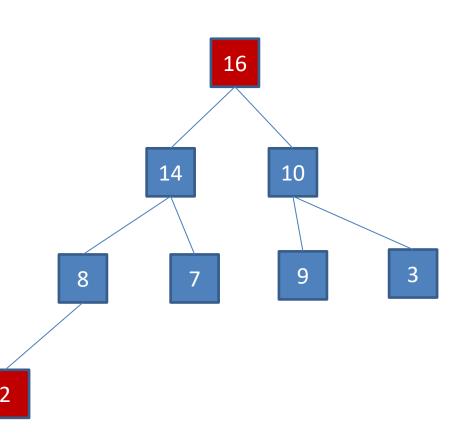


```
HEAP-SORT (A):
```

- 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);
  }

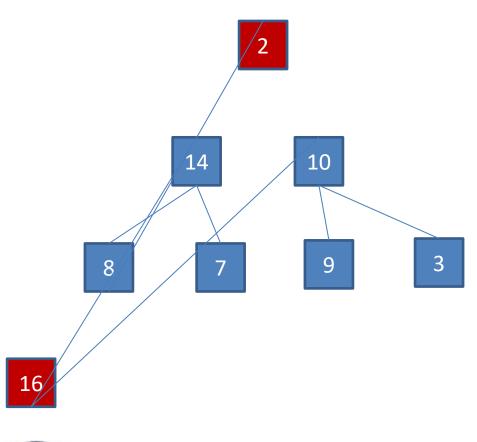
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



```
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);
  }

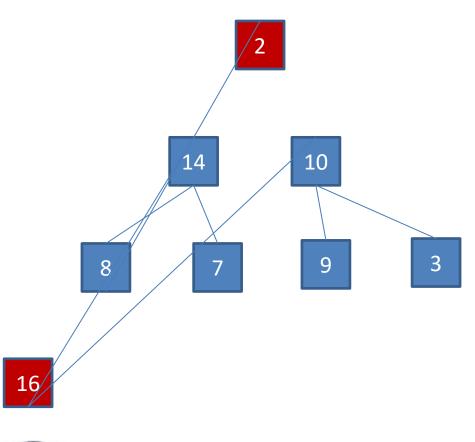
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



#### **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);
  }

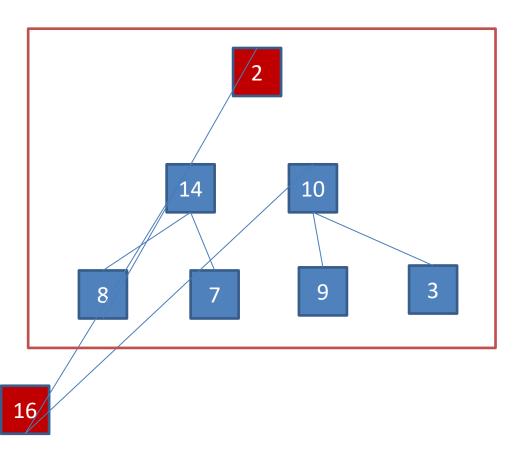
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

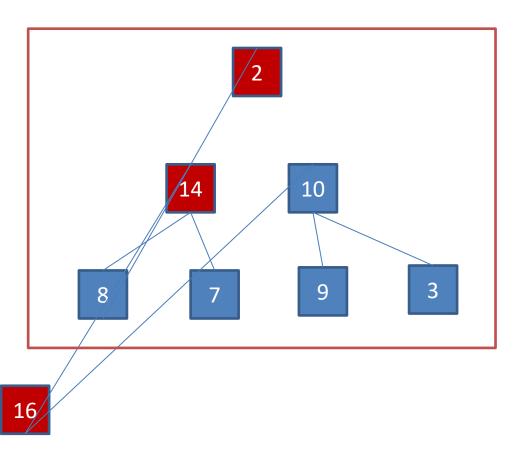
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



#### **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);
  }

- 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)

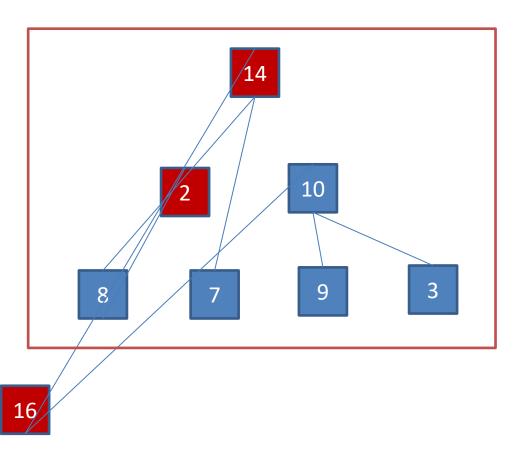




```
HEAP-SORT (A):
```

- 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);
  }

- 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)



#### **HEAP-SORT** (A):

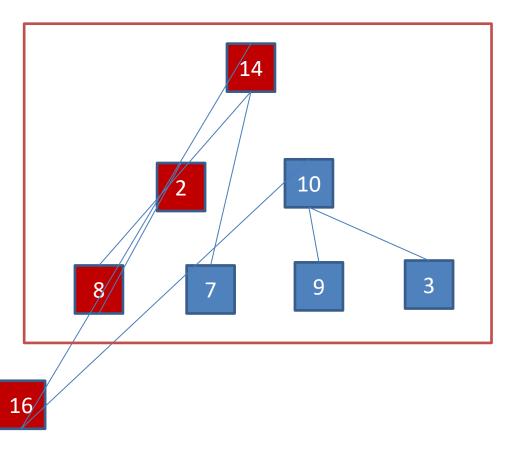
- 1. BUILD-MAX-HEAP(A)
- 2. Last node index i = A's last node index
- 3. From last element to the second in A {
   exchange (i, root);
   i--;
   MAX-HEAPIFY(A, root, i);
  }

#### **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)



**CSE 100 L07 Heaps 7** 

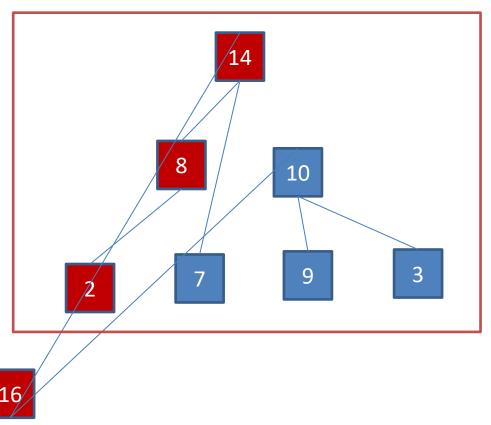


# O CONTRACTOR OF THE CONTRACTOR

```
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);

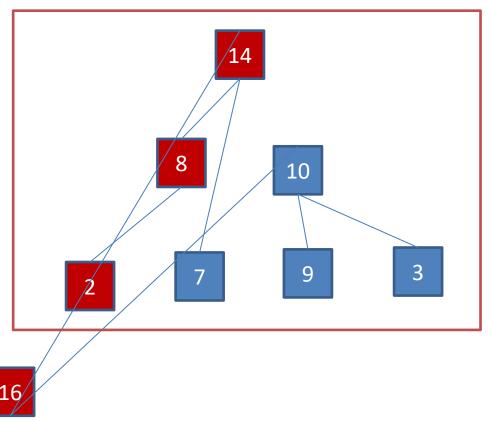
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



#### **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);
  1

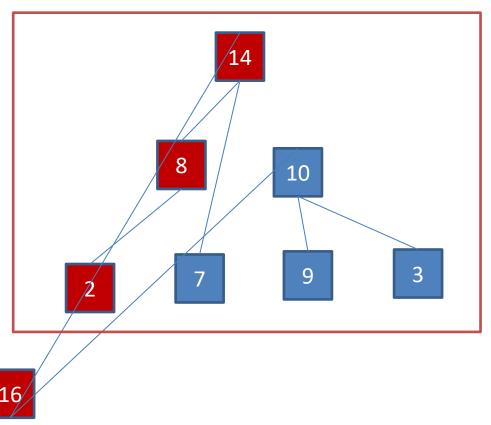
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



### **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);
  }

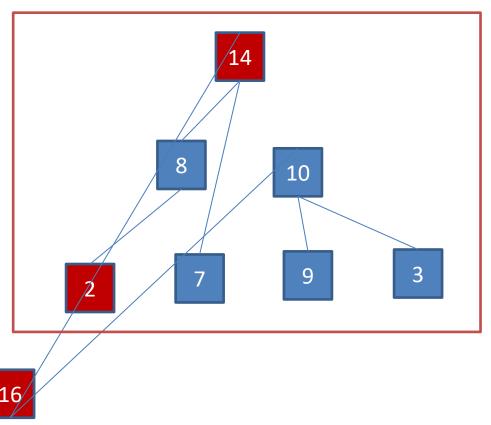
- if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



### **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);
  }

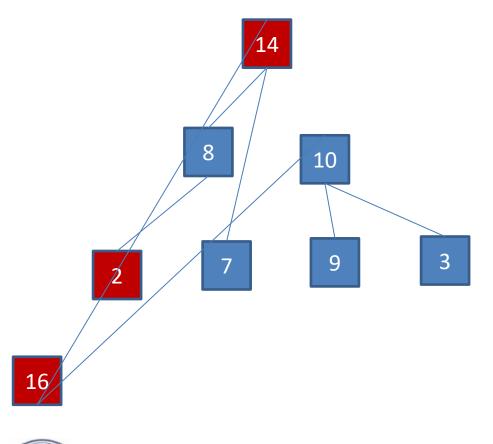
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



### **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);
  }

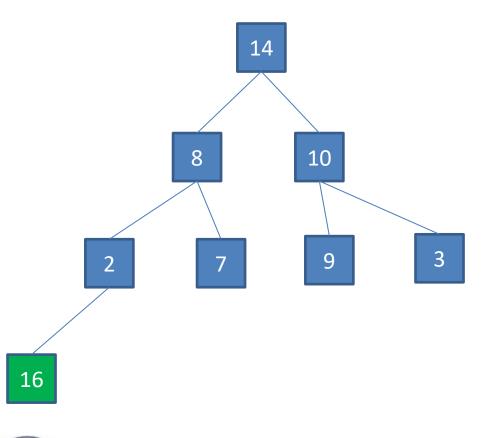
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



```
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);
  }

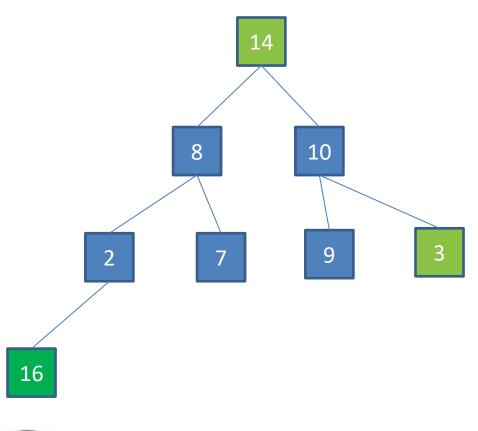
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



```
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);
  }

- 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)

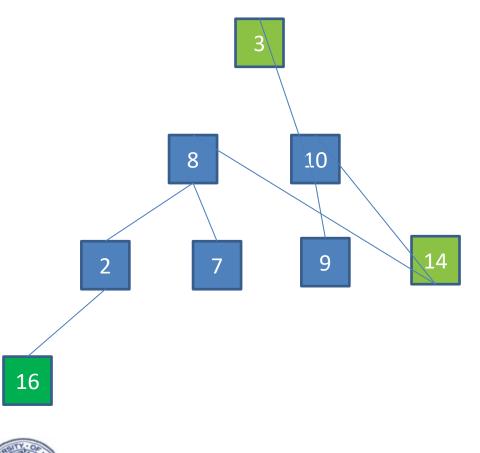


#### **HEAP-SORT** (A):

- 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);

- 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)

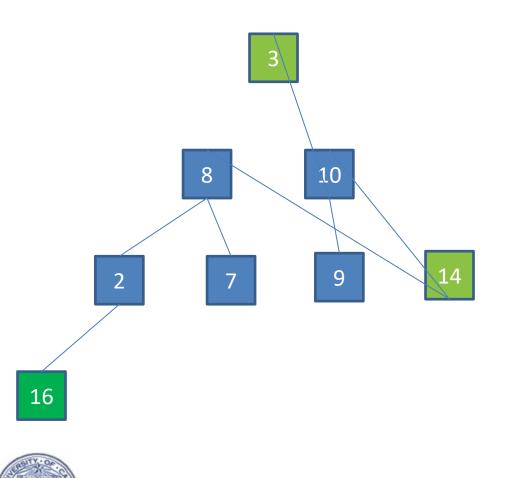




```
HEAP-SORT (A):
```

- 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);
  }

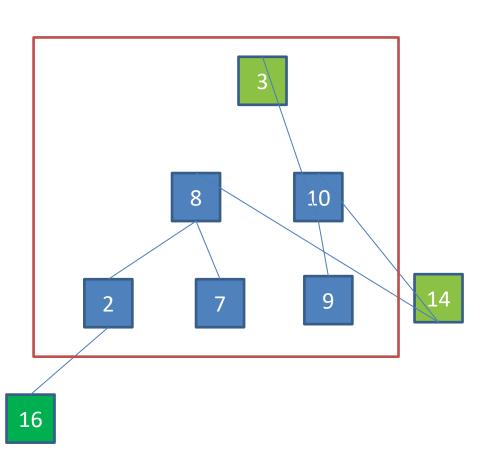
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

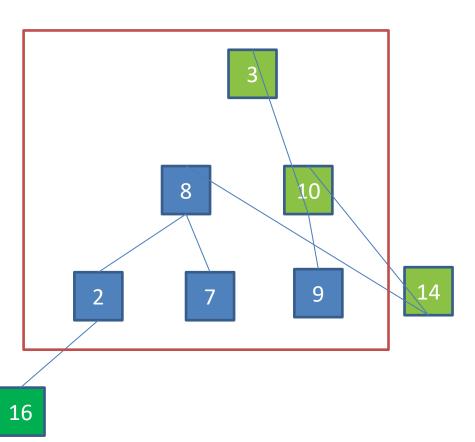
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

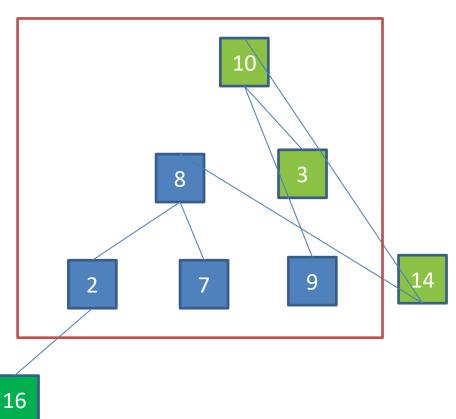
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



```
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);
  }

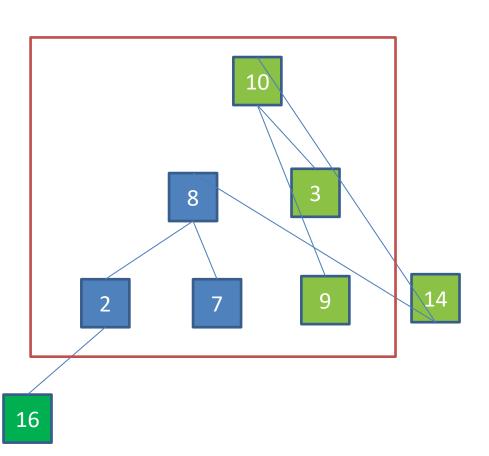
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



```
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);

- 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)

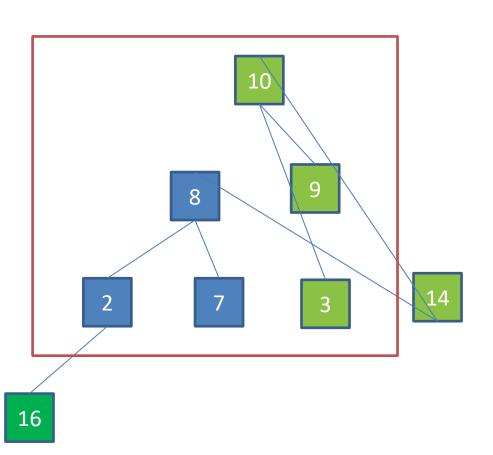


T Too

```
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);
  }

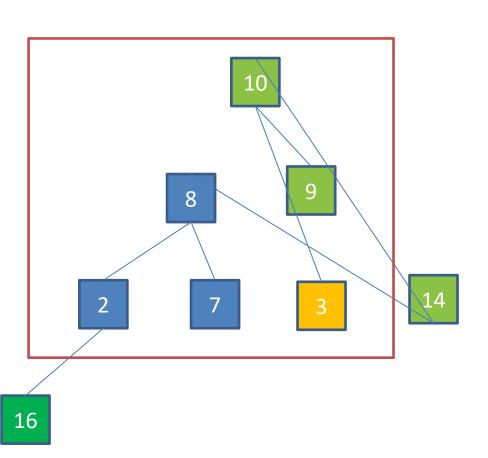
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

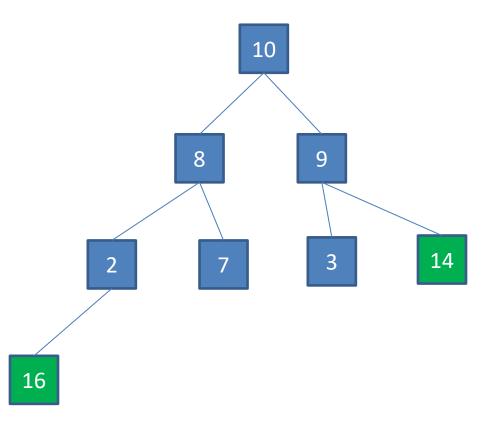
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

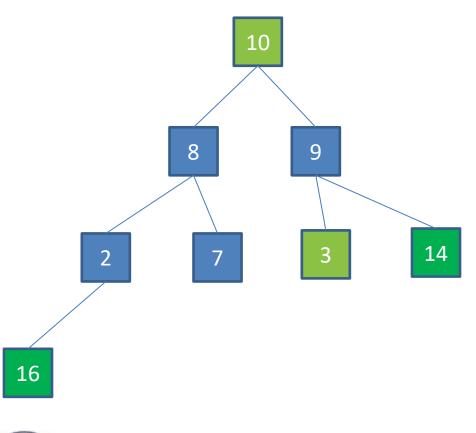
- if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



#### **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);
  }

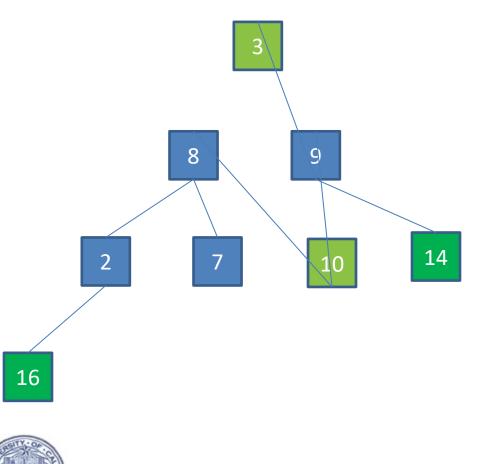
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

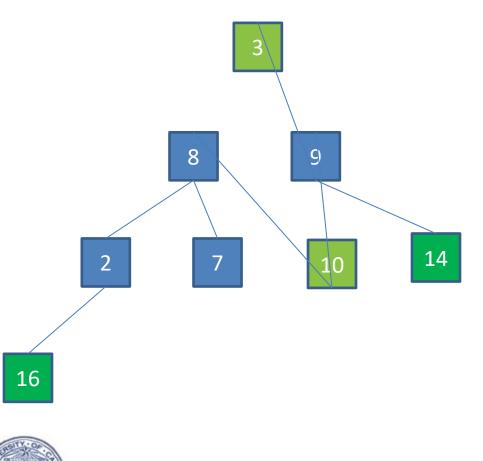
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

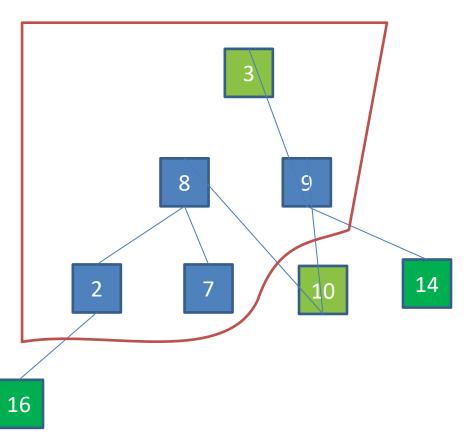
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



#### **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);
  }

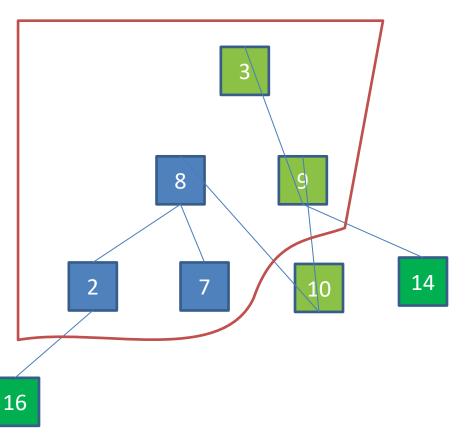
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

- 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)

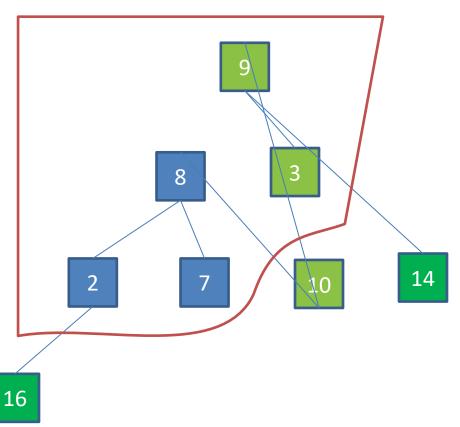


```
Services of the services of th
```

```
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);
  }

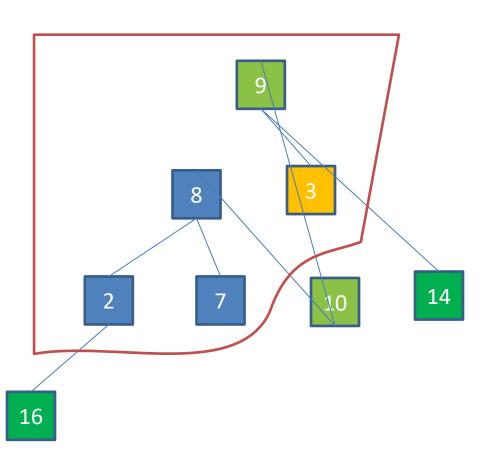
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



### **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);
  }

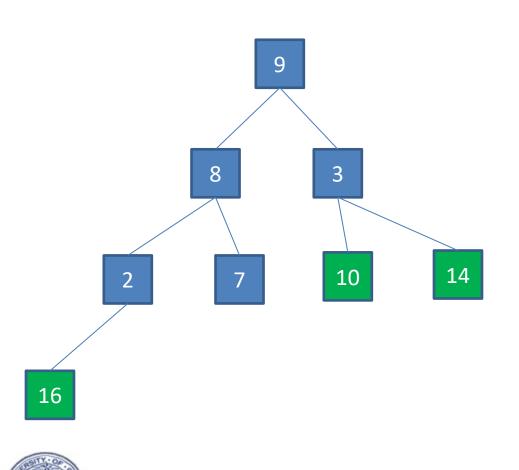
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

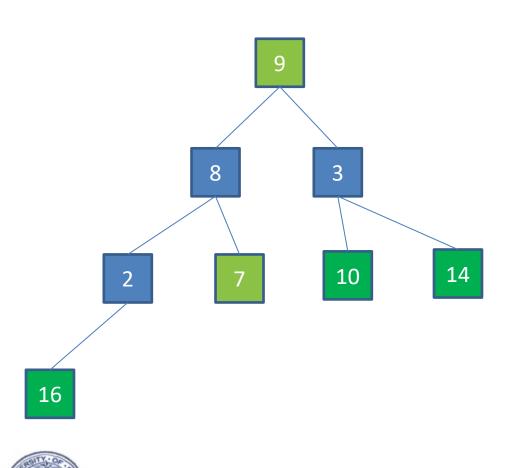
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);

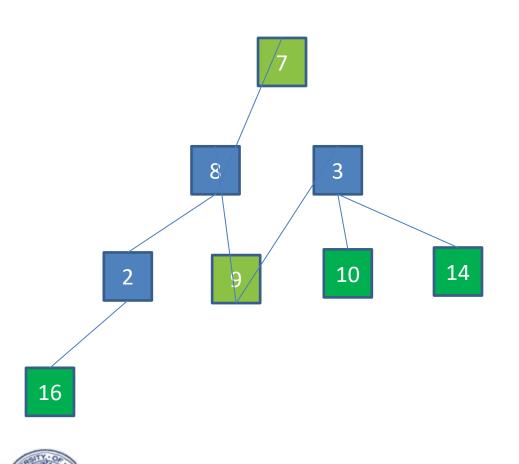
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);

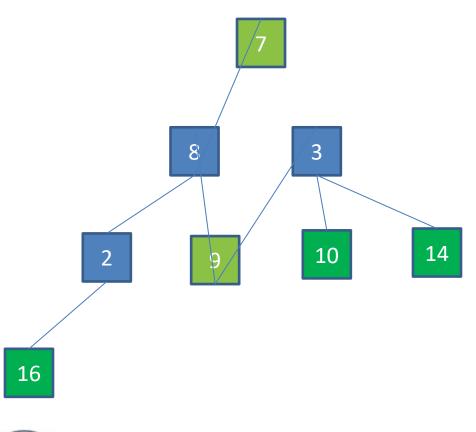
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



```
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);
  }

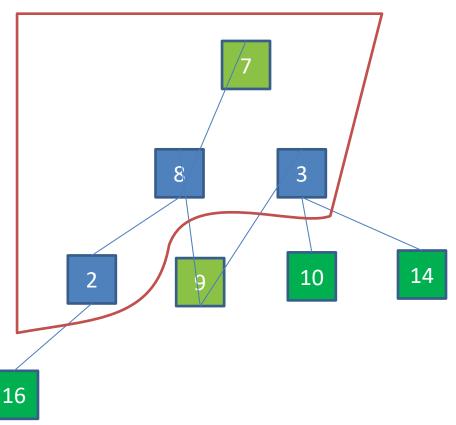
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



#### **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);
  }

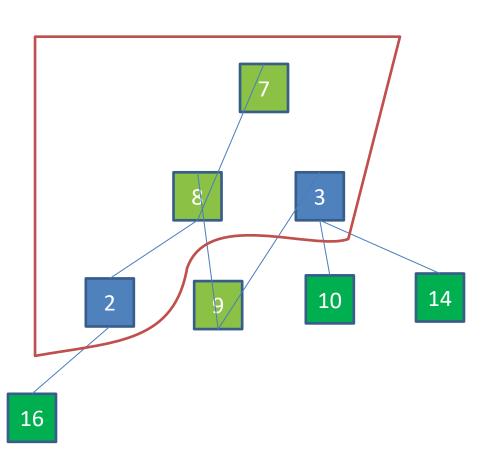
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

- 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)

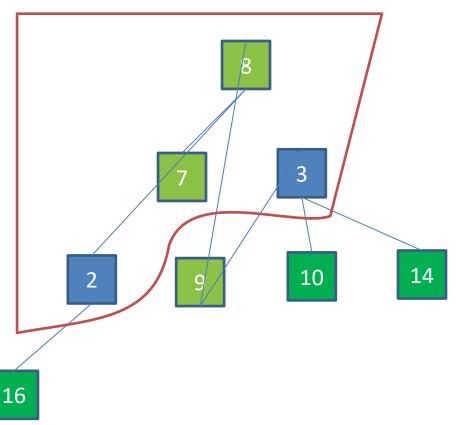


# **HEAP-SORT** (A):

- 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);
  }

- 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)

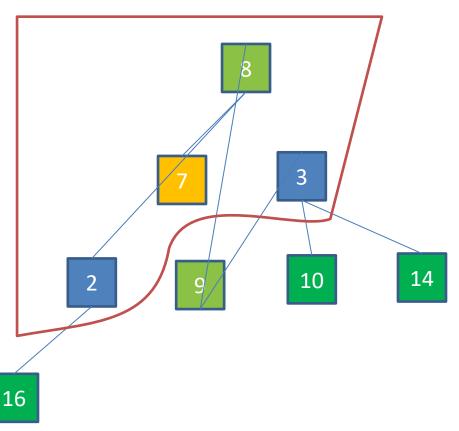




# **HEAP-SORT** (A):

- 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);
  }

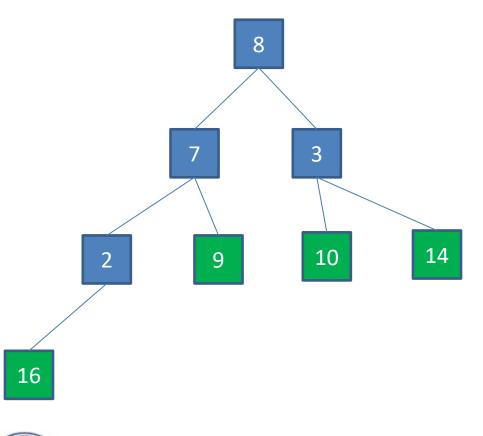
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

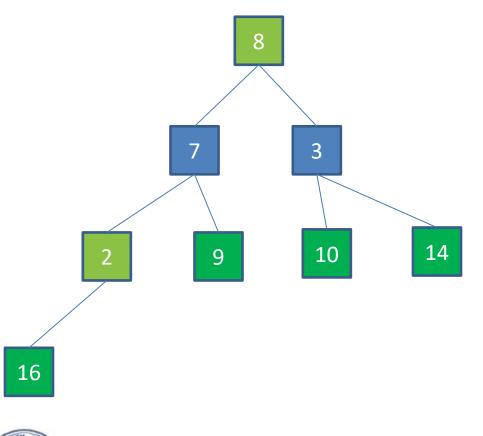
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



#### **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);
  }

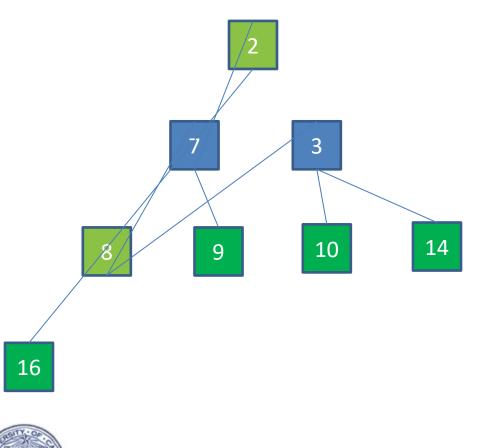
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);

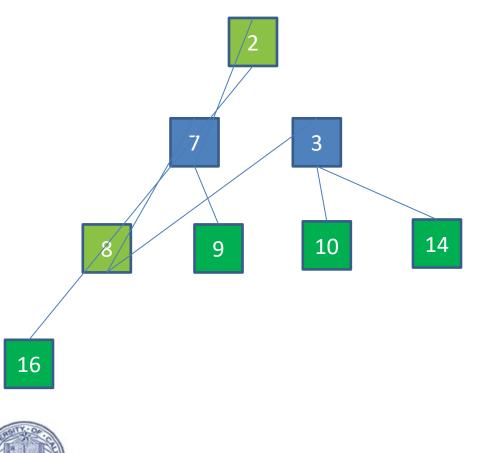
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



# **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);
  }

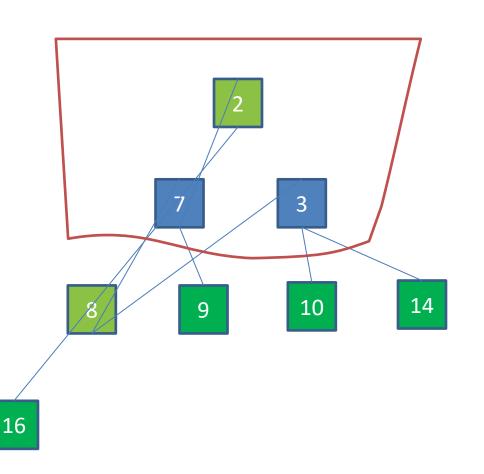
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

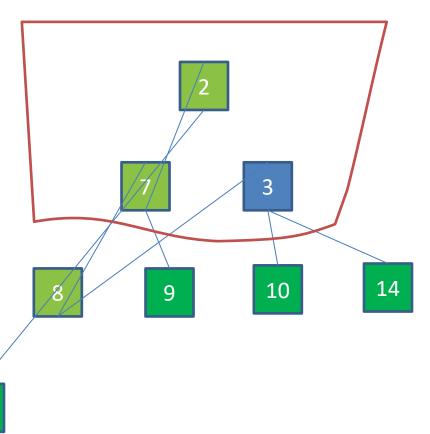
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

- 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)



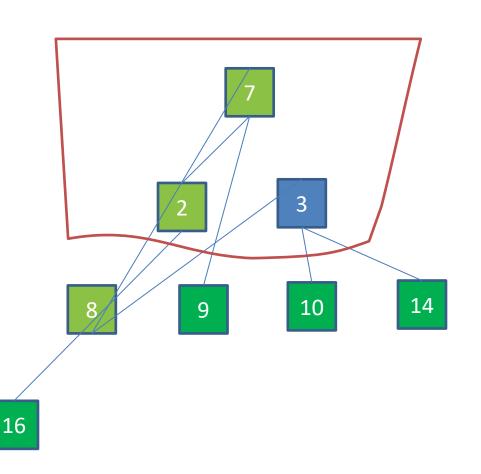
# **HEAP-SORT** (A):

- 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)

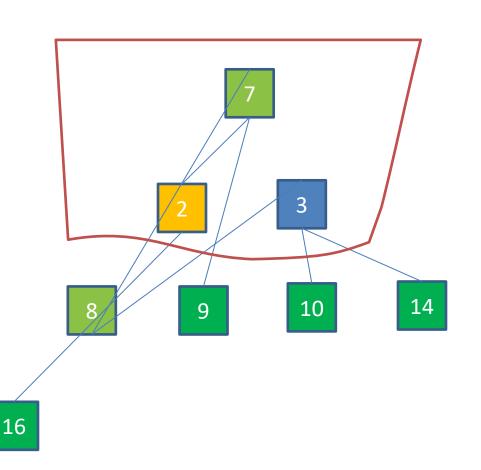
16



```
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);
  }

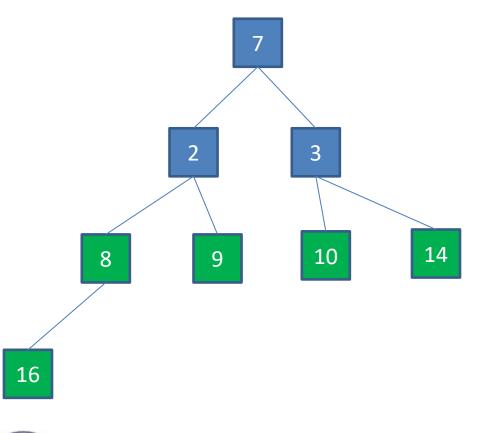
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

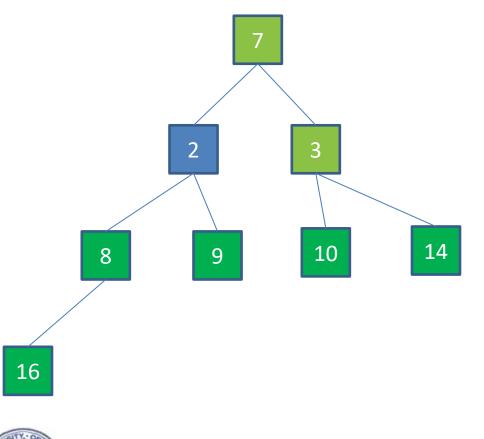
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);

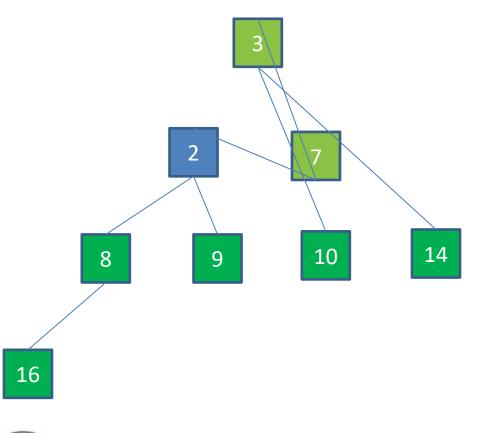
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

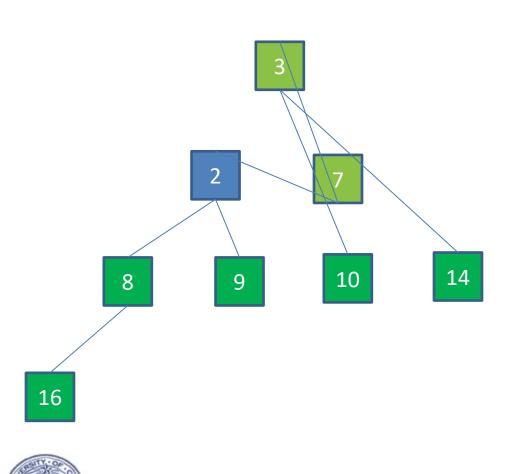
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

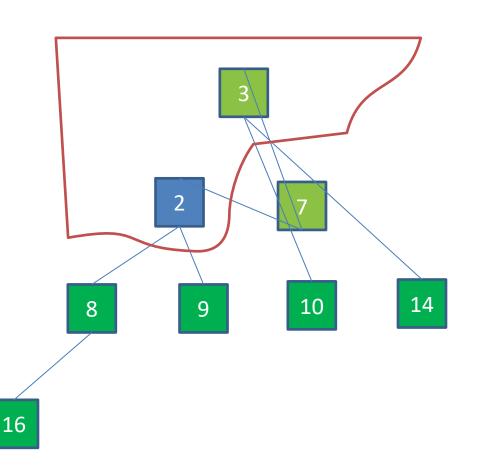
- if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

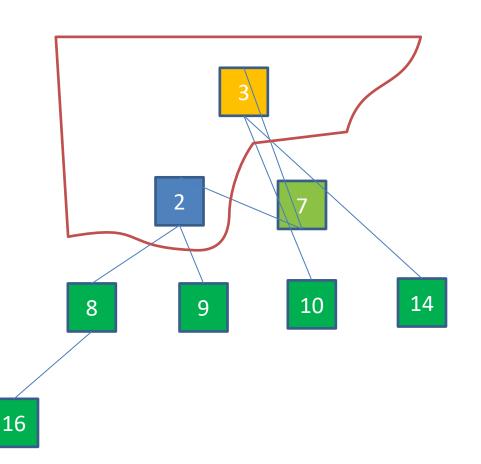
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

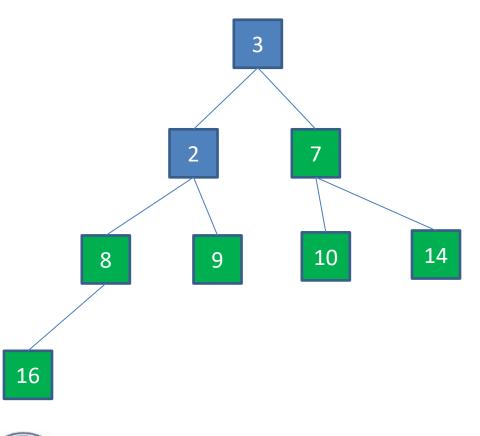
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
   m = the index of the larger node
   Exchange i with the largest node
   MAX-HEAPIFY (A, m, t)



```
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);
  }

- 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)

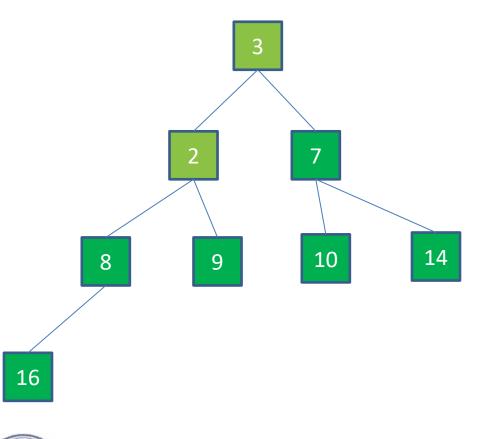


#### **HEAP-SORT** (A):

- 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);
  }

- 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)

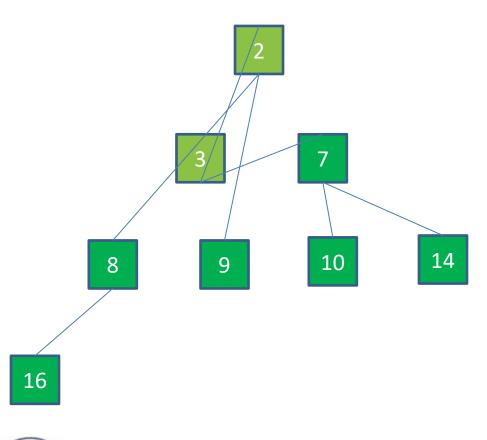




```
HEAP-SORT (A):
```

- 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);
  }

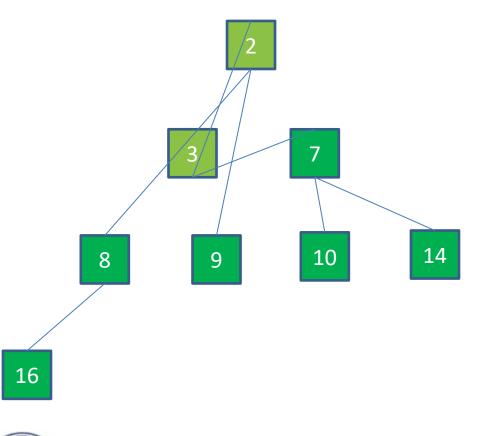
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

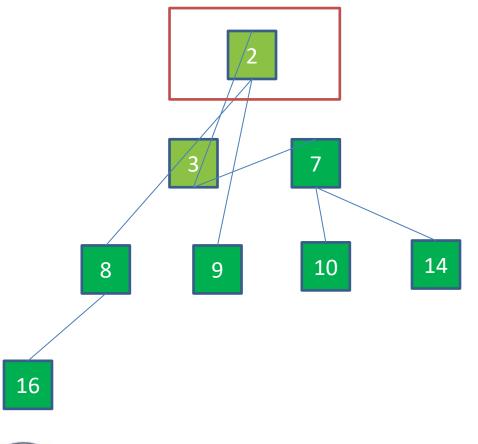
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

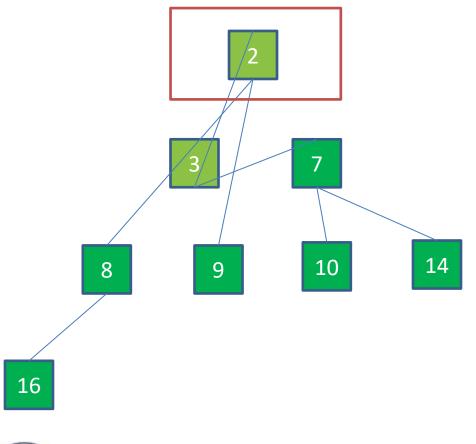
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

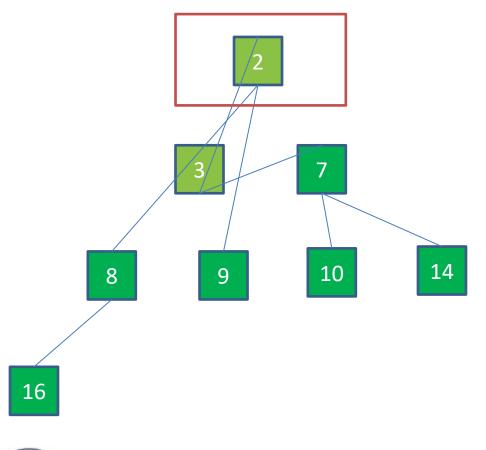
- if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



#### **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);
  }

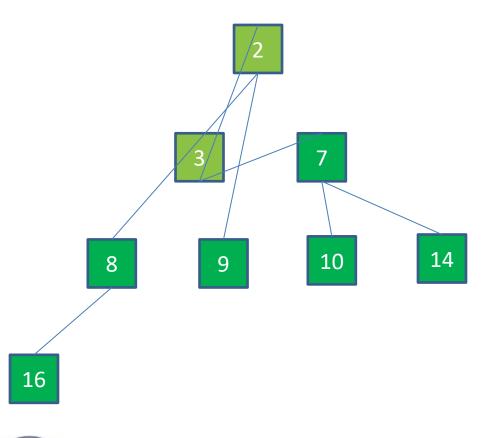
- if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

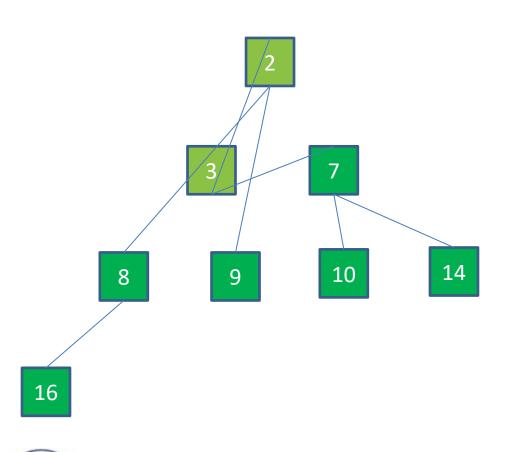
- 1. if(right(i)>t and left(i)>t) return;
- 2. Choose largest (node i, left(i), right(i))
- 3. if(the largest node is not i) {
  m = the index of the larger node
  Exchange i with the largest node
  MAX-HEAPIFY (A, m, t)



```
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);
  }

- 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 sorted array A

CSE 100 L07 Heaps 62
```

#### **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);
  }

- 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)