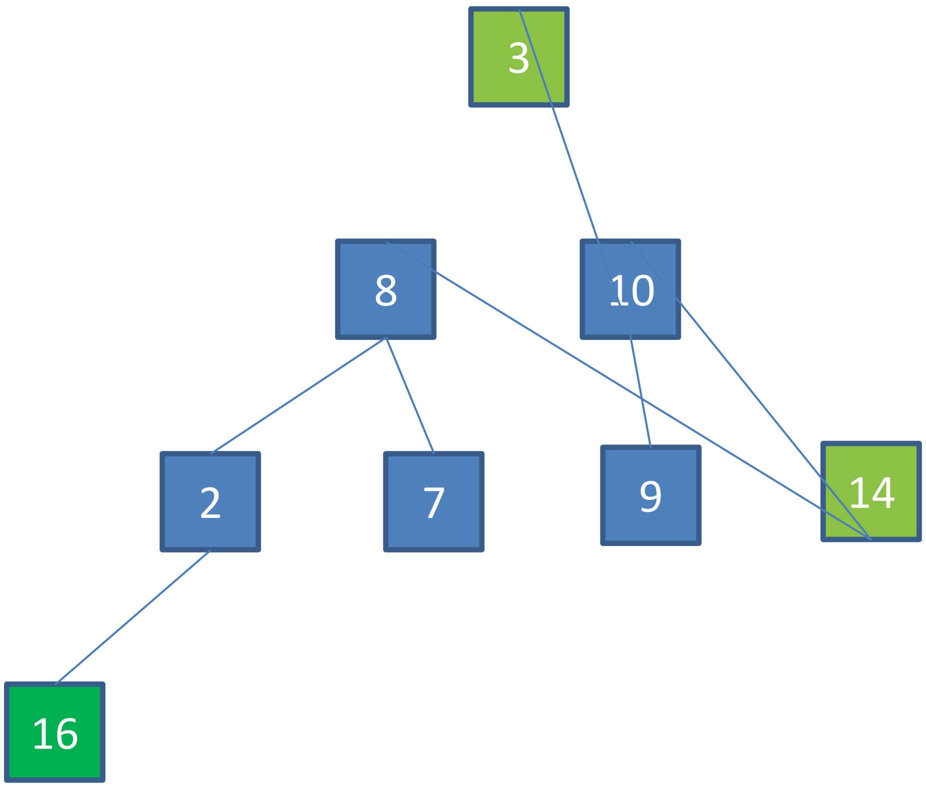
        exchange (i, root);
        i--;
        MAX-HEAPIFY(A, root, i);
    }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

        m = the index of the larger node
        Exchange i with the largest node
        MAX-HEAPIFY (A, m, t)
    }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

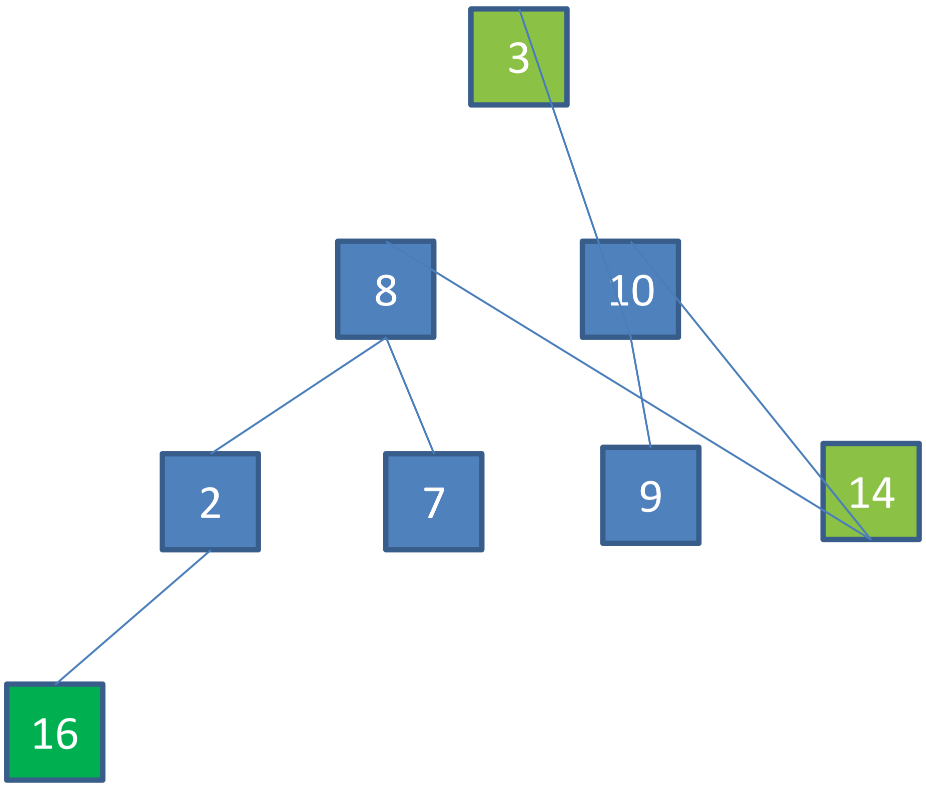
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

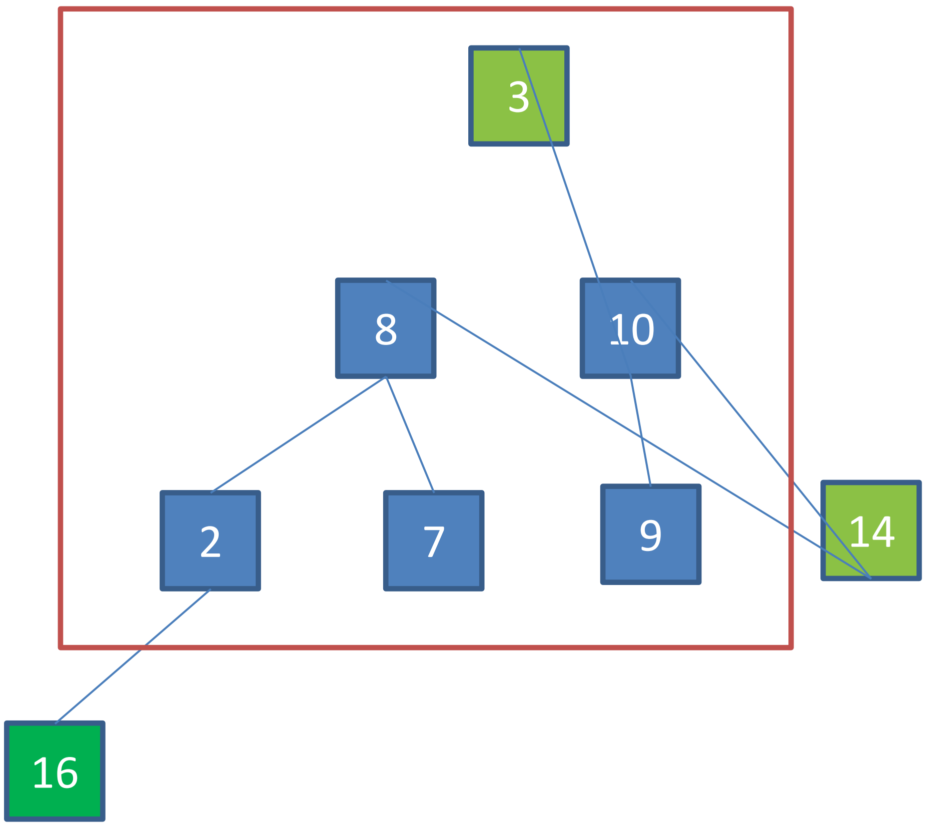
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

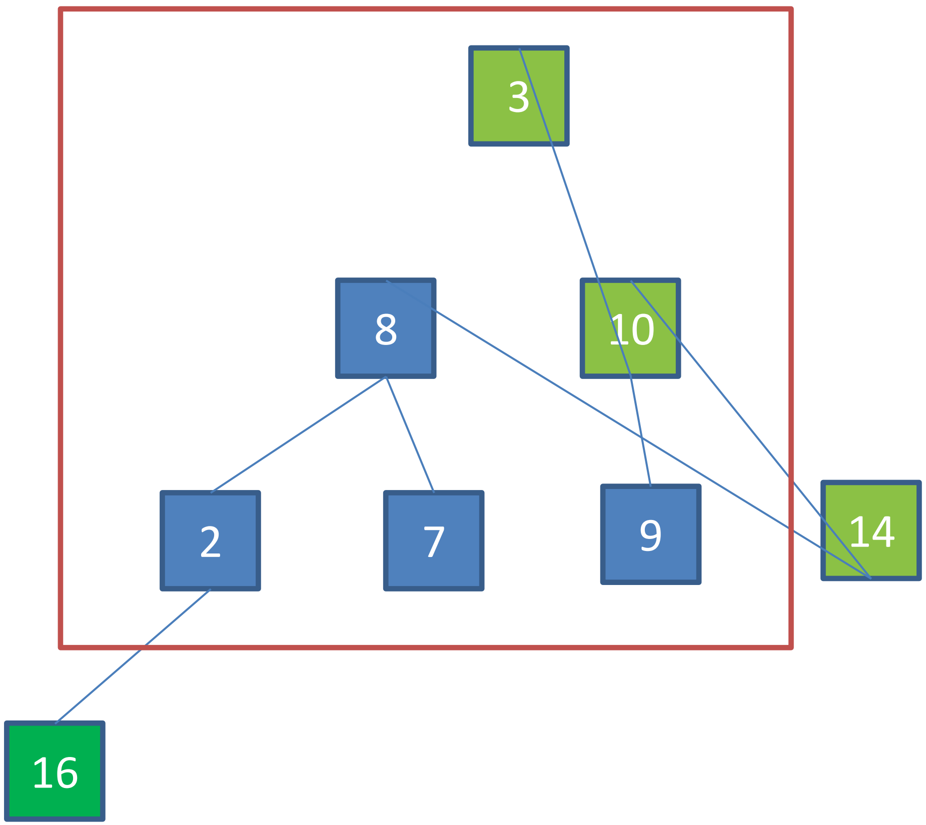
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

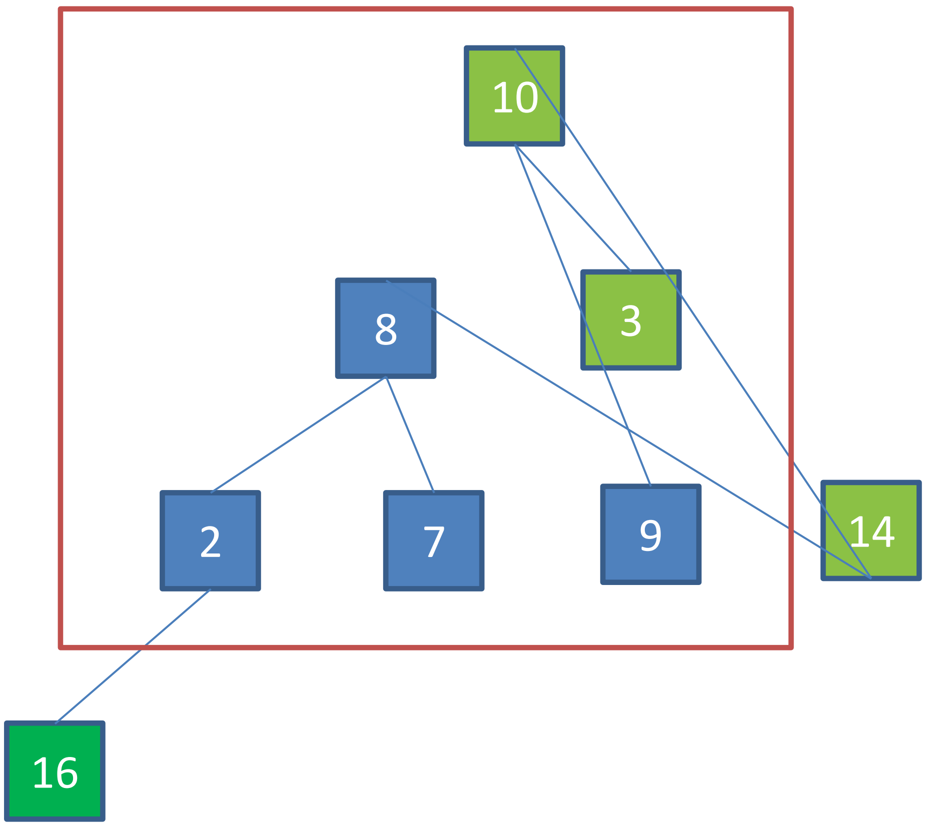
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

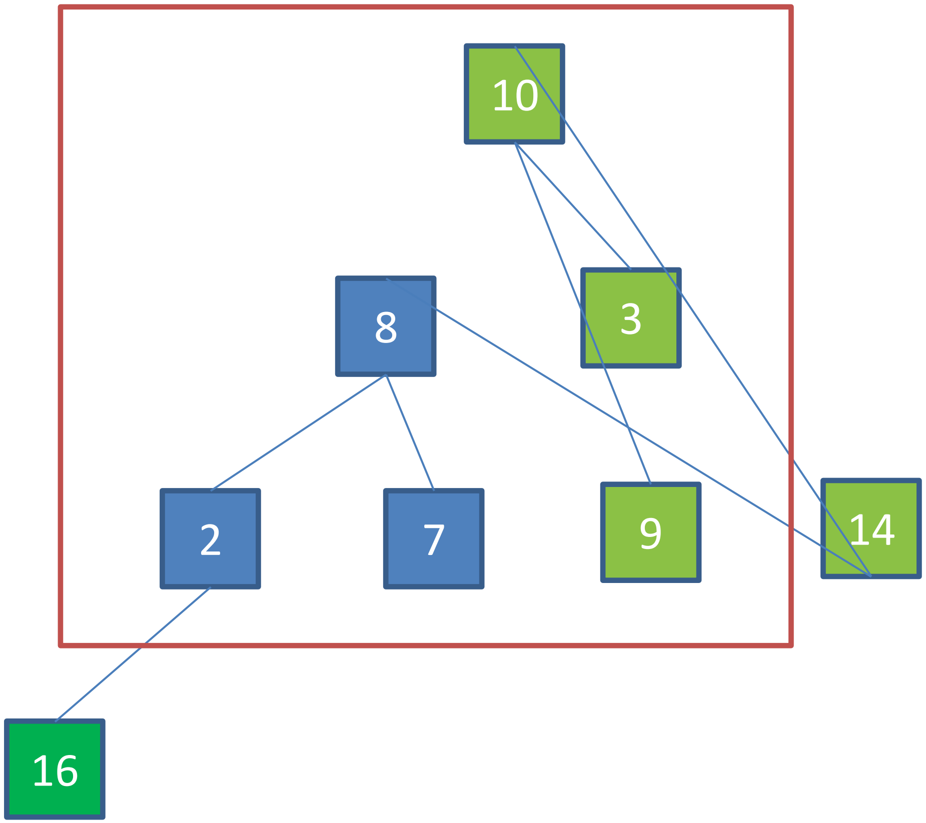
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

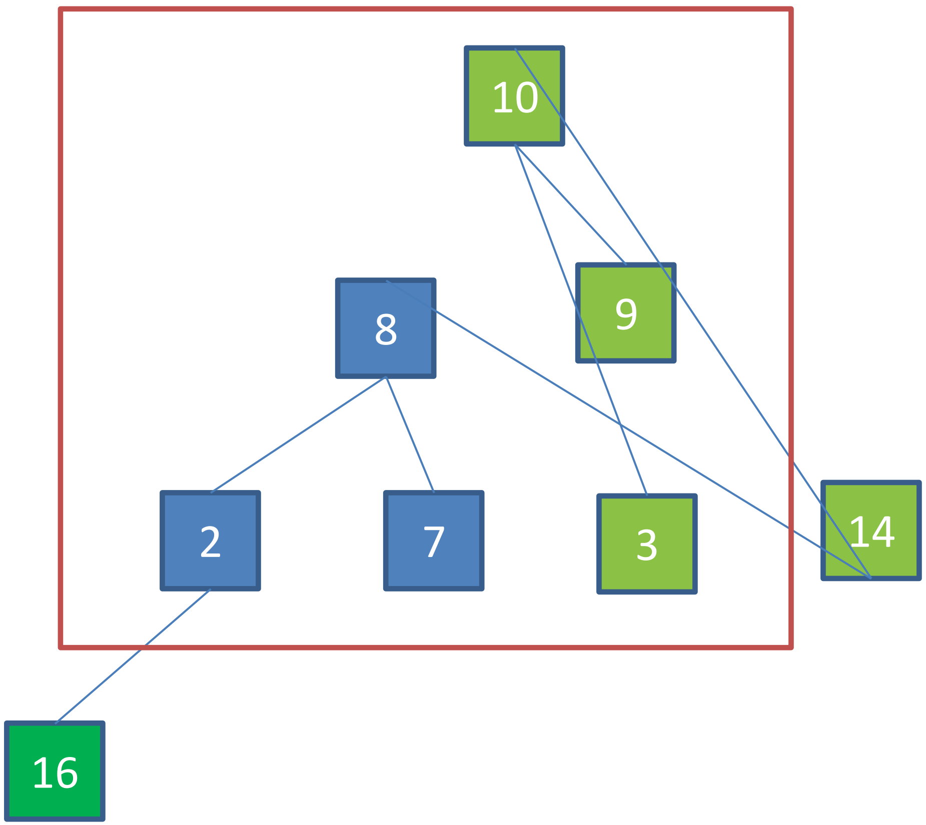
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

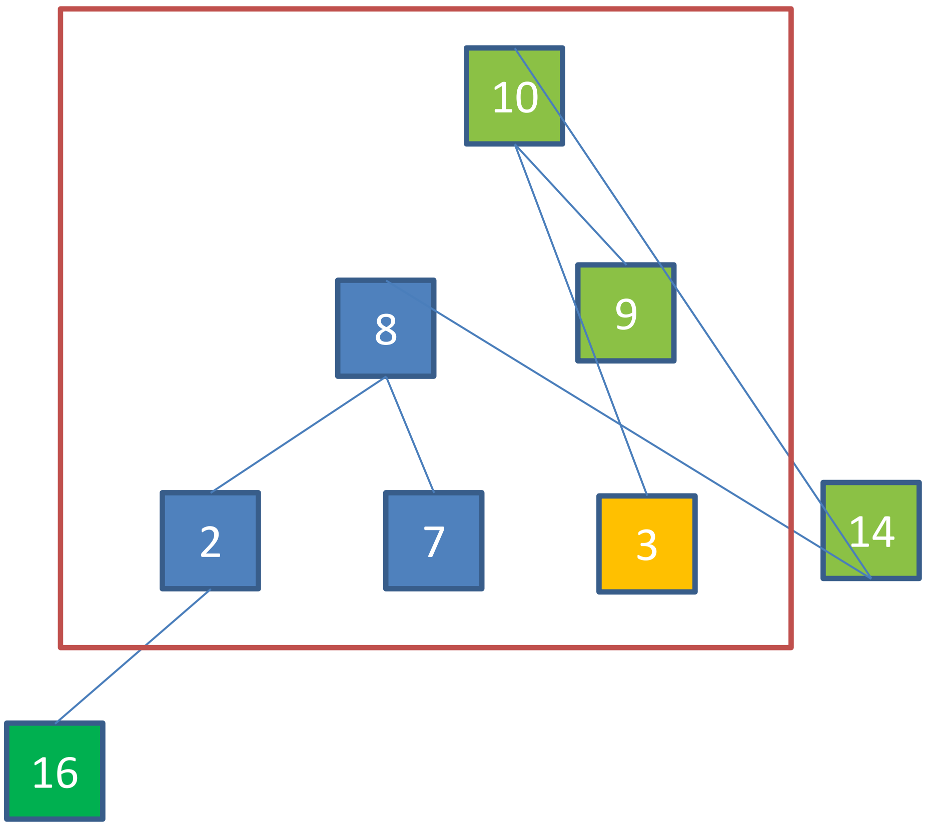
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

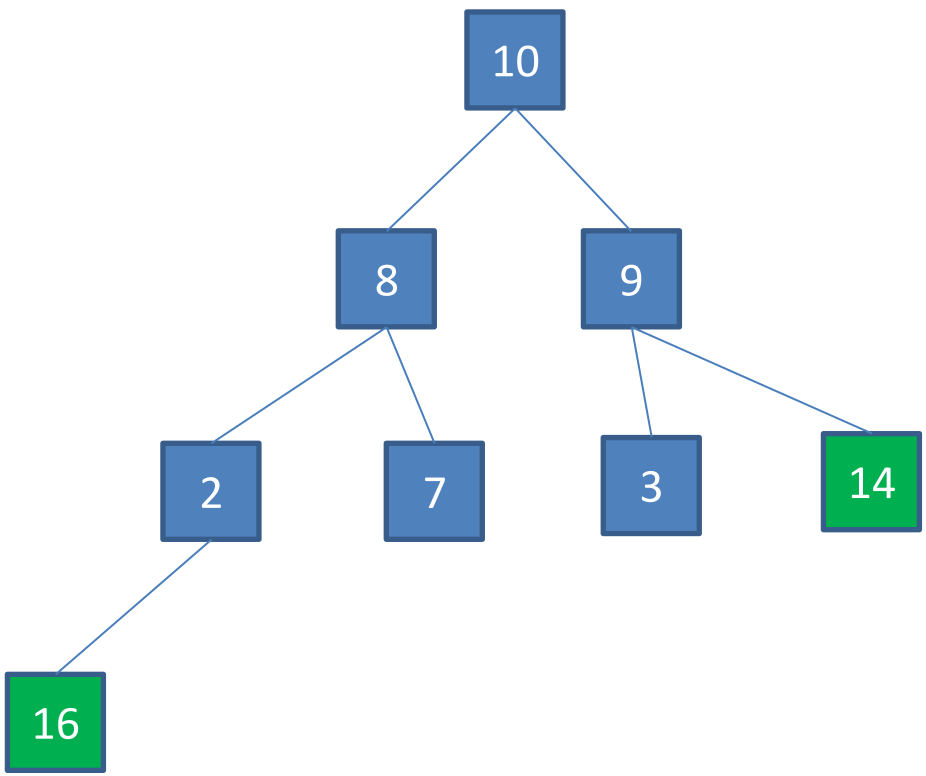
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

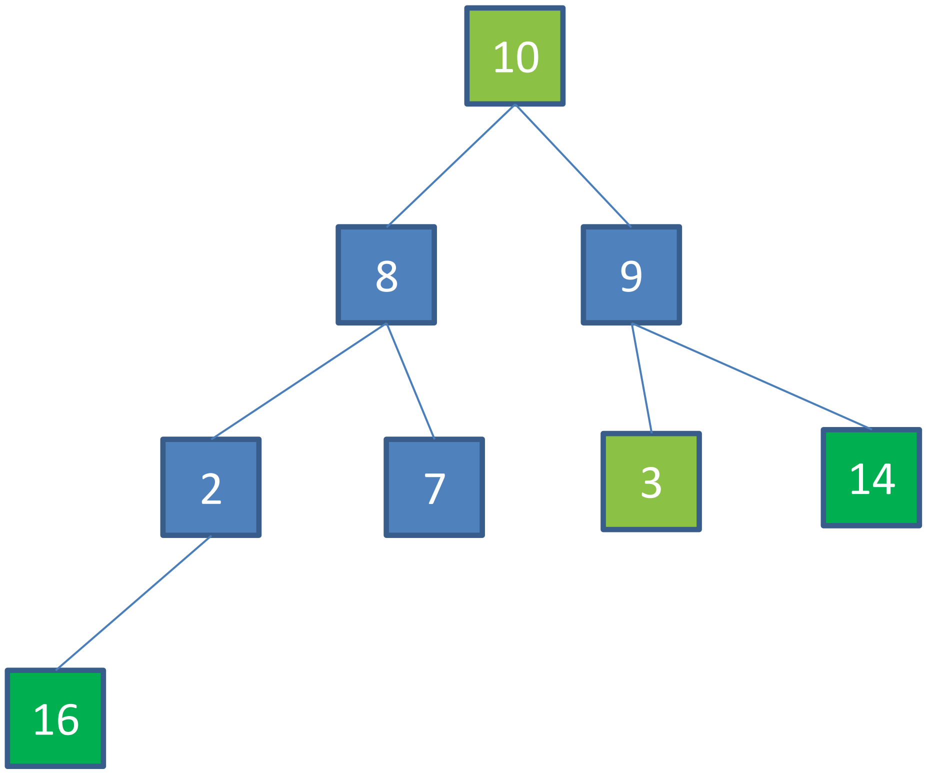
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A’s last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

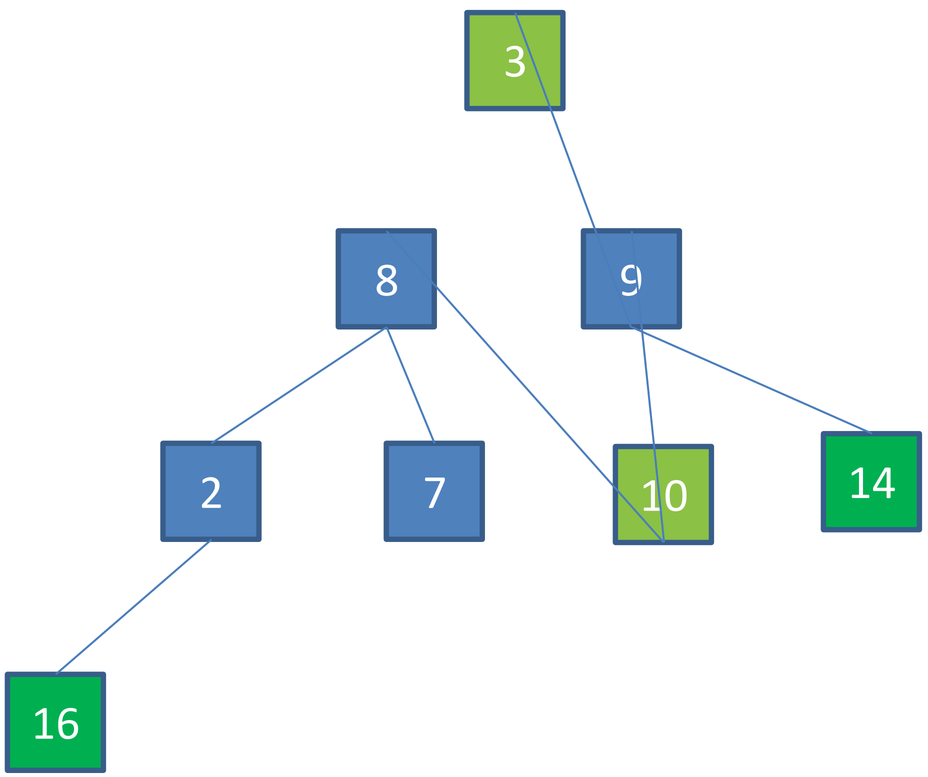
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

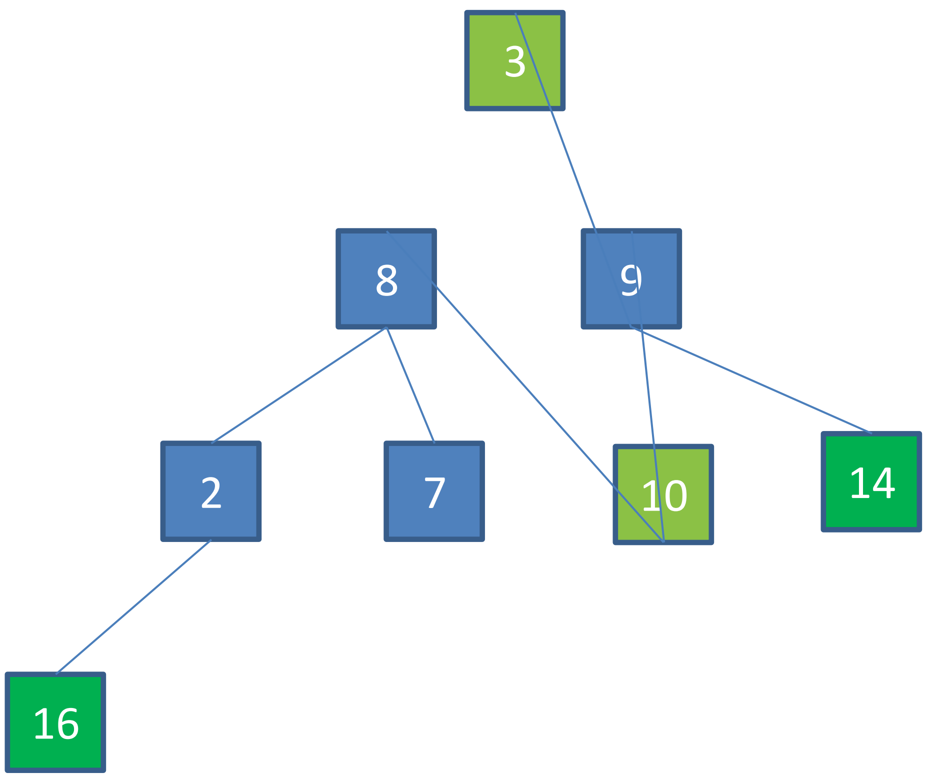
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

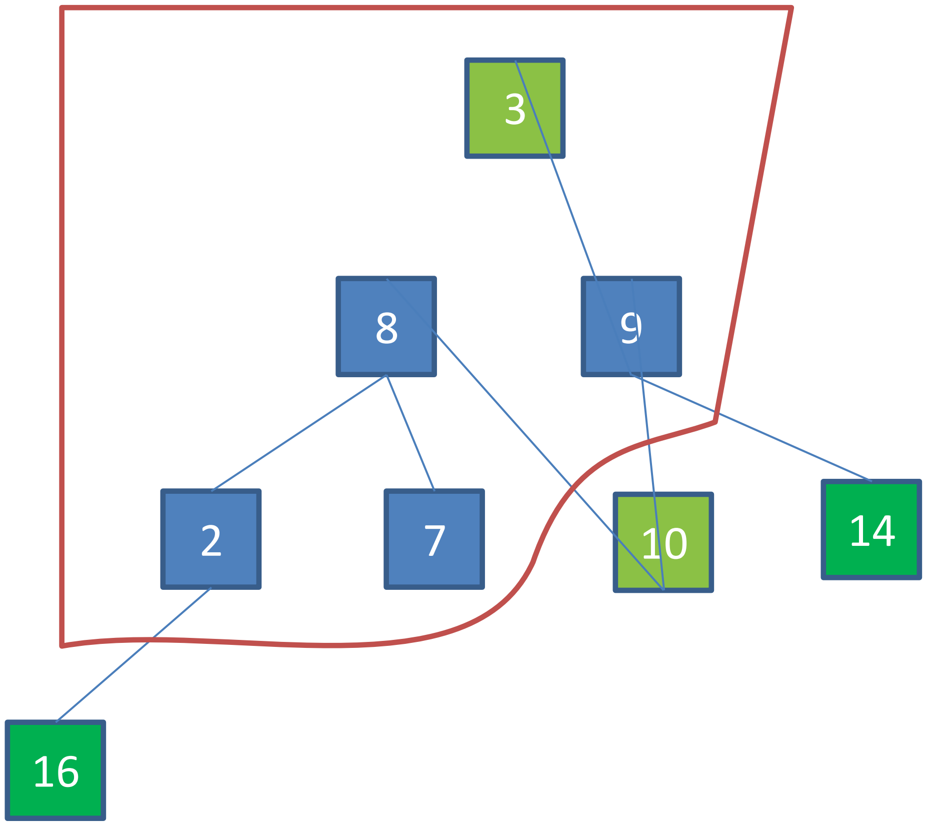
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

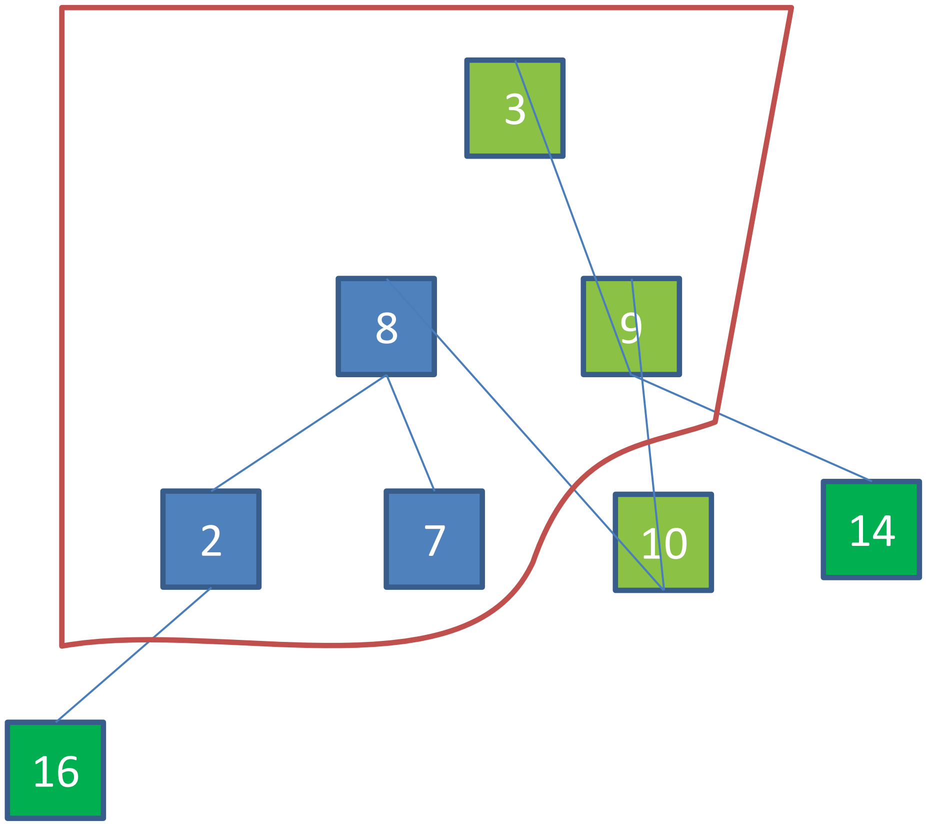
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

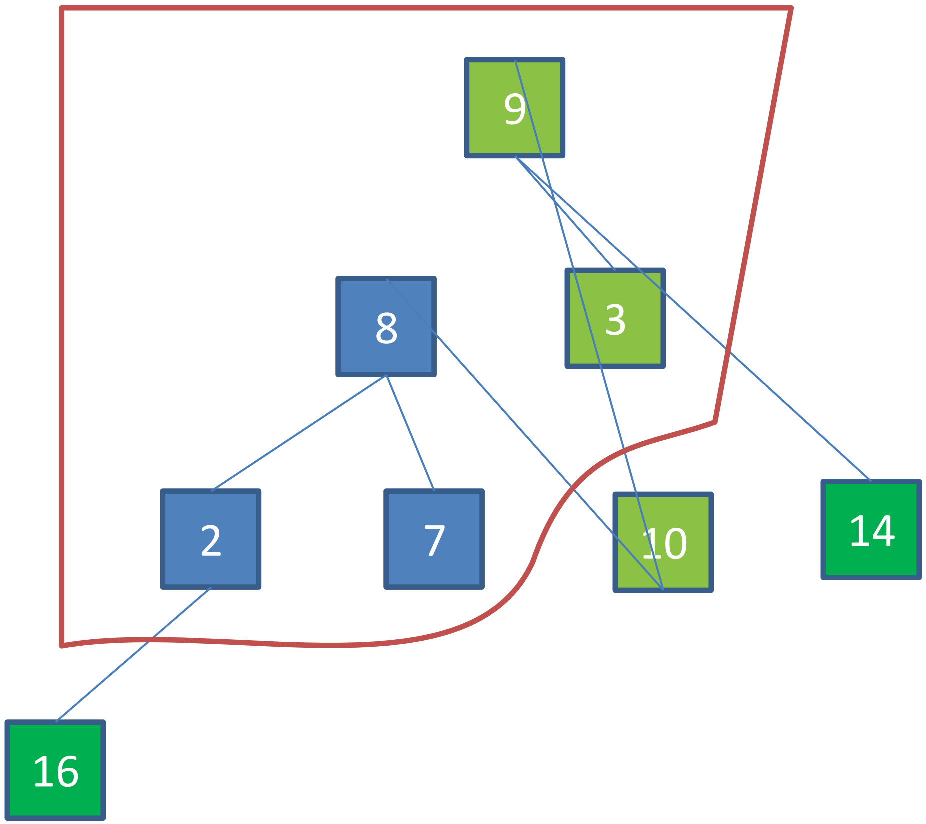
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

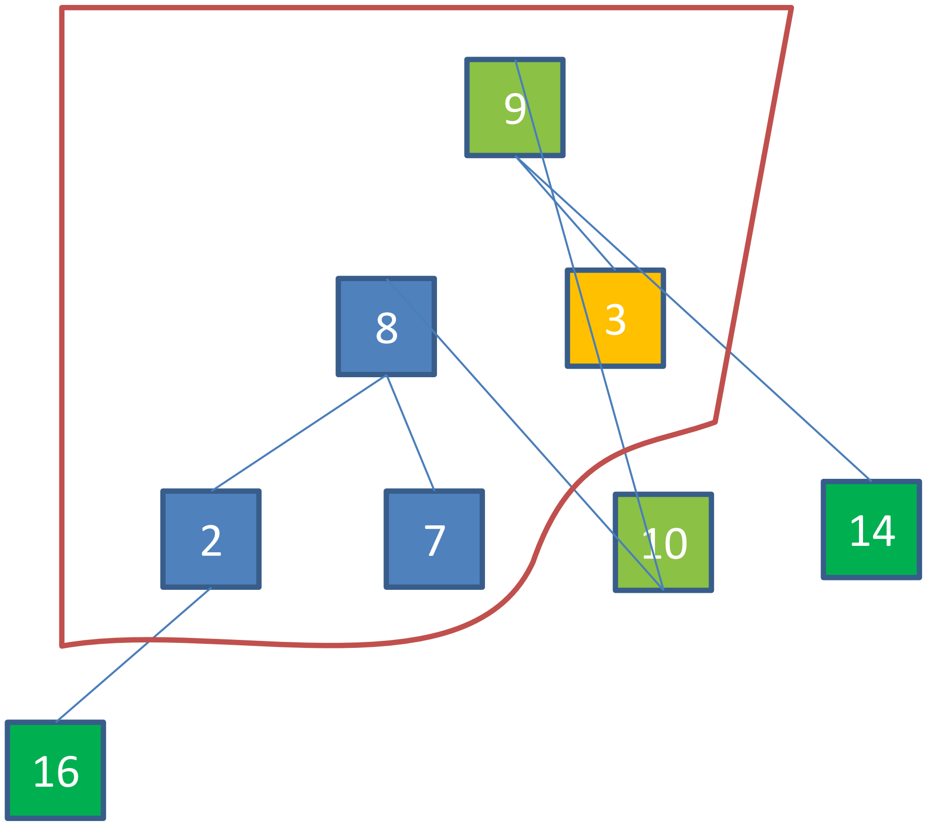
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

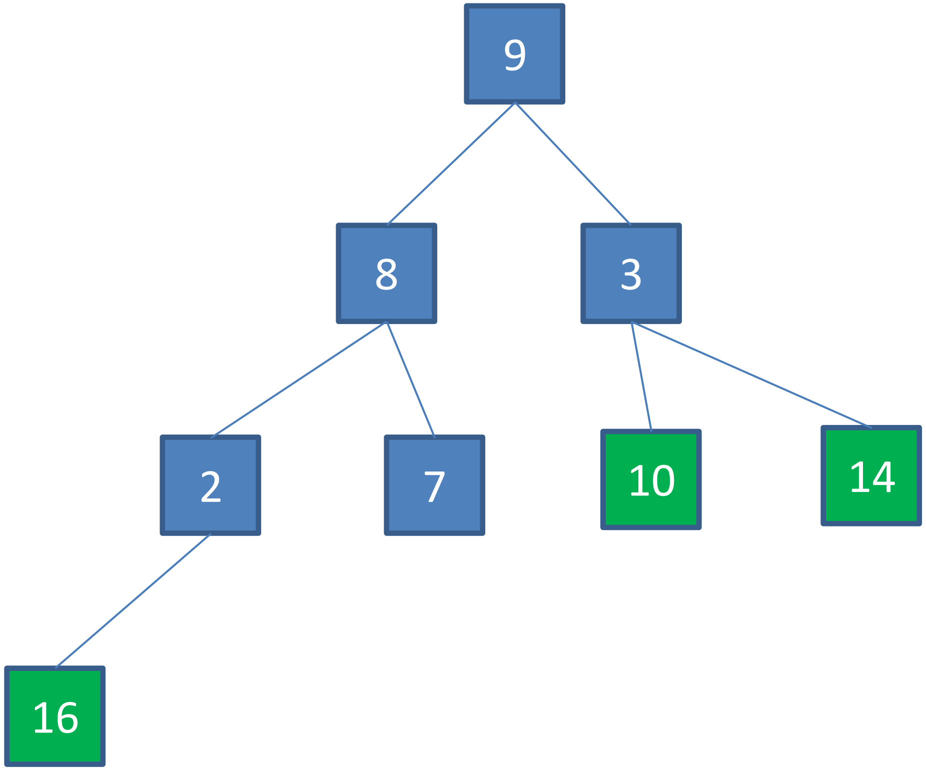
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

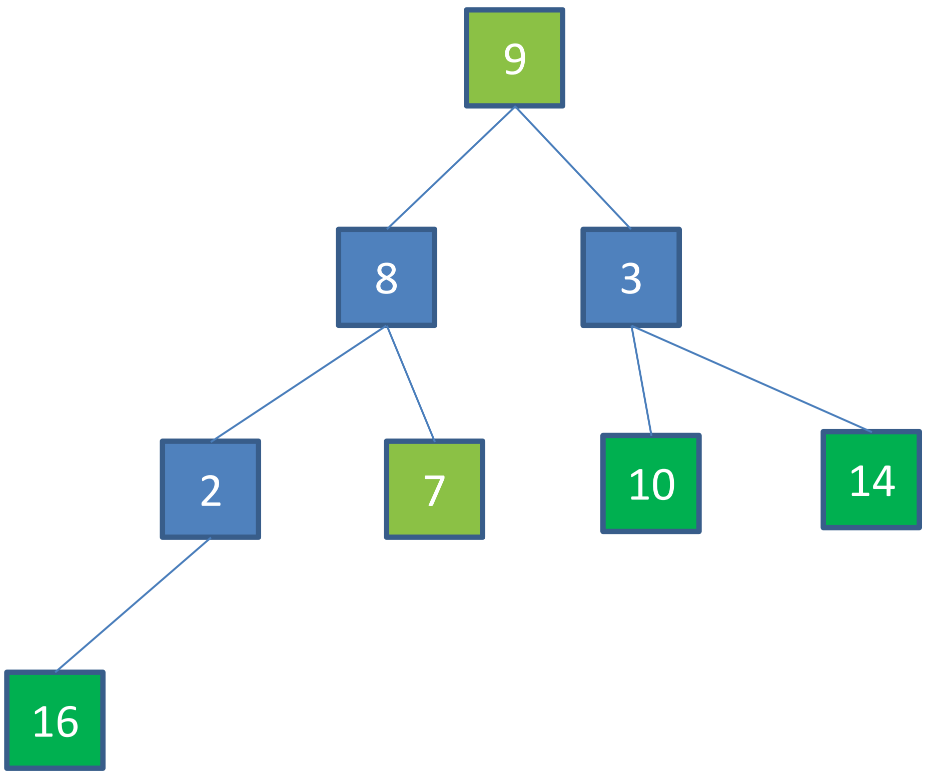
        exchange (i, root);
        i--;
        MAX-HEAPIFY(A, root, i);
    }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

        m = the index of the larger node
        Exchange i with the largest node
        MAX-HEAPIFY (A, m, t)
    }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

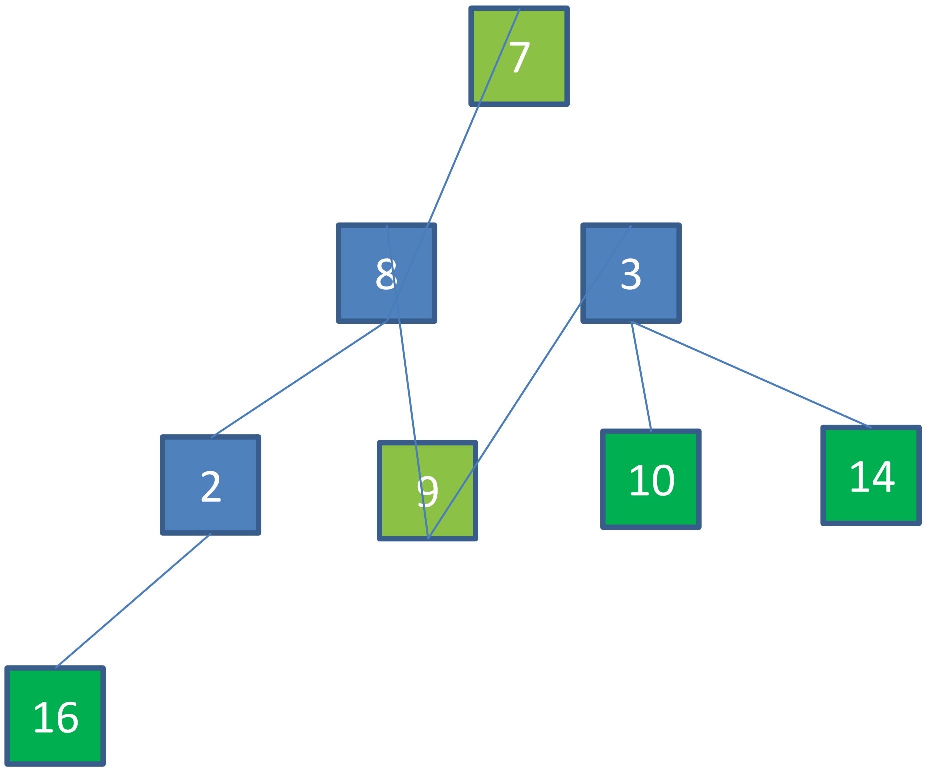
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

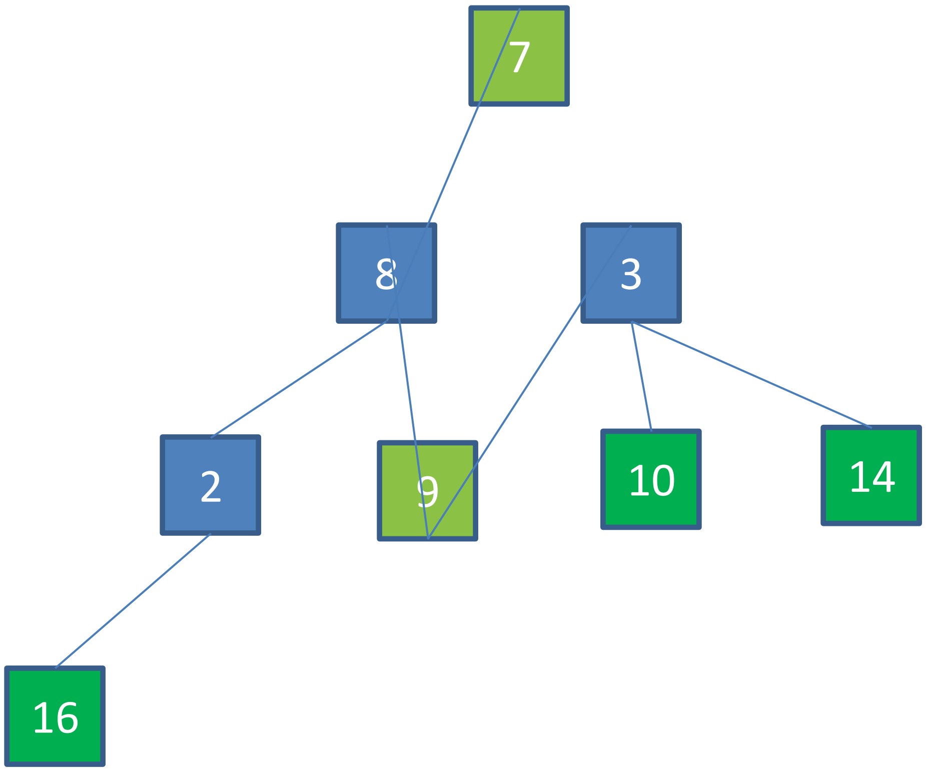
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

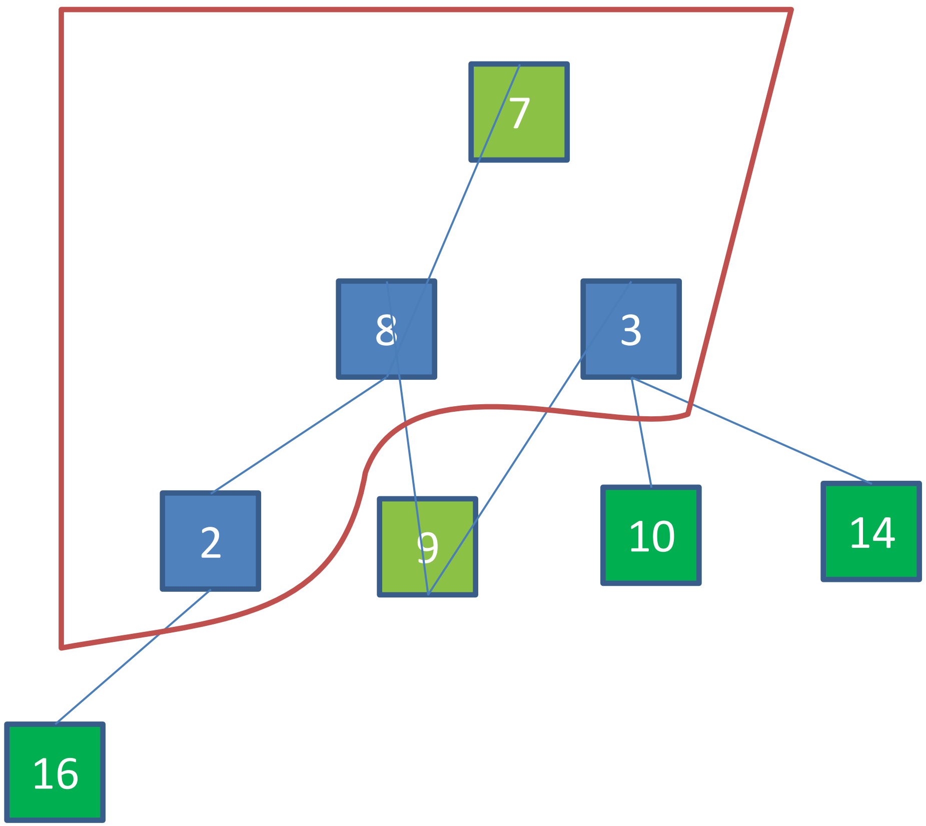
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

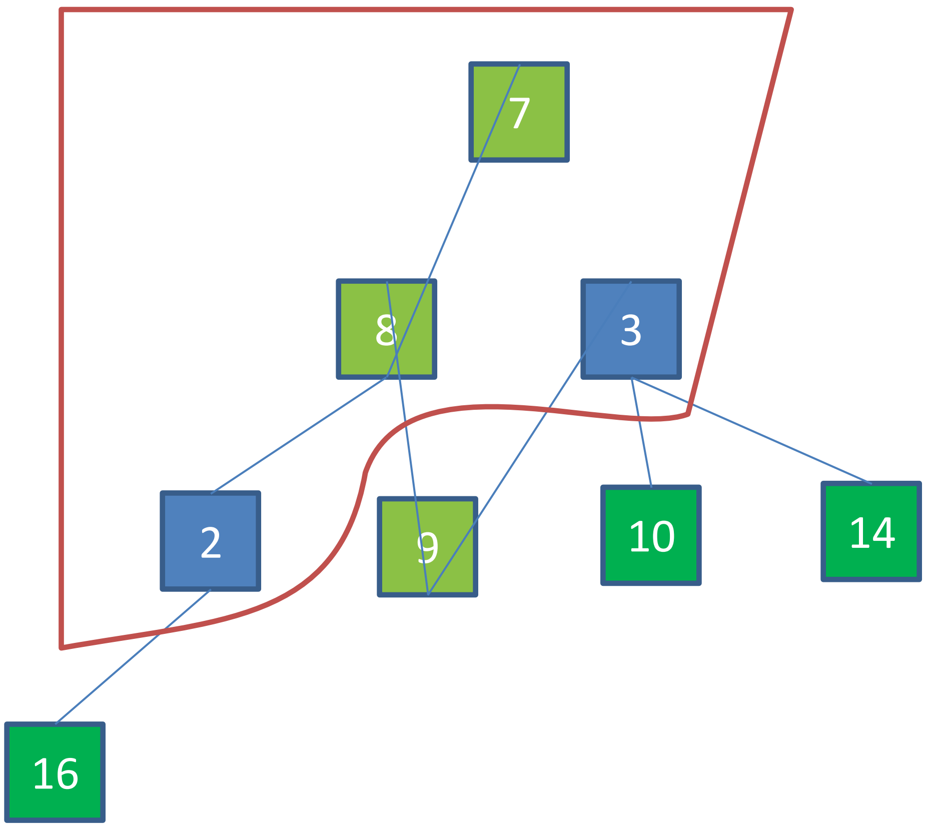
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

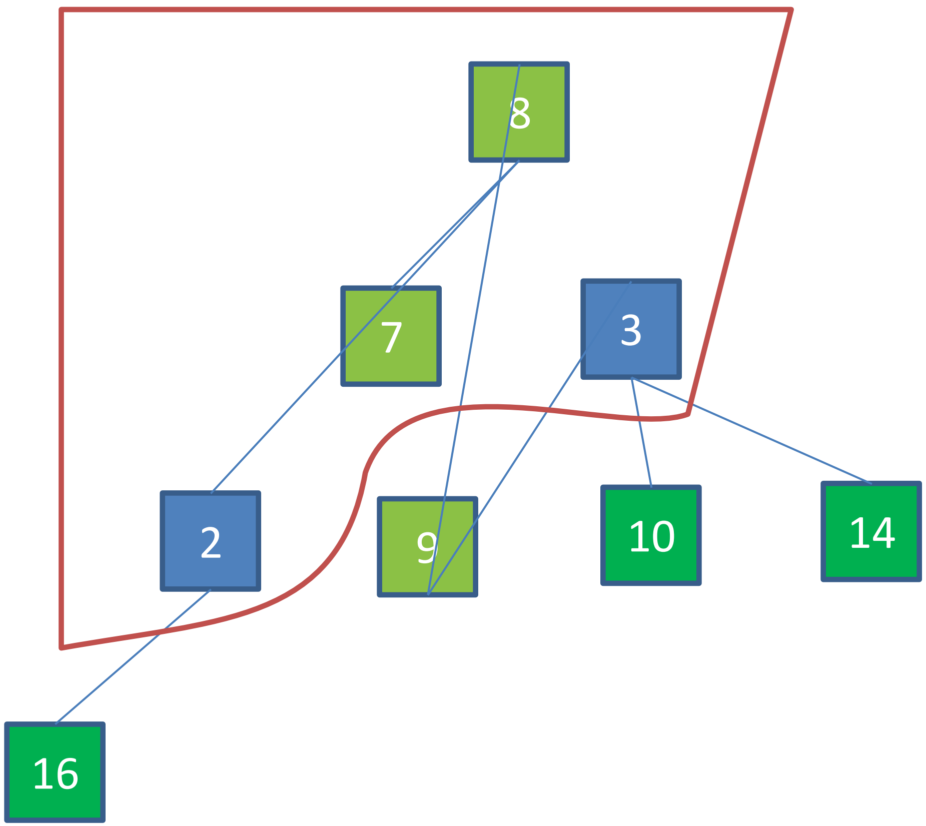
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

}

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

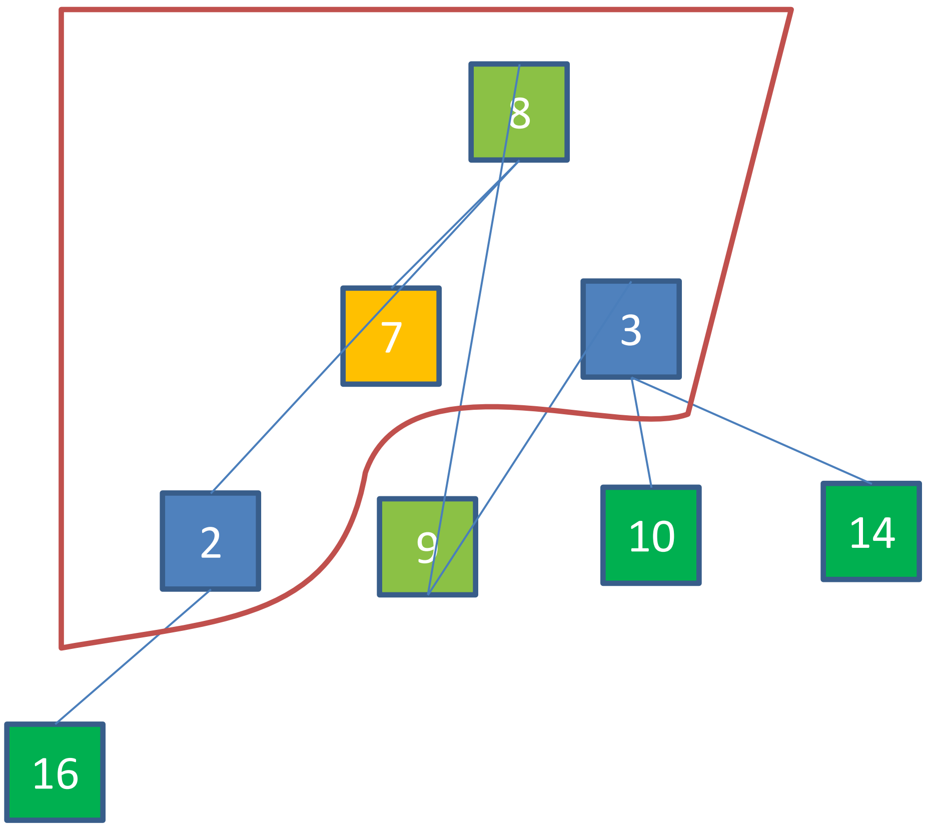
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

}

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

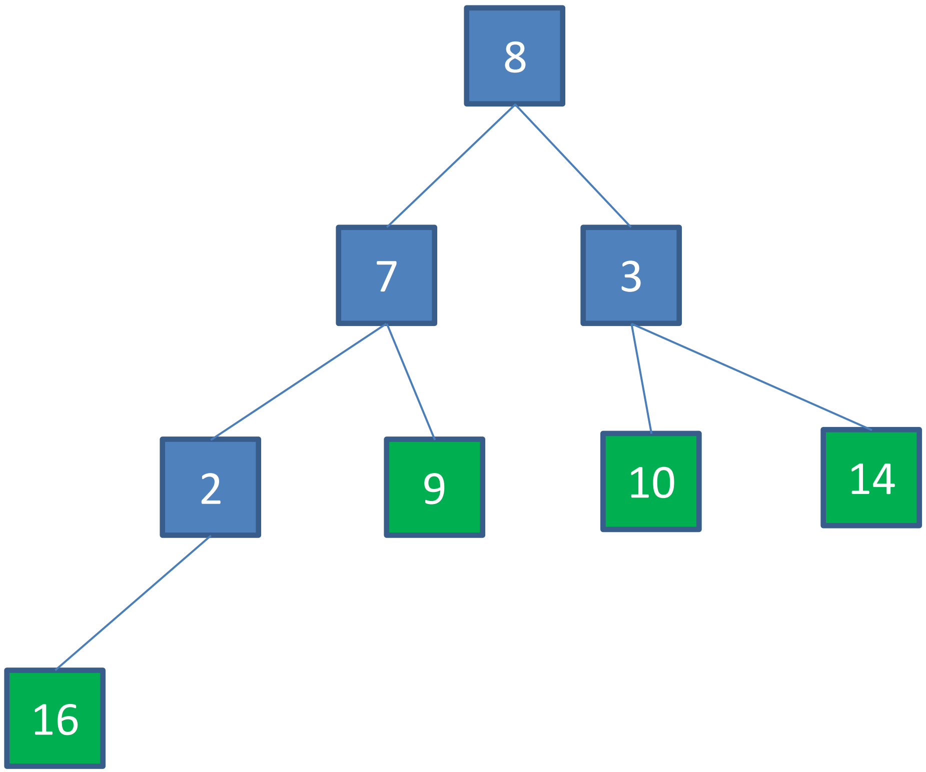
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

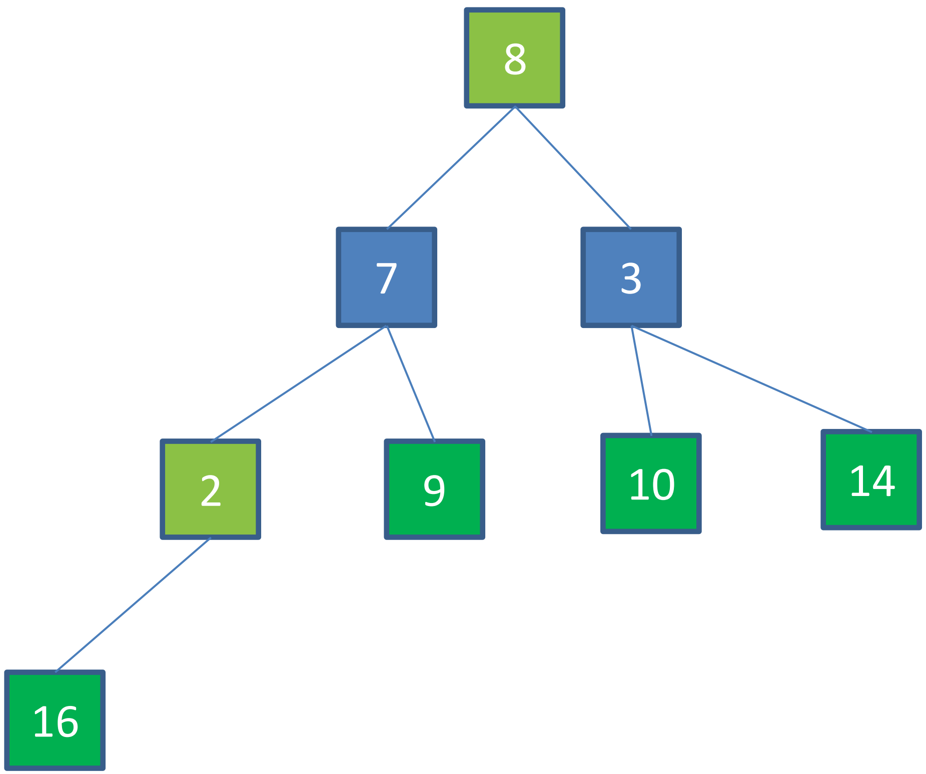
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

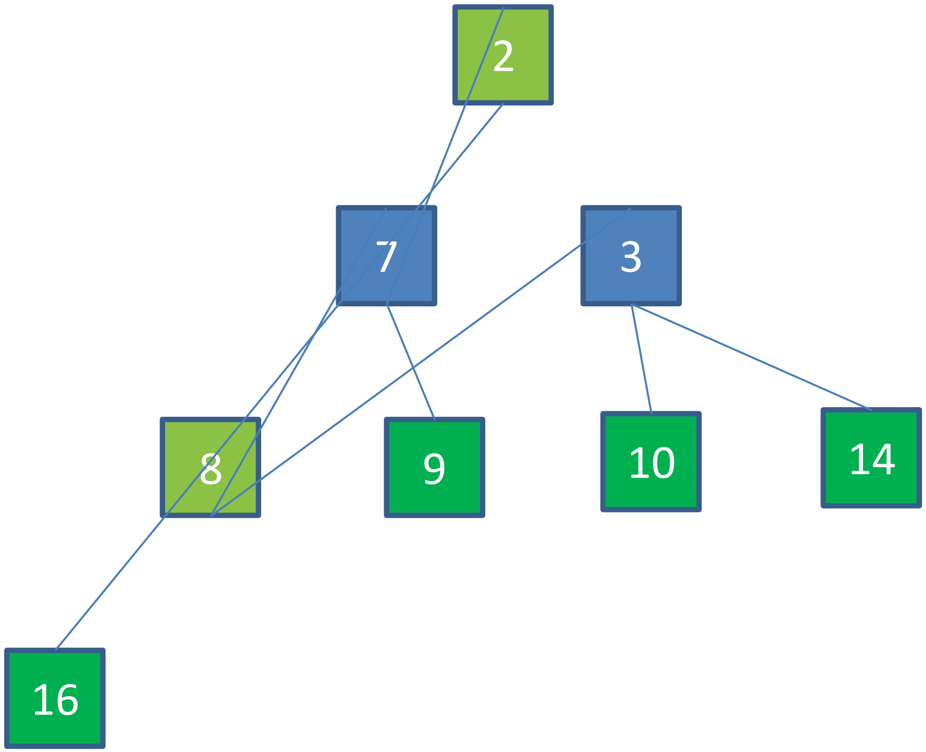
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

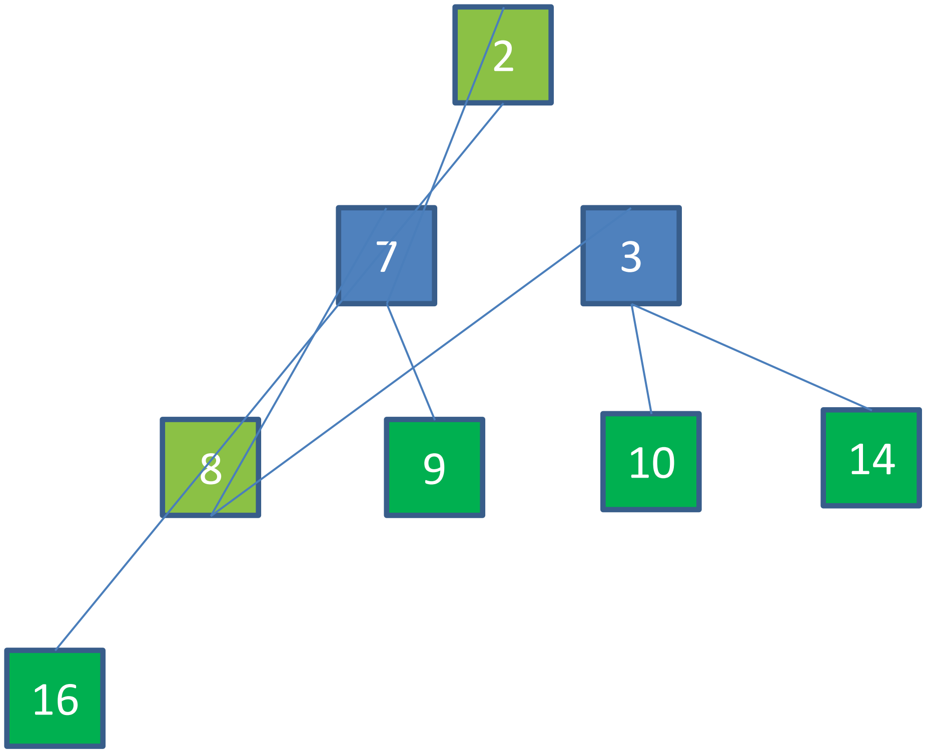
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

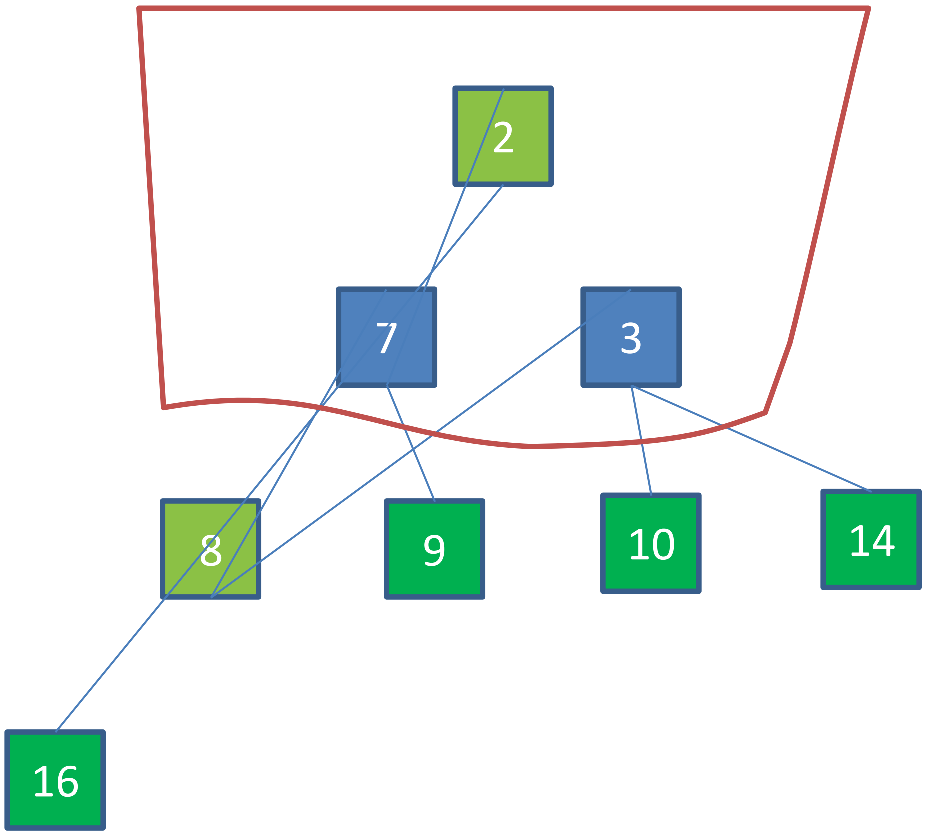
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

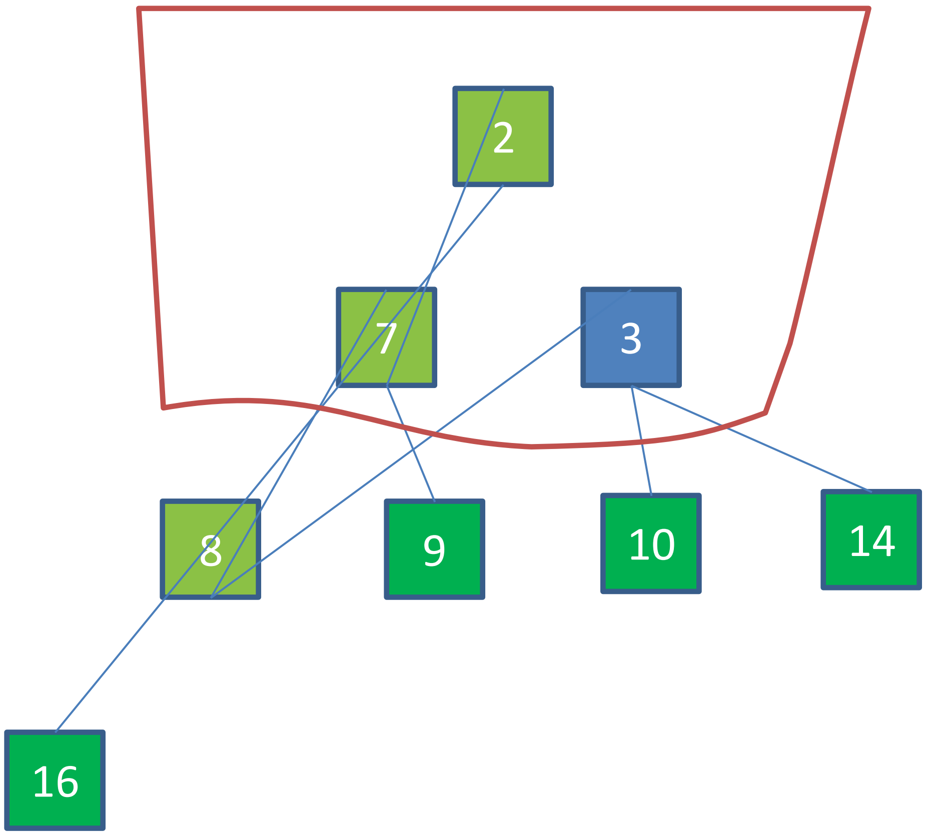
        exchange (i, root);
        i--;
        MAX-HEAPIFY(A, root, i);
    }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

        m = the index of the larger node
        Exchange i with the largest node
        MAX-HEAPIFY (A, m, t)
    }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

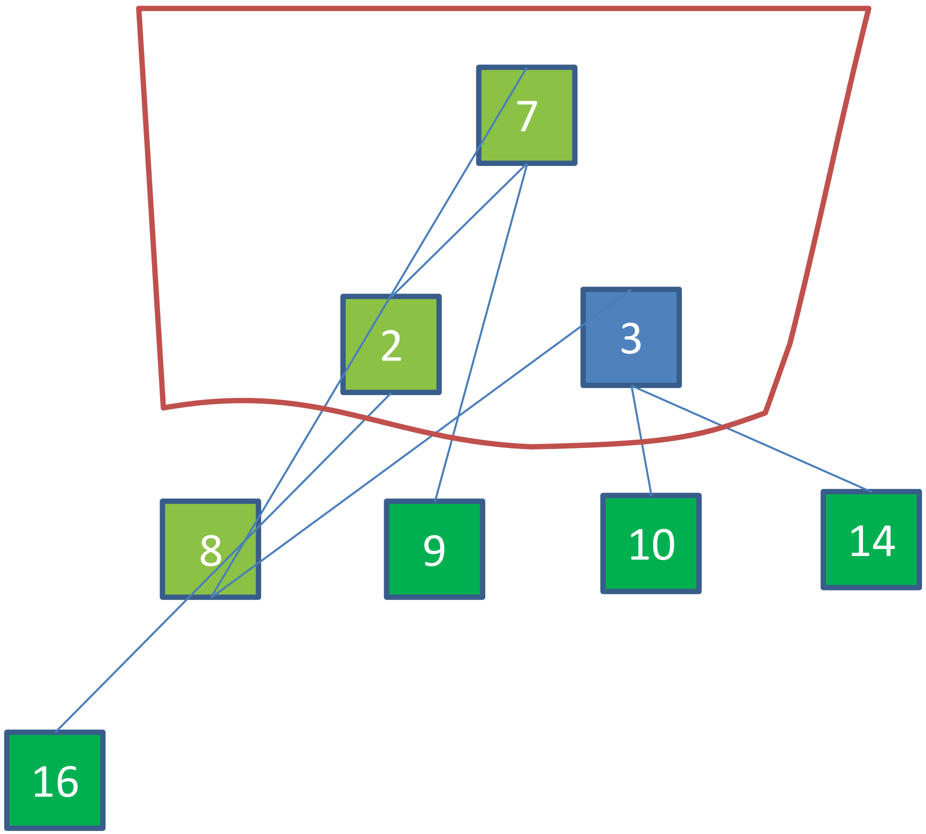
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

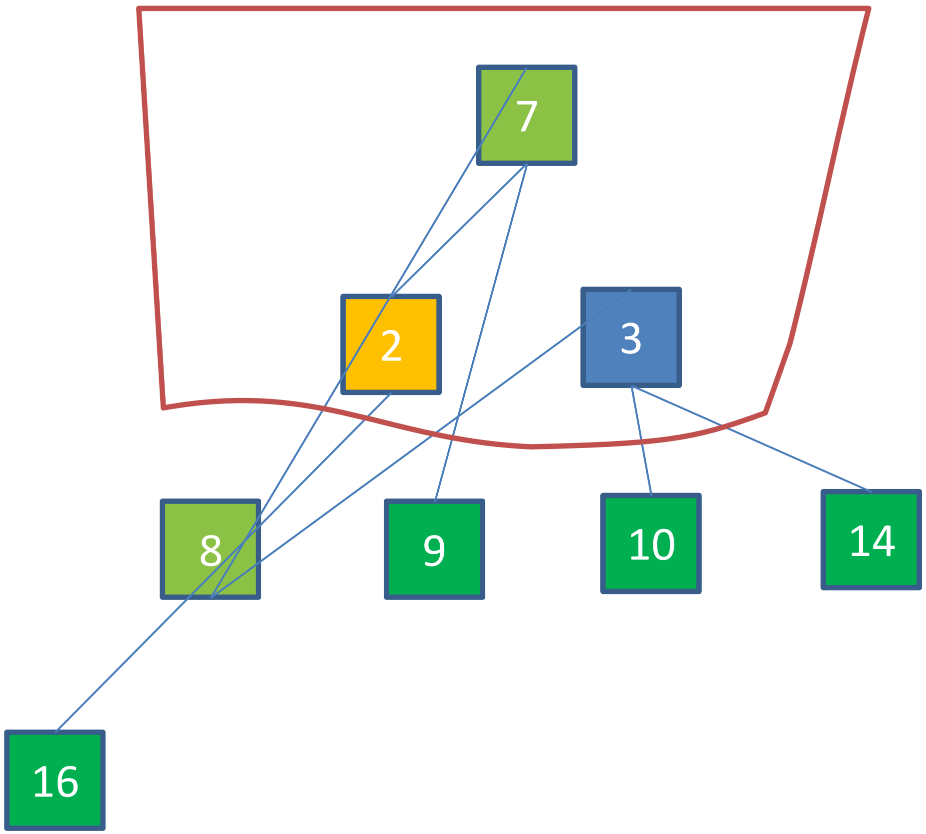
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

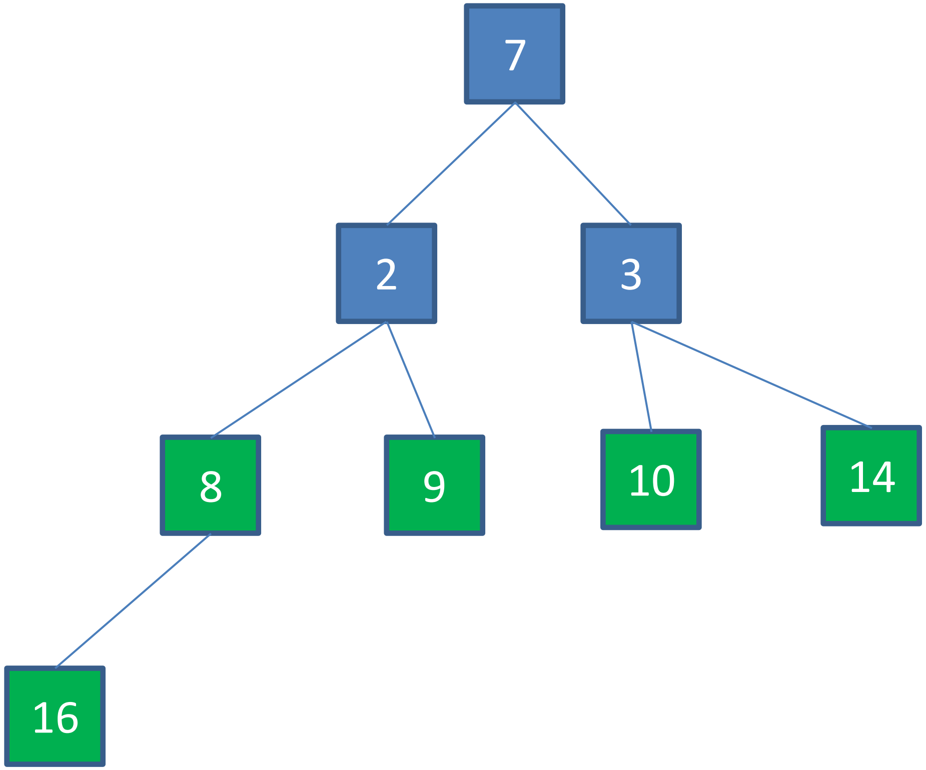
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

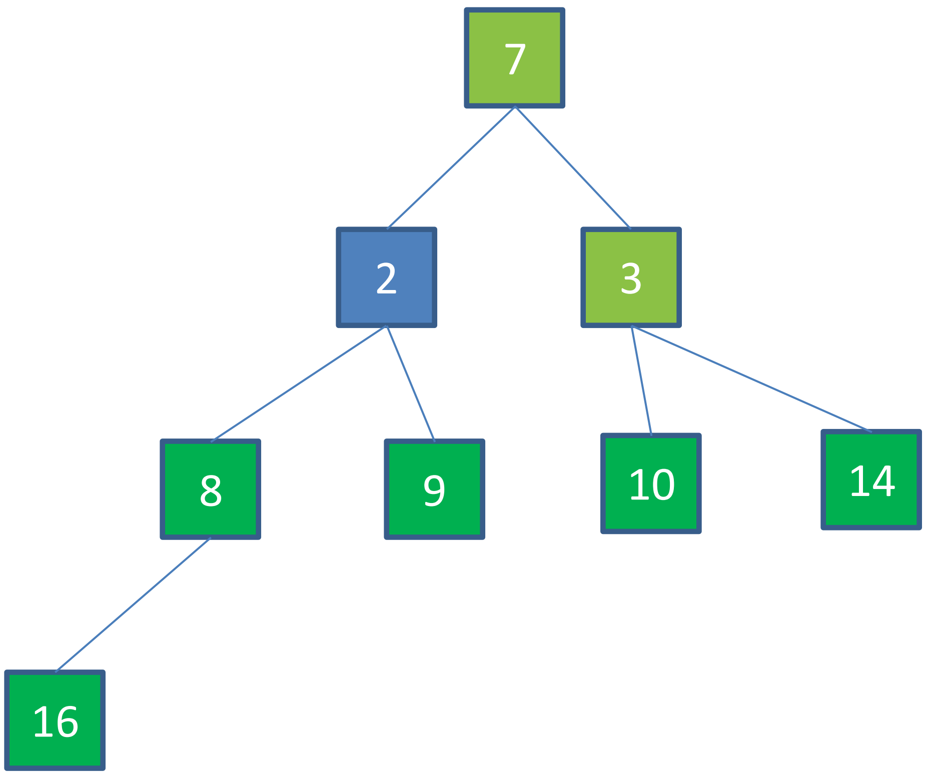
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

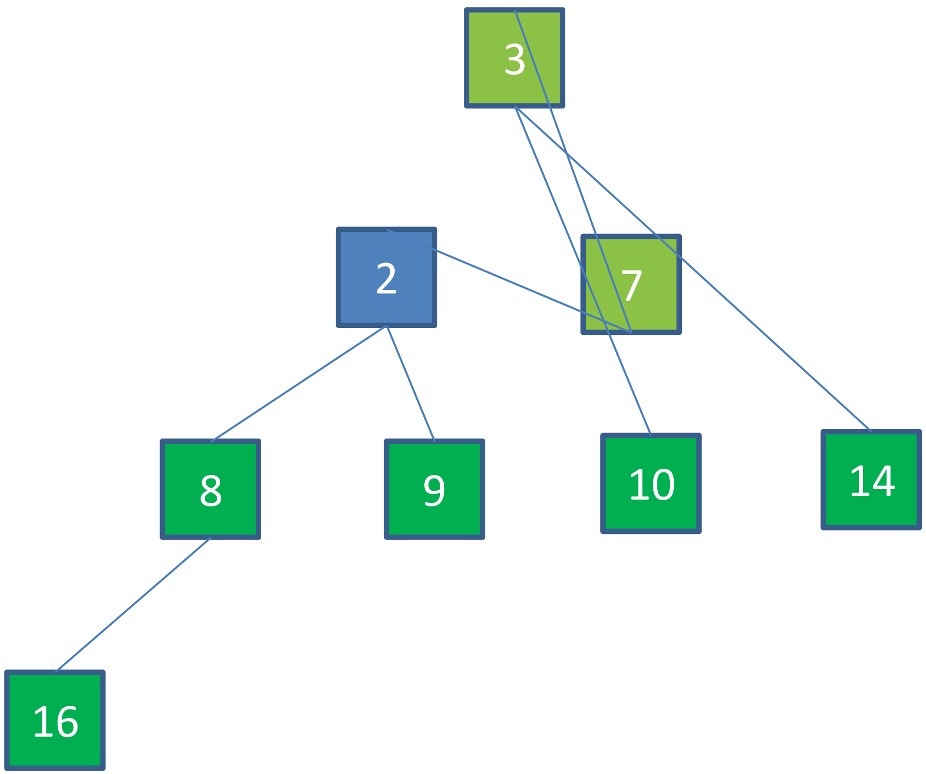
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

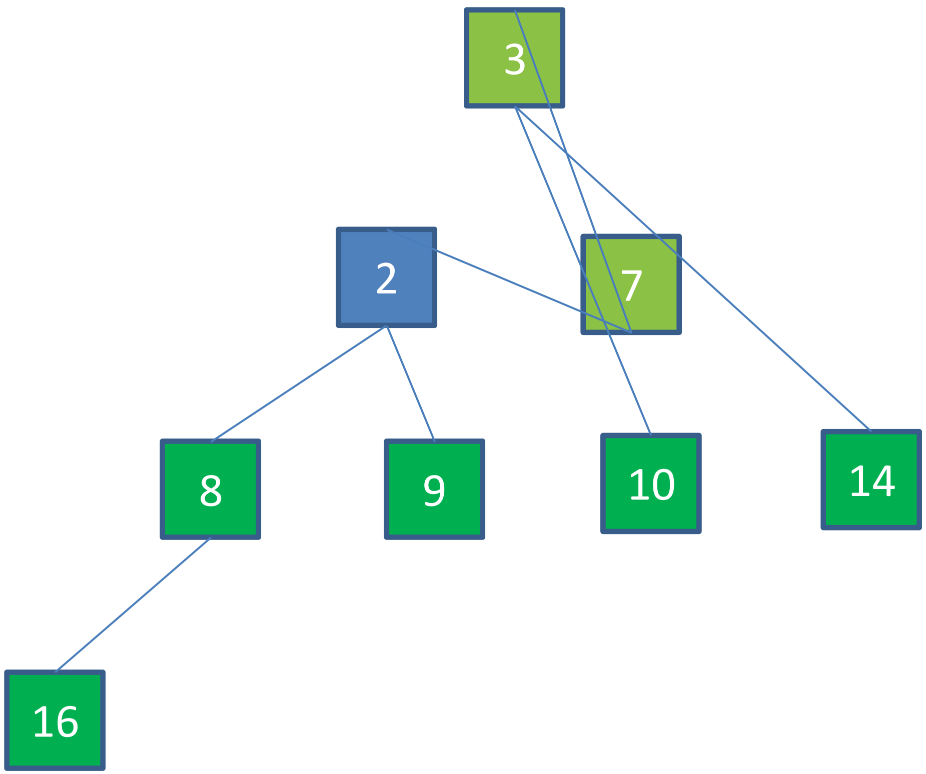
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

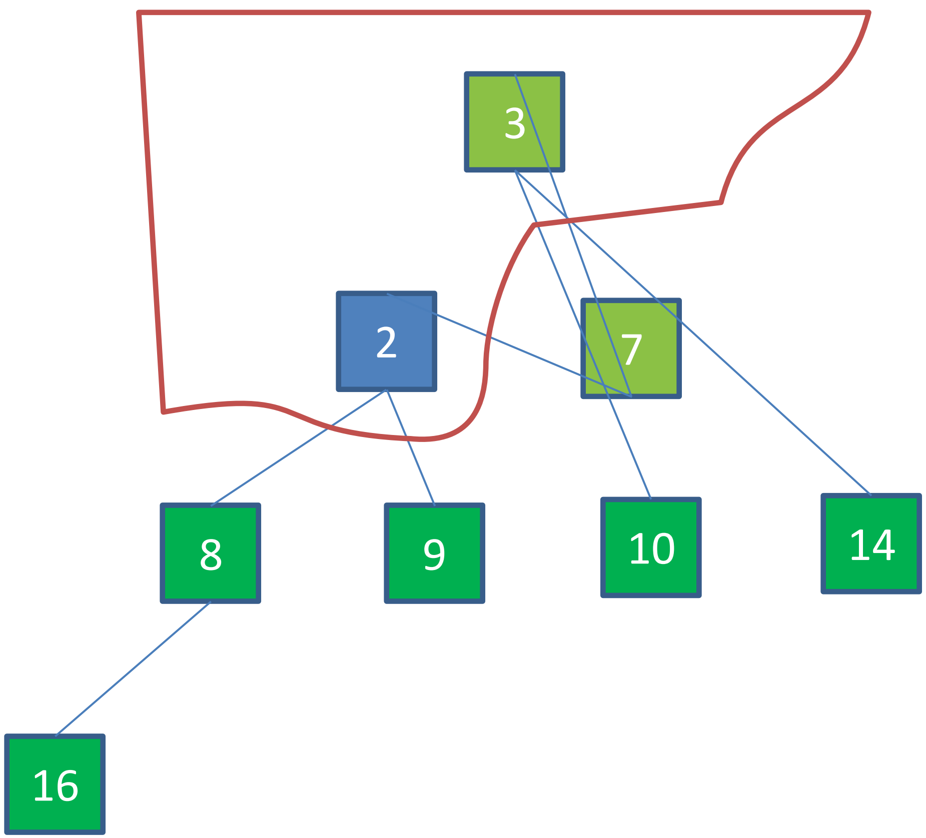
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

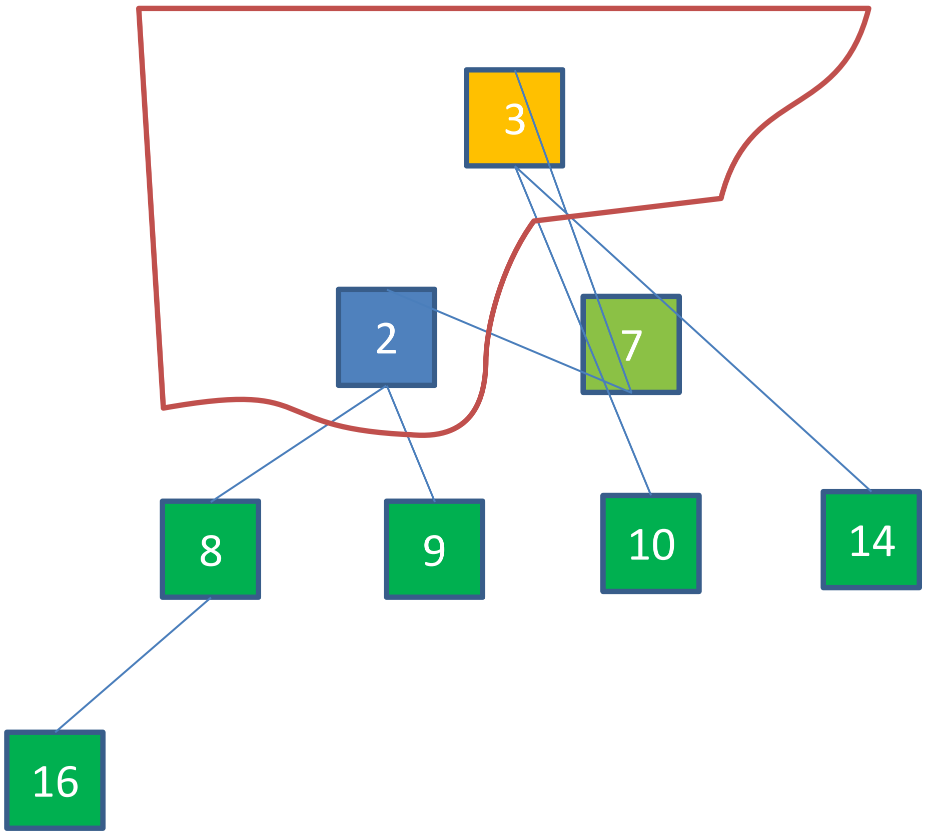
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

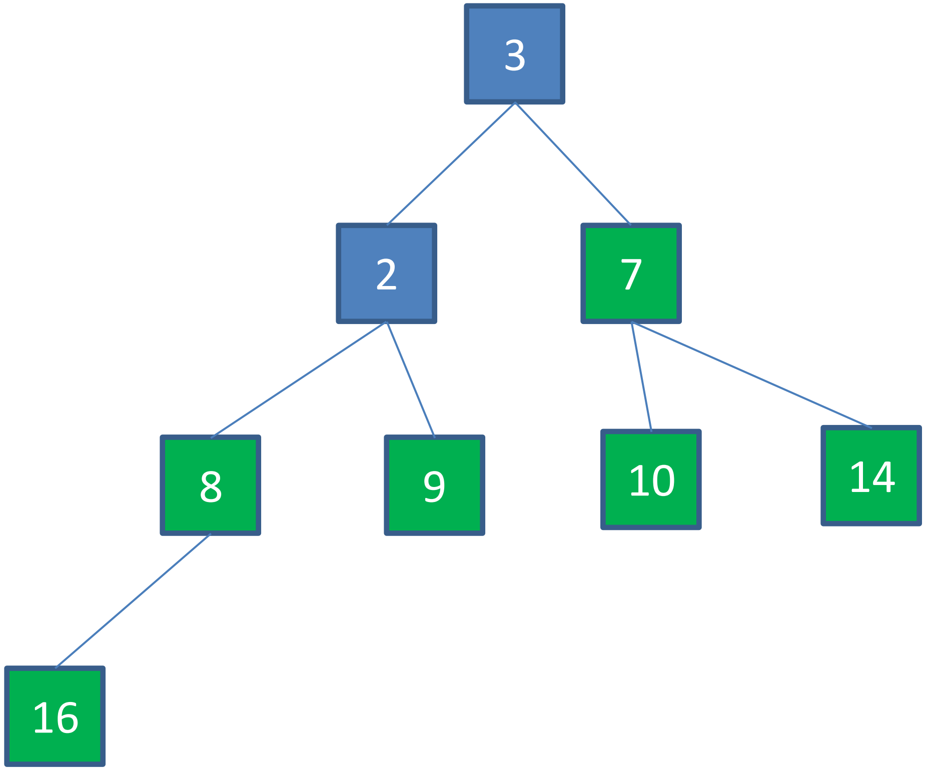
        exchange (i, root);
        i--;
        MAX-HEAPIFY(A, root, i);
    }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

        m = the index of the larger node
        Exchange i with the largest node
        MAX-HEAPIFY (A, m, t)
    }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

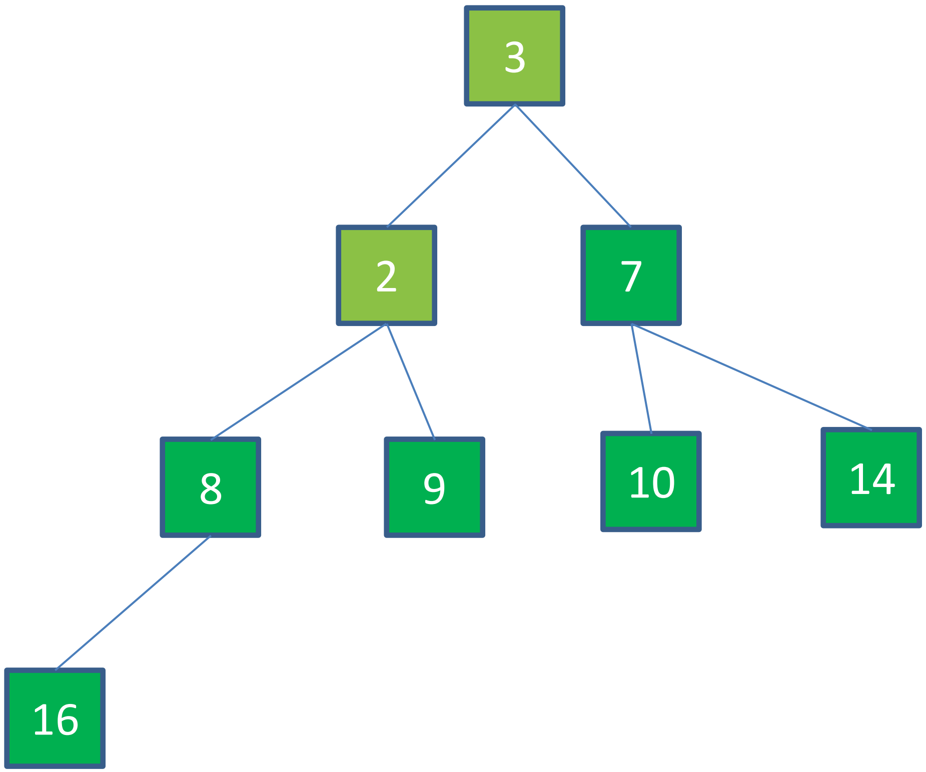
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

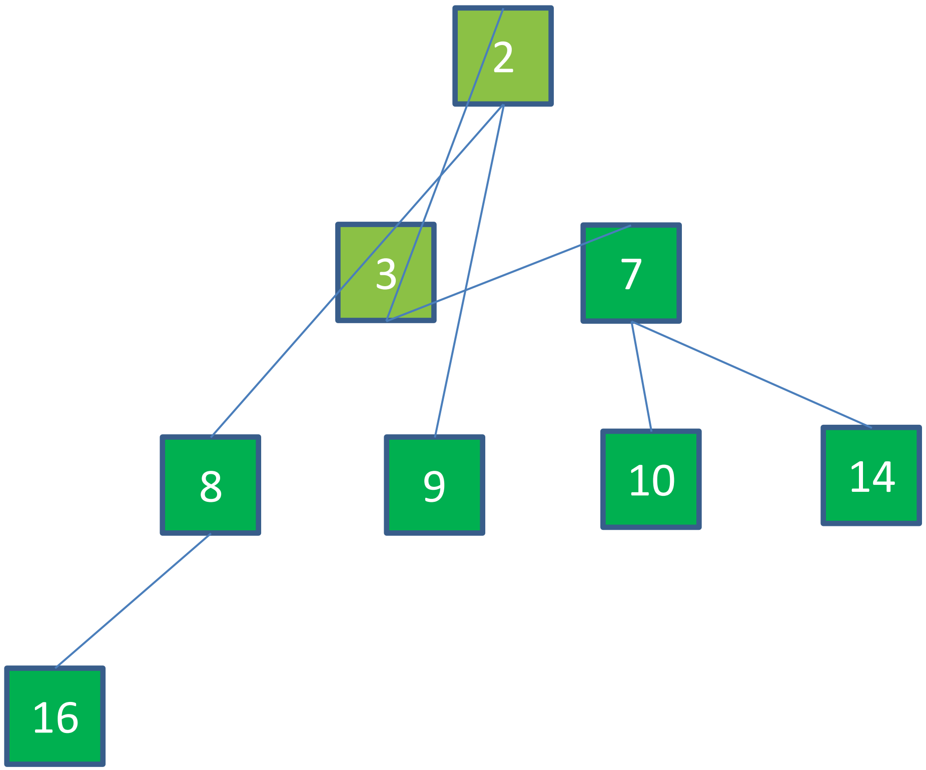
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

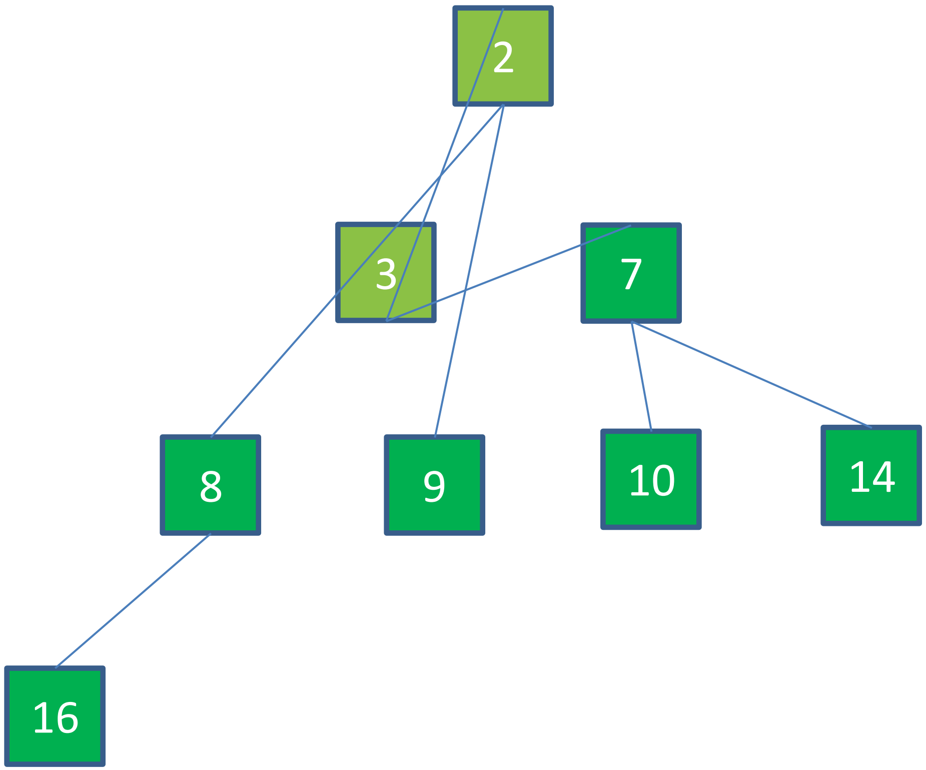
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

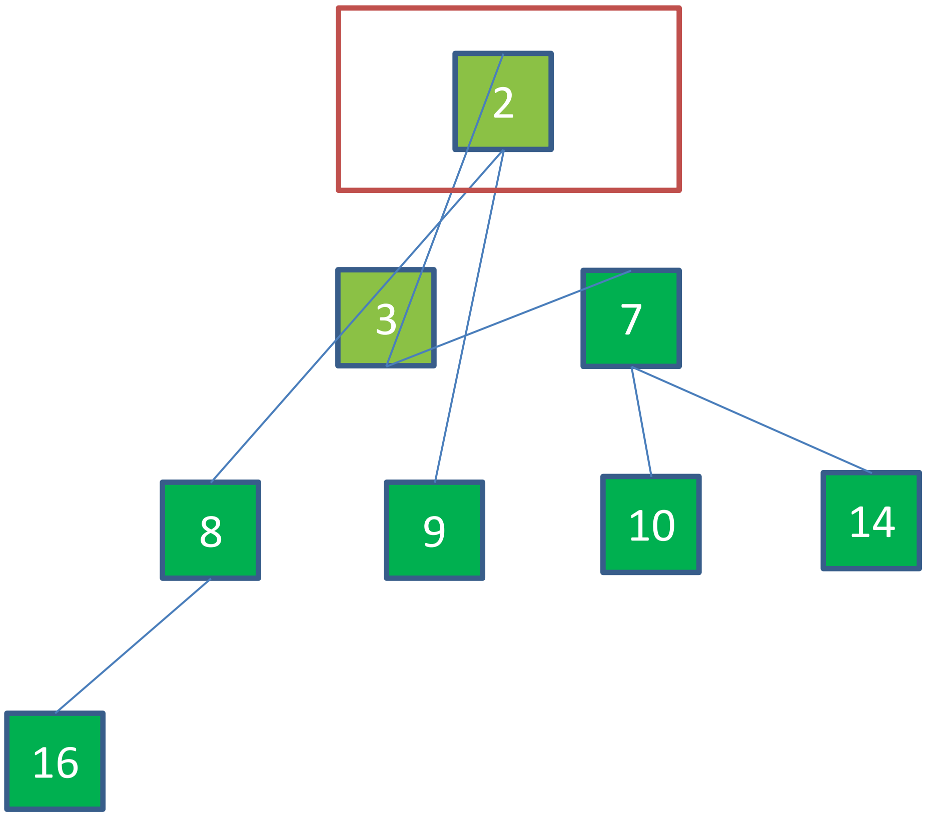
   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

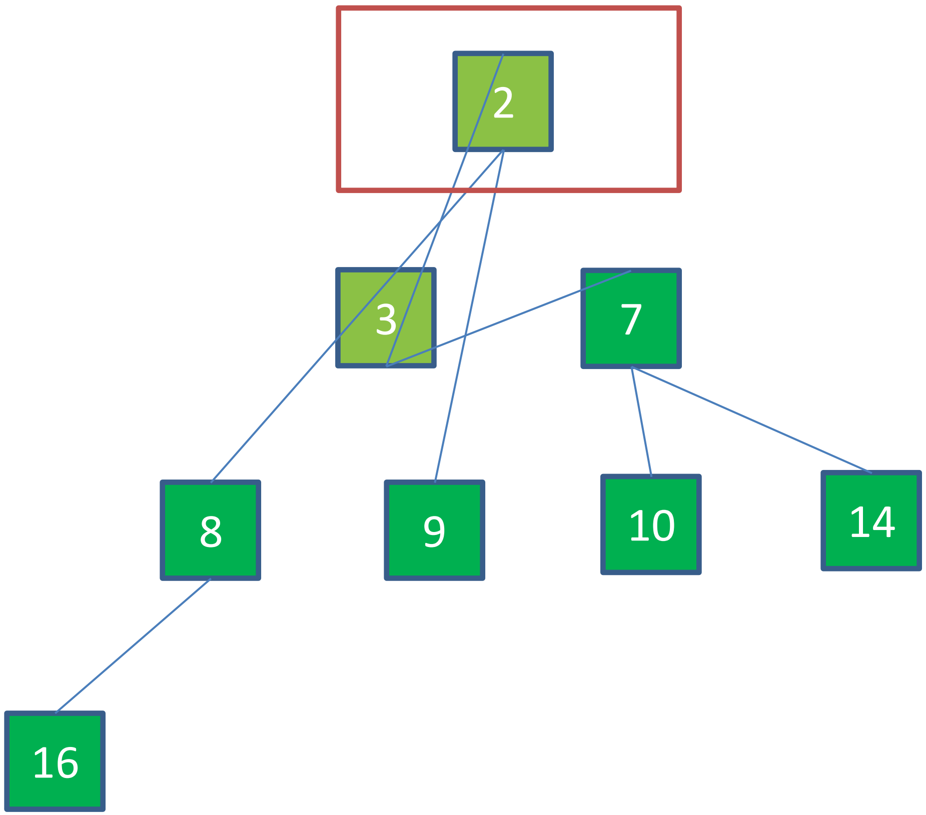
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

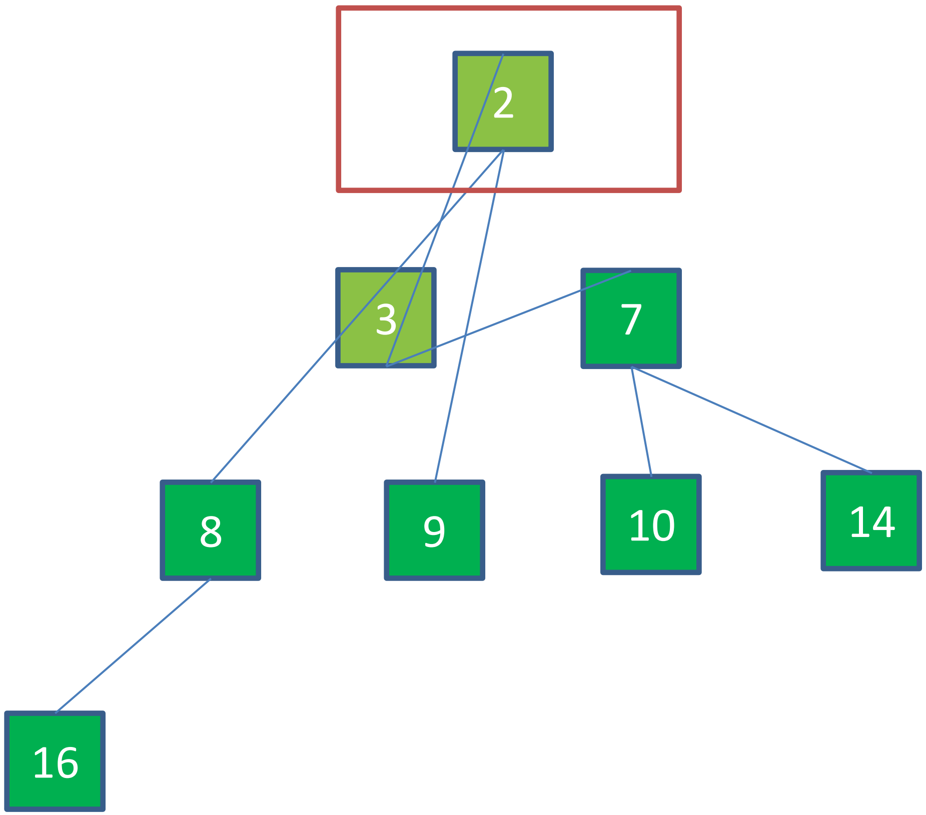
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

    exchange (i, root);

    i--;

    MAX-HEAPIFY(A, root, i);

  }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

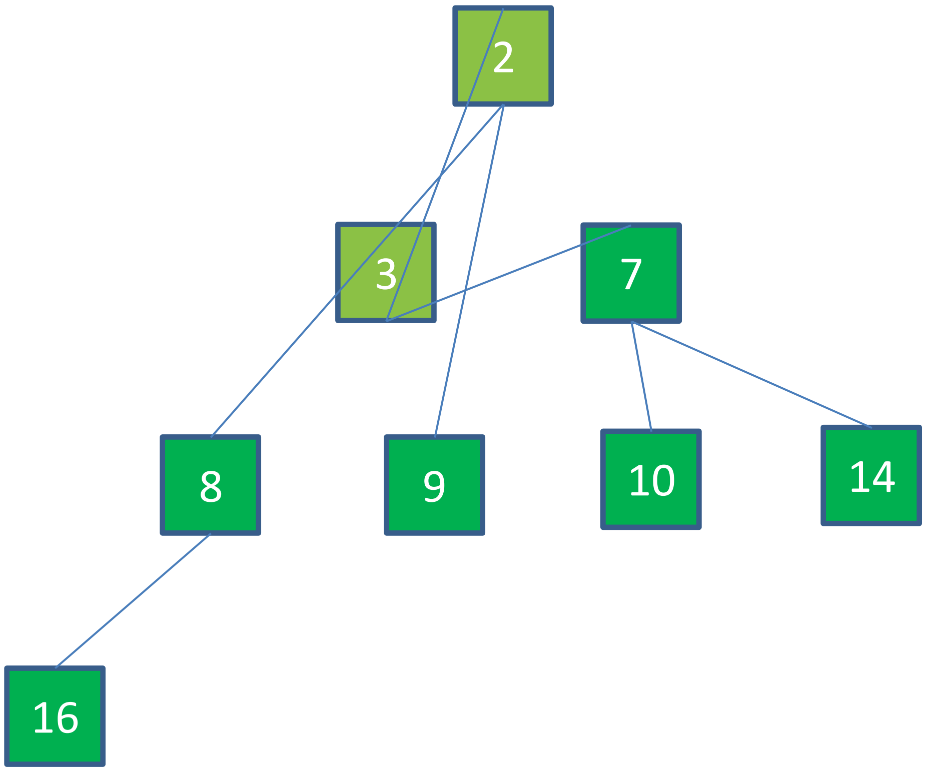
    m = the index of the larger node

    Exchange i with the largest node

    MAX-HEAPIFY (A, m, t)

  }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {
   - exchange (i, root);
   - i--;
   - **MAX-HEAPIFY**(A, root, i);

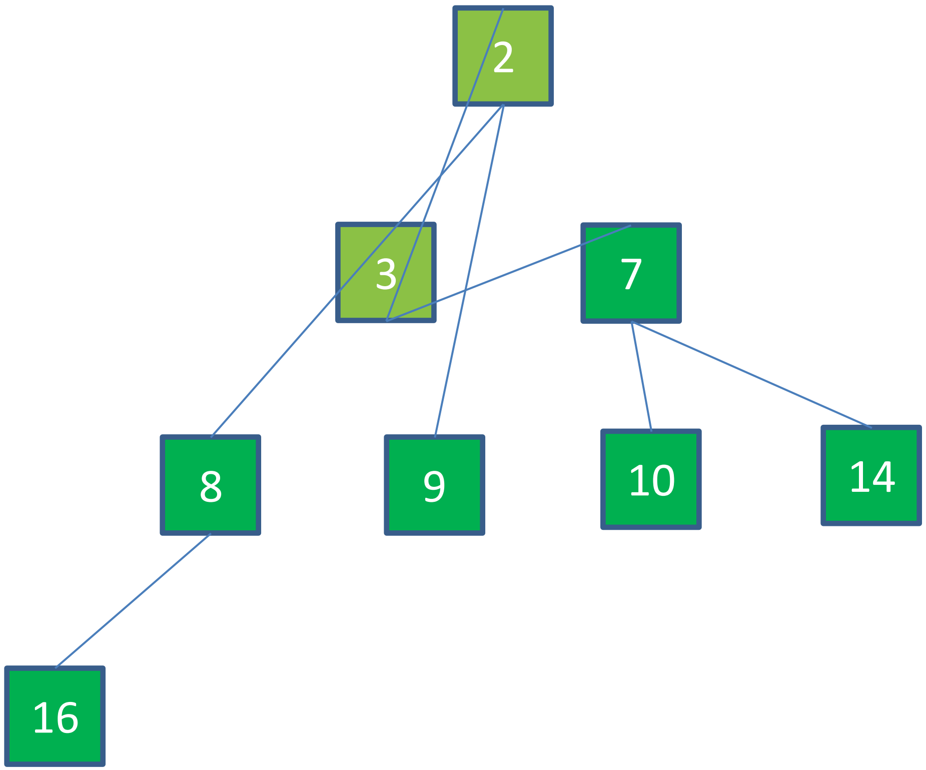
   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {
   - m = the index of the larger node
   - Exchange i with the largest node
   - **MAX-HEAPIFY** (A, m, t)

   }

Heapsort Example



HEAP-SORT (A):

1. **BUILD-MAX-HEAP**(A)
2. Last node index i = A's last node index
3. From last element to the second in A {

   exchange (i, root);

   i--;

   MAX-HEAPIFY(A, root, i);

   }

MAX-HEAPIFY (A, i, t)

1. if(right(i)>t and left(i)>t) return;
2. Choose largest (node i, left(i), right(i))
3. if(the largest node is not i) {

   m = the index of the larger node

   Exchange i with the largest node

   MAX-HEAPIFY (A, m, t)

   }

<u>Final result</u>: a sorted array A

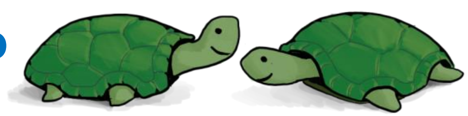
Heapsort Running Time

- The call to Build-Max-Heap() takes O(n) time
- Each of the n-1 calls to Max-Heapify() takes O(log(n)) time
- Thus the total time taken by HeapSort() is:
- = O(n) + (n-1) O(log(n))
- = O(n) + O(n.log(n))
- = O(n.log(n))

Priority Queues

- Heapsort is a nice algorithm, but in practice QuickSort (upcoming Lecture) usually wins
- But the heap data structure is incredibly useful for implementing priority queues
  - A data structure for maintaining a set *S* of elements, each with an associated value or *key*
  - Supports the operations Insert(), Maximum(), and ExtractMax()

Think-Pair-Share Terrapins



- What might a priority queue be useful for?
  - Finding next edge in graphical algorithms (e.g. Dijkstra)
  - Finding next job in an operating system scheduler

Priority Queue Operations

- **Insert(S, x)**: inserts the element x into set S
- **Maximum (S)**: returns the element of S with a maximum key
- **ExtractMax(S)**: removes and returns the element of S with the maximum key

- How could we implement these operations using a heap?

Heap-Insert()

```
HeapInsert(A, key)
{
    heap_size[A] ++;
    i = heap_size[A];
    while (i > 1  AND  A[Parent(i)] < key)
    {
        A[i] = A[Parent(i)];
        i = Parent(i);
    }
    A[i] = key;
}
```

- Running Time?
  - O(log n)

Heap-Maximum ()

```
HeapMaximum(A)
{
    // This one is really tricky:

    return A[1];
}
```

- Running Time?
    - O(1)

Heap-ExtractMax ()

```
HeapExtractMax(A)
{
    if (heap_size[A] < 1) { error; }
    max = A[1];
    A[1] = A[heap_size[A]]
    heap_size[A] --;
    Heapify(A, 1);
    return max;
}
```

- Running Time?
  - It performs only a constant amount of work on top of the O(log(n)) time for Heapify

Recap

- We saw some basic tree and heap structures
- We also saw the properties of heaps and how they can be represented as linear arrays
- We covered some heap operations, including Heapify, Build-Heap, and HeapSort, analyzing their asymptotic running times
- We introduce Priority Queues, as a useful data structure with multiple applications.
- We'll also see some faster sorting randomized algorithms…

…in a few lectures