Heapsort

Prof. Navrati Saxena

Heap Sort: Key Idea

HEAPSORT(A)

1. *Build a MAX-HEAP with the Array*
2. *Exchange the root (maximum element) with the last element*
3. *Reduce the Size of the heap (array size) by 1*
4. *Call MAX-HEAPIFY with array A and the 1st index*

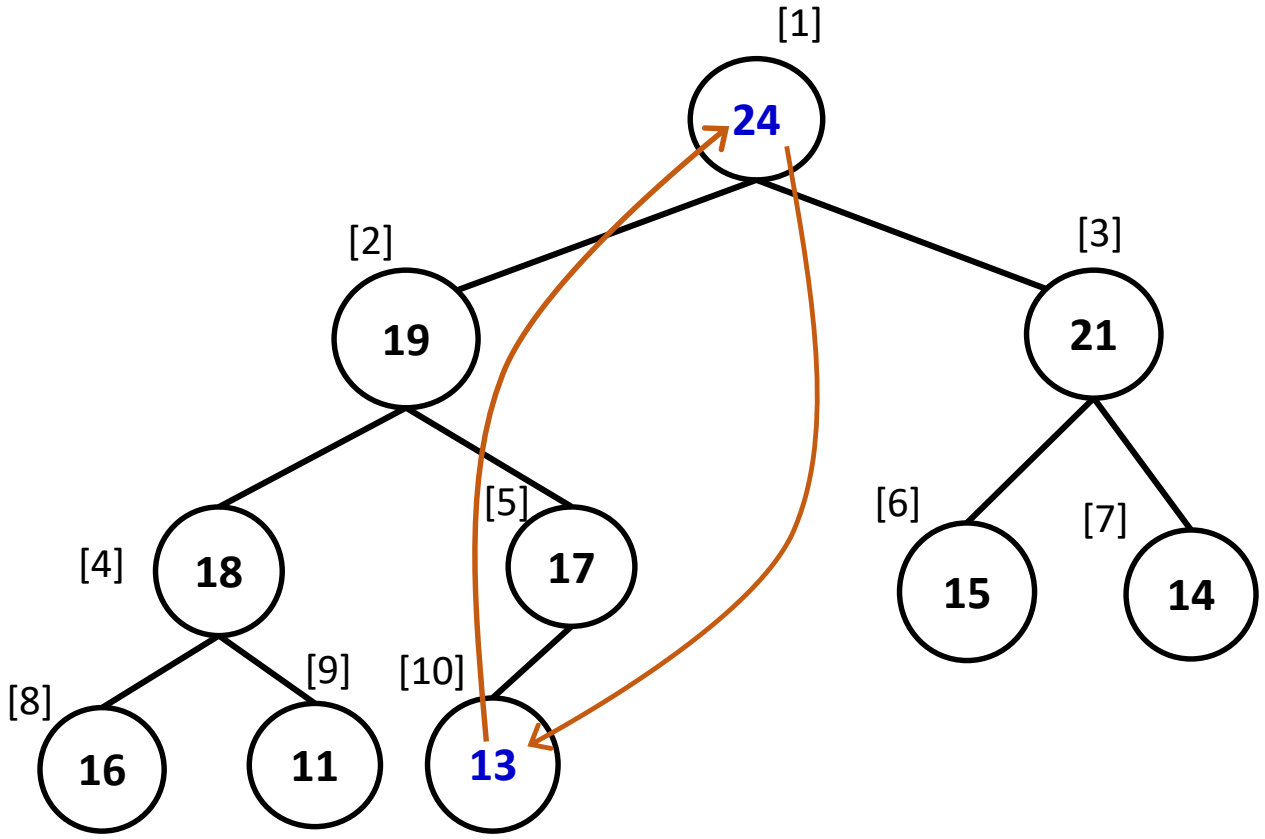
Heap Sort: Pseudo-code

```
HEAPSORT(A)
1.  BUILD-MAX-HEAP(A)
2.  for i ← length of A down to 2
3.      exchange A[1] ↔ A[i]
4.      heapsize of A ← heapsize of A-1
5.      MAX-HEAPIFY (A,1)
```

Heapsort: An Example (1/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*

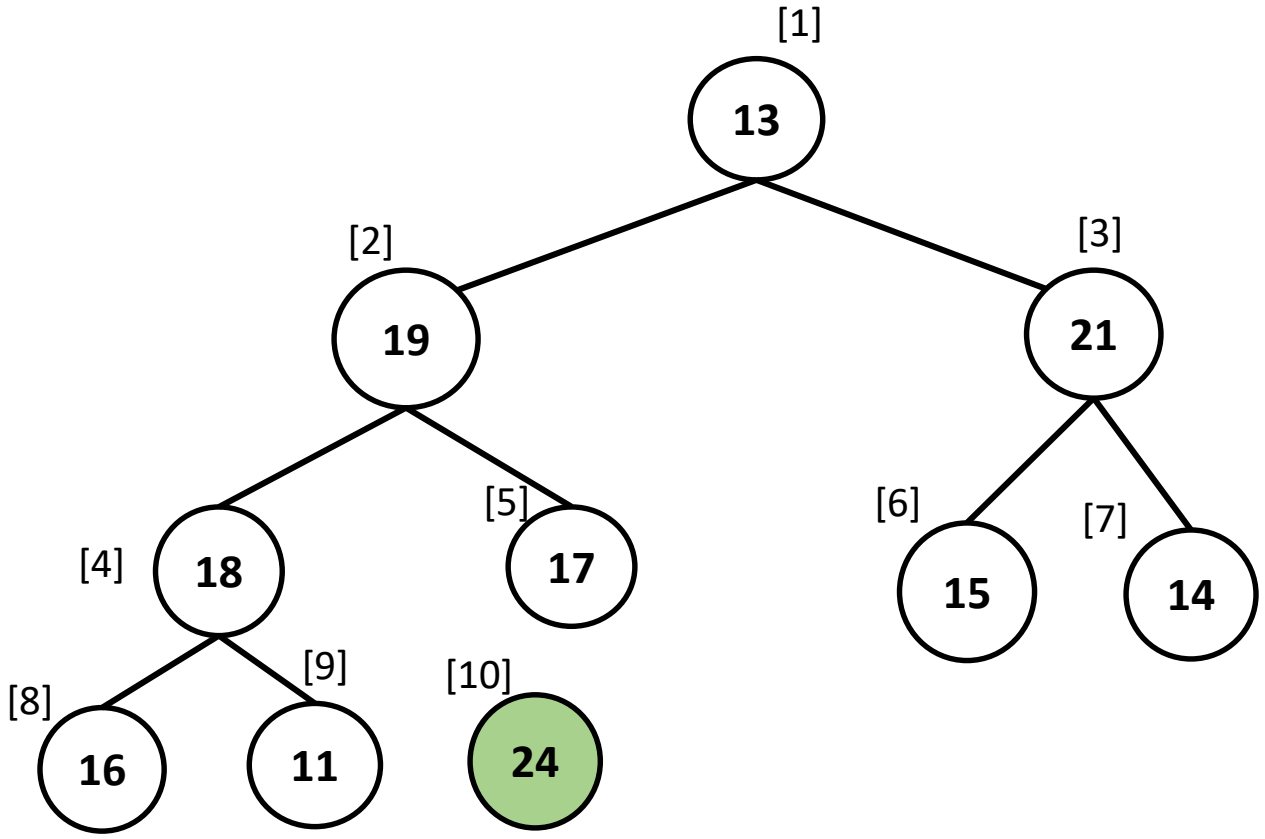


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**13**	**19**	**21**	**18**	**17**	**15**	**14**	**16**	**11**	**24**

Heapsort: An Example (2/18)

HEAPSORT(A)

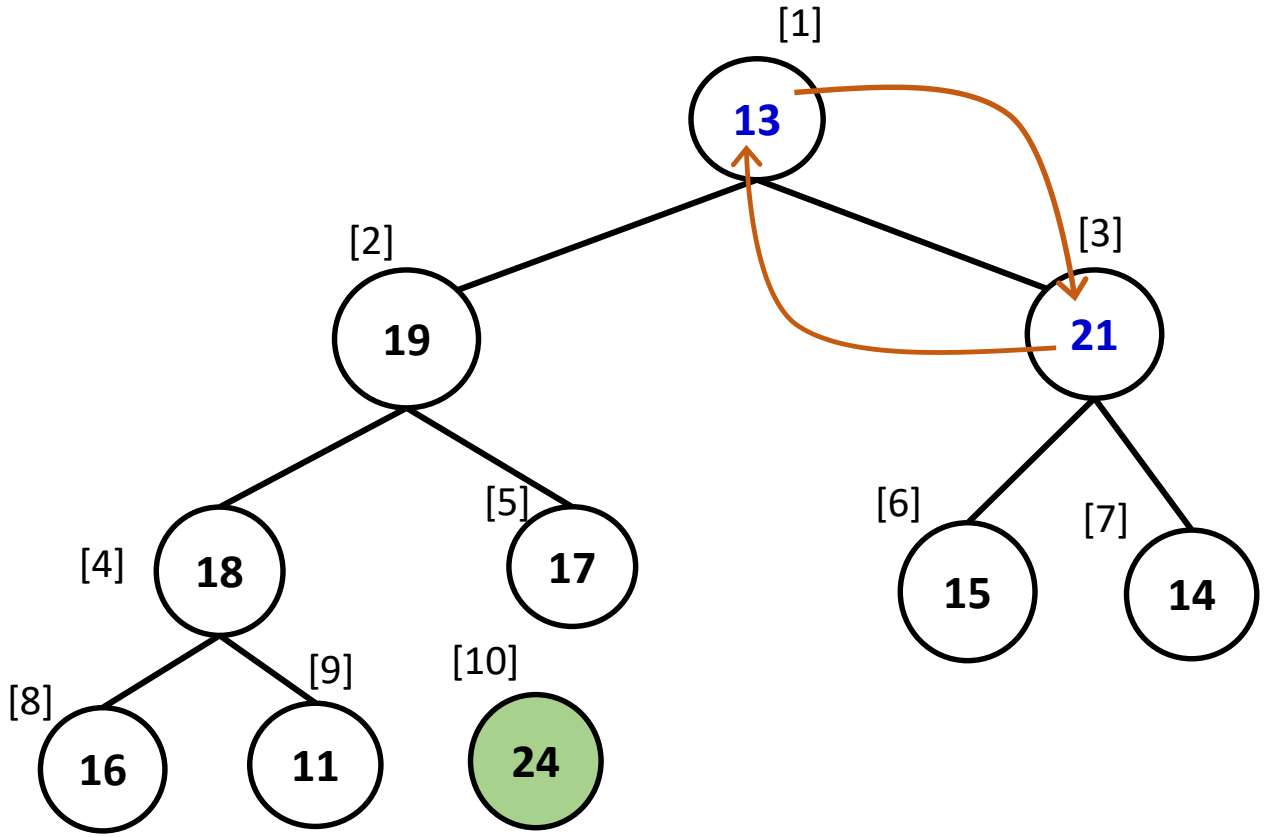
1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*



	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**13**	**19**	**21**	**18**	**17**	**15**	**14**	**16**	**11**	**24**

Heapsort: An Example (3/18)

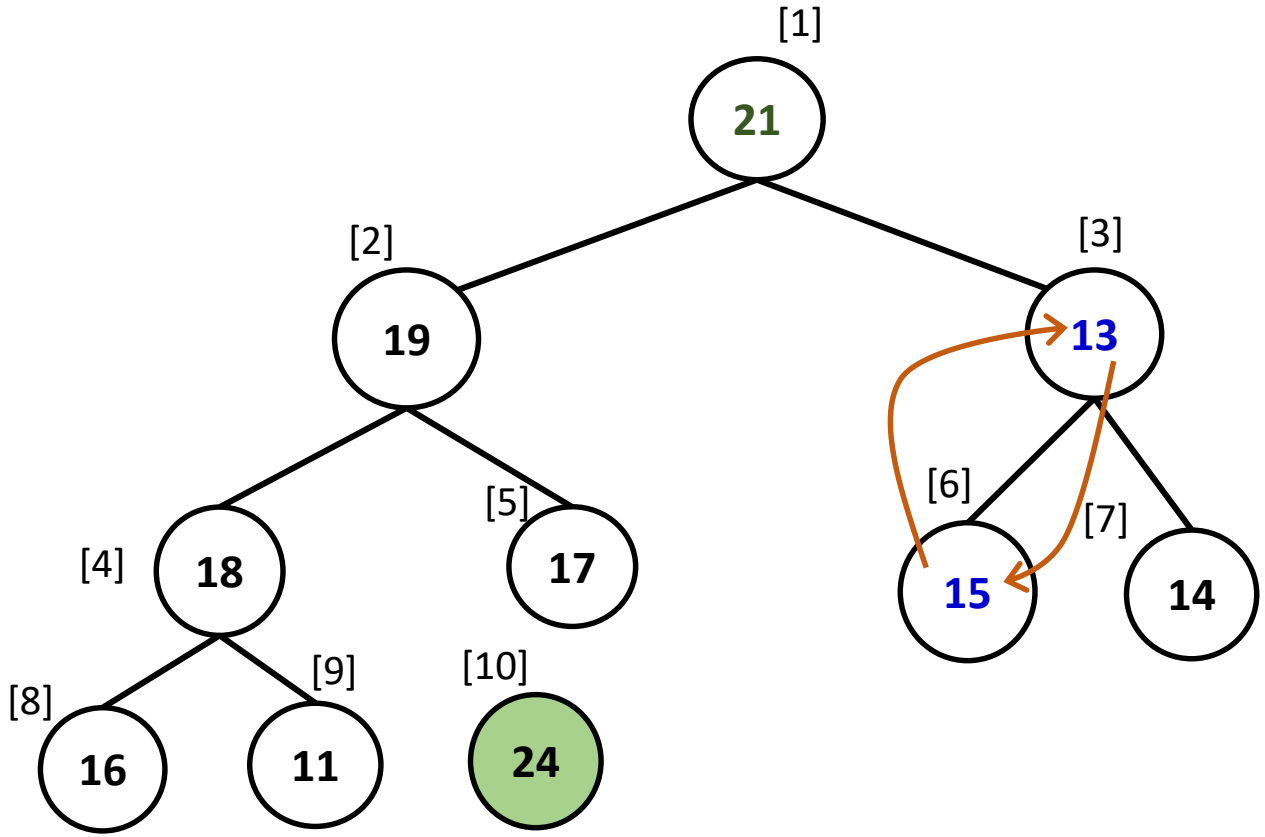
```
HEAPSORT(A)
1.  BUILD-MAX-HEAP(A)
2.  for i ← length of A down to 2
3.      exchange A[1] ↔ A[i]
4.      heapsize of A ← heapsize of A-1
5.      MAX-HEAPIFY (A,1)
```



	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**21**	**19**	**13**	**18**	**17**	**15**	**14**	**16**	**11**	**24**

Heapsort: An Example (4/18)

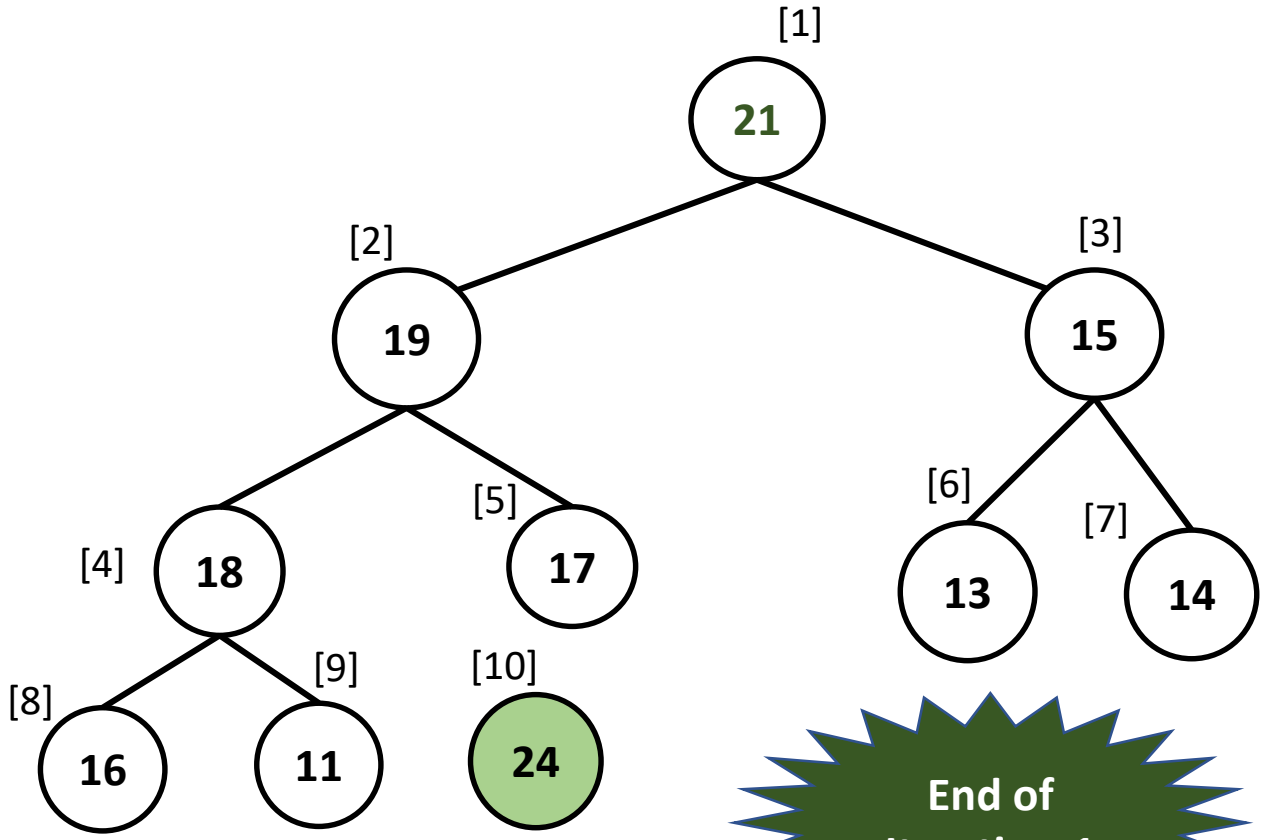
```
HEAPSORT(A)
1.  BUILD-MAX-HEAP(A)
2.  for i ← length of A down to 2
3.      exchange A[1] ↔ A[i]
4.      heapsize of A ← heapsize of A-1
5.      MAX-HEAPIFY (A,1)
```



	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**21**	**19**	**15**	**18**	**17**	**13**	**14**	**16**	**11**	**24**

Heapsort: An Example (5/18)

```
HEAPSORT(A)
1.    BUILD-MAX-HEAP(A)
2.    for i ← length of A down to 2
3.        exchange A[1] ↔ A[i]
4.        heapsize of A ← heapsize of A-1
5.        MAX-HEAPIFY (A,1)
```



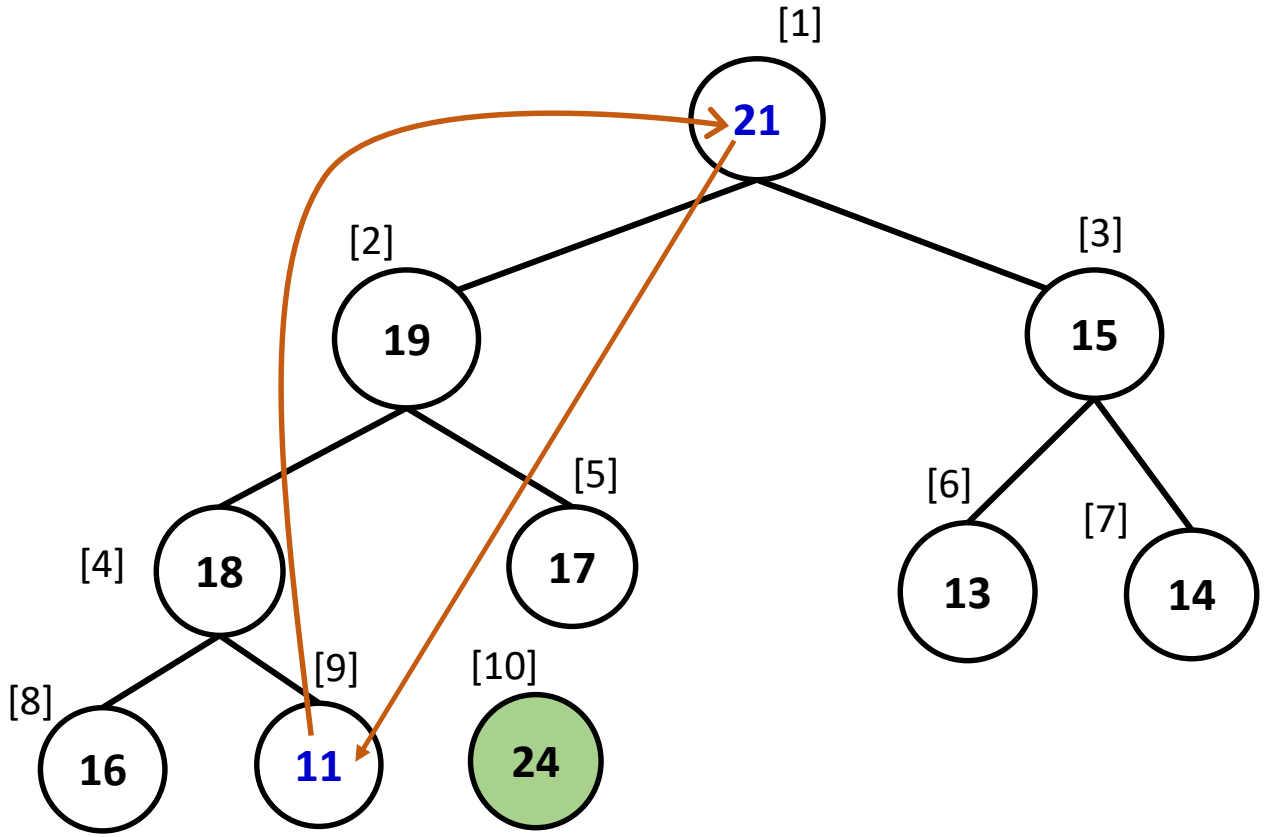
End of Iteration-1

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**21**	**19**	**15**	**18**	**17**	**13**	**14**	**16**	**11**	**24**

Heapsort: An Example (6/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*

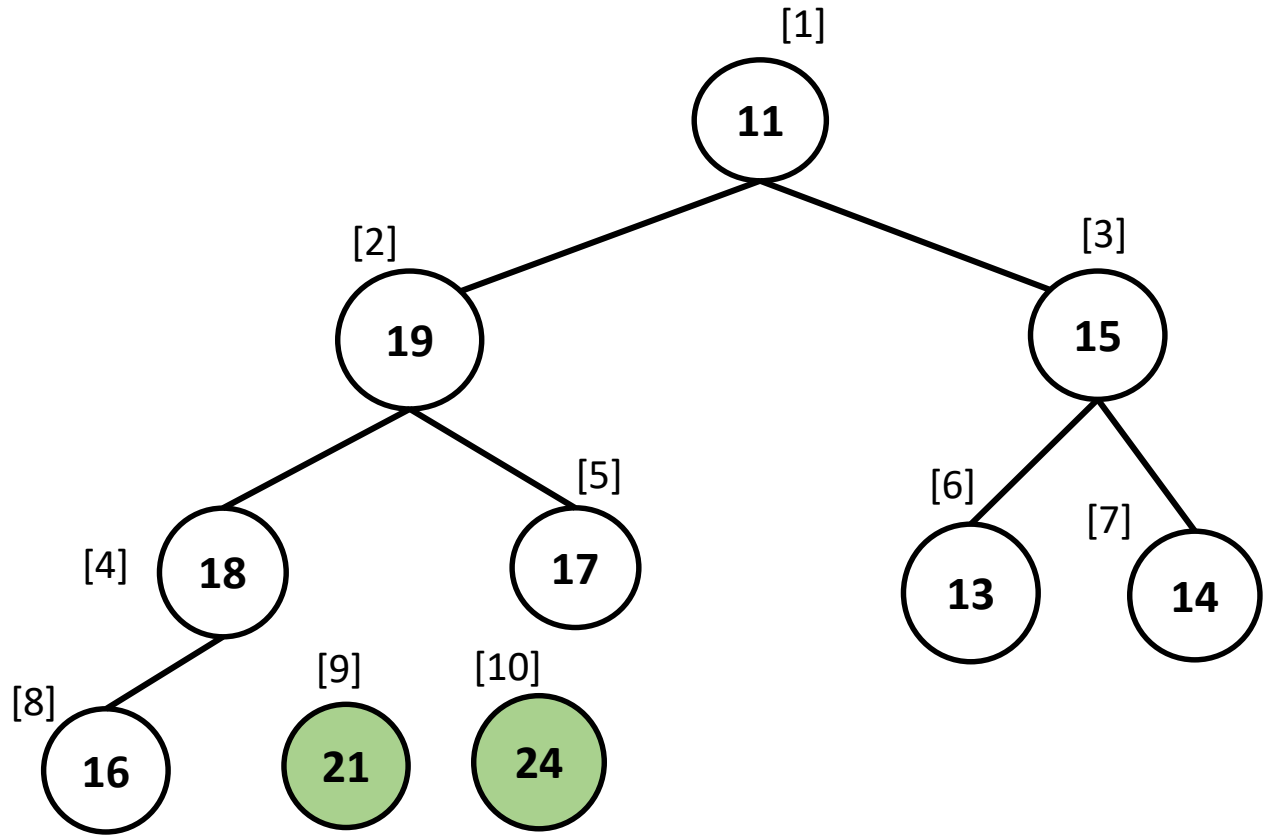


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**11**	**19**	**15**	**18**	**17**	**13**	**14**	**16**	**21**	**24**

Heapsort: An Example (7/18)

HEAPSORT(A)

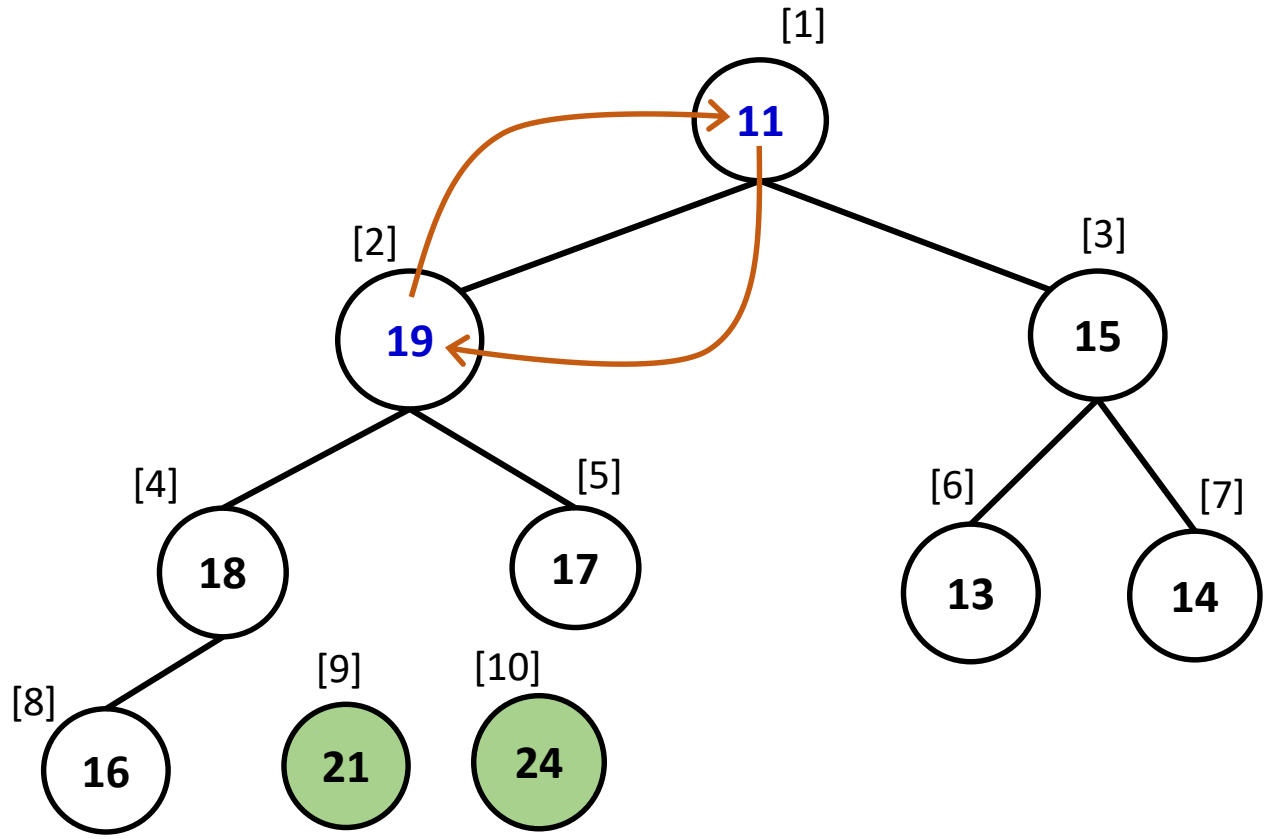
1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*



	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**11**	**19**	**15**	**18**	**17**	**13**	**14**	**16**	**21**	**24**

Heapsort: An Example (8/18)

```
HEAPSORT(A)
1.  BUILD-MAX-HEAP(A)
2.  for i ← length of A down to 2
3.      exchange A[1] ↔ A[i]
4.      heapsize of A ← heapsize of A-1
5.      MAX-HEAPIFY (A,1)
```

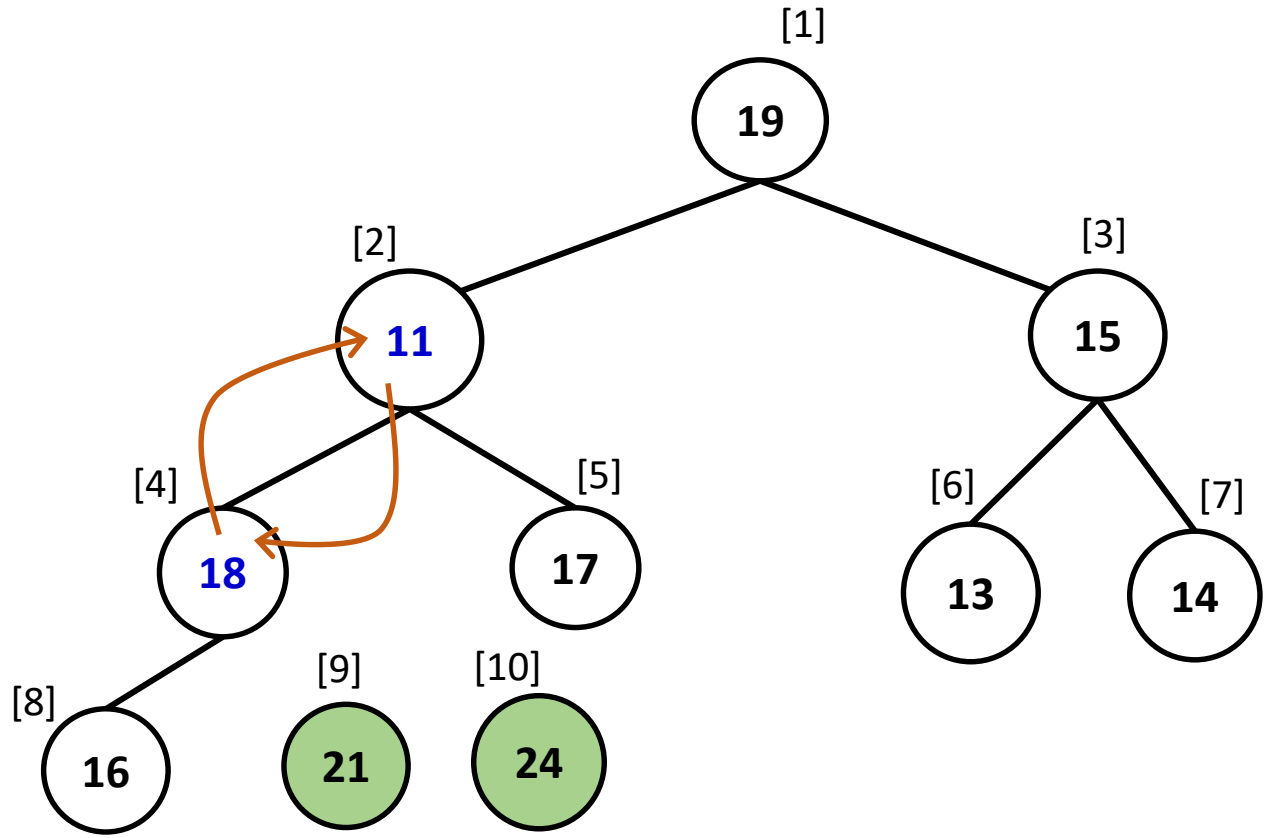


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	19	11	15	18	17	13	14	16	21	24

Heapsort: An Example (9/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*

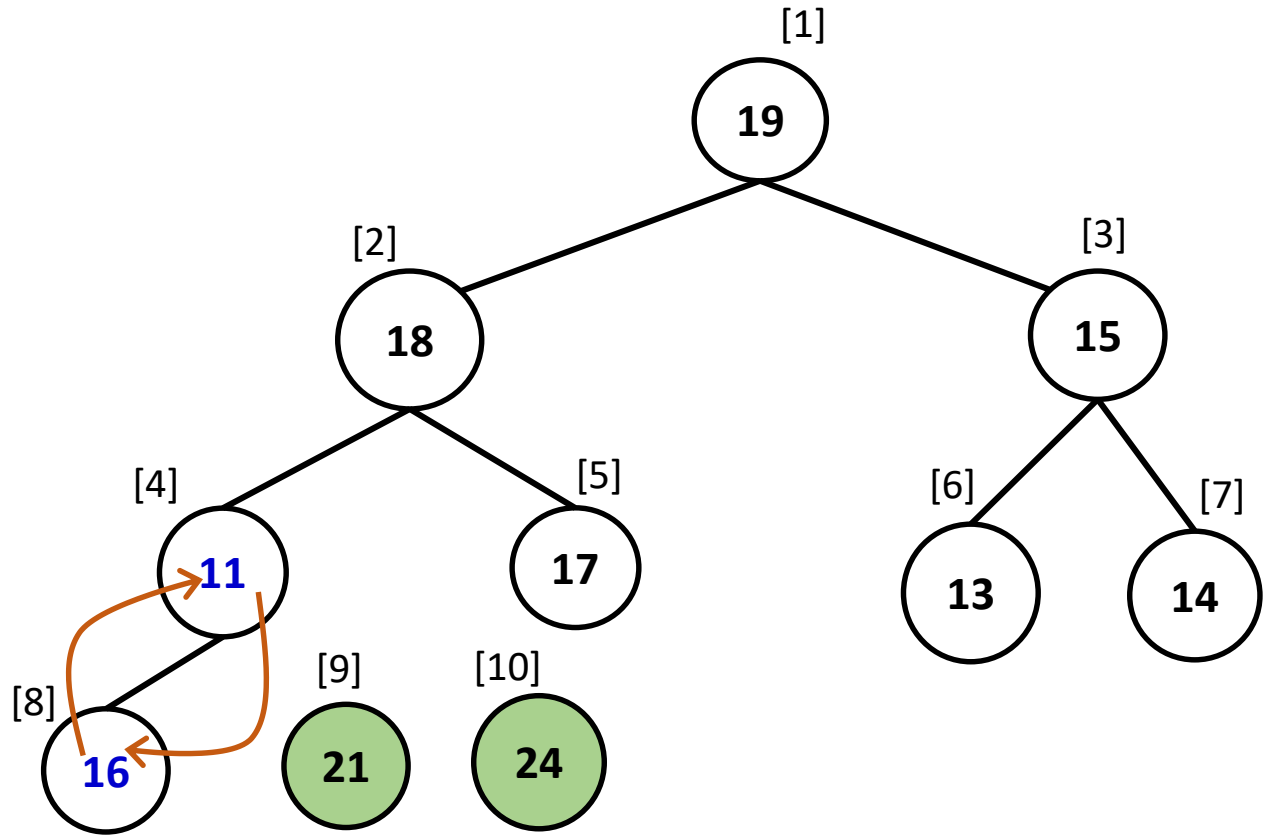


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	19	18	15	11	17	13	14	16	21	24

Heapsort: An Example (10/18)

HEAPSORT(A)

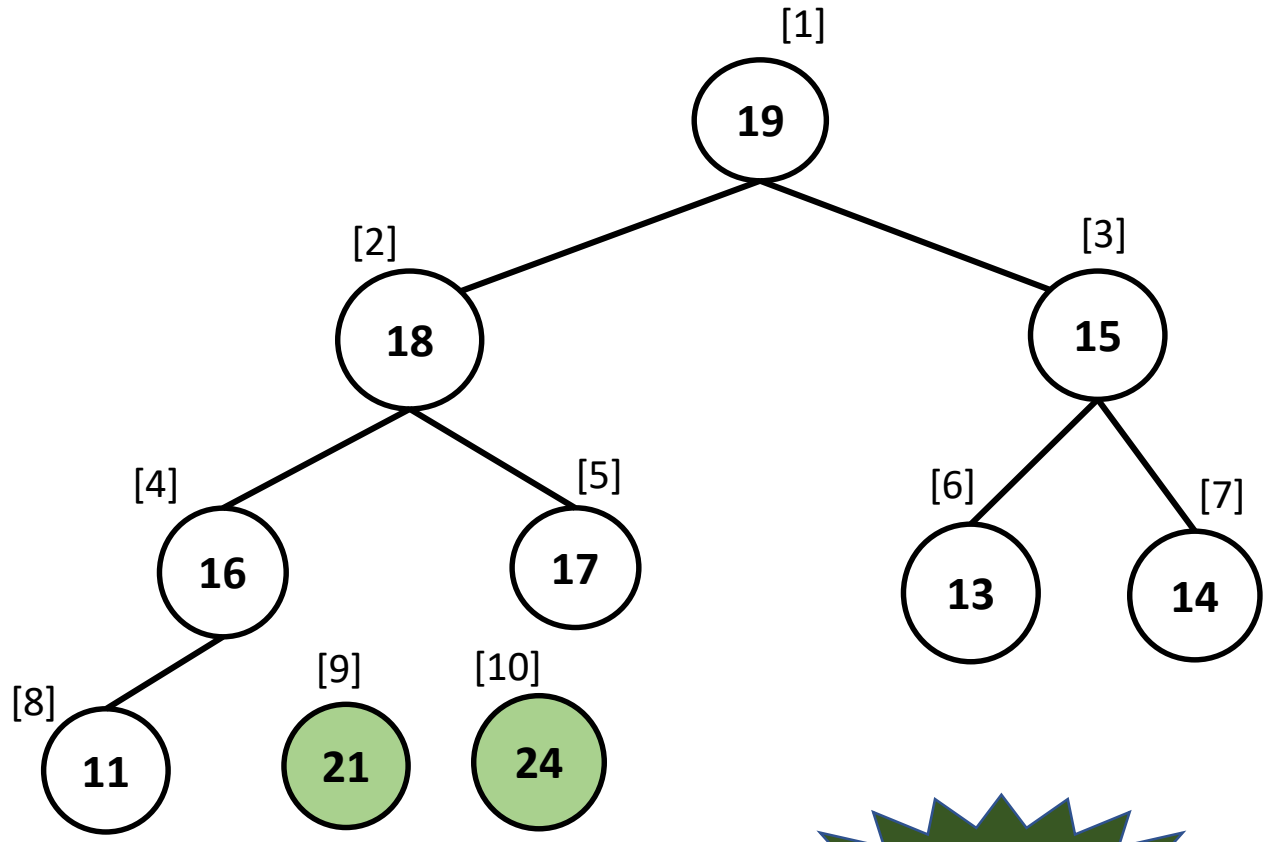
1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*



	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	19	18	15	11	17	13	14	16	21	24

Heapsort: An Example (11/18)

```
HEAPSORT(A)
1.  BUILD-MAX-HEAP(A)
2.  for i ← length of A down to 2
3.      exchange A[1] ↔ A[i]
4.      heapsize of A ← heapsize of A-1
5.      MAX-HEAPIFY (A,1)
```



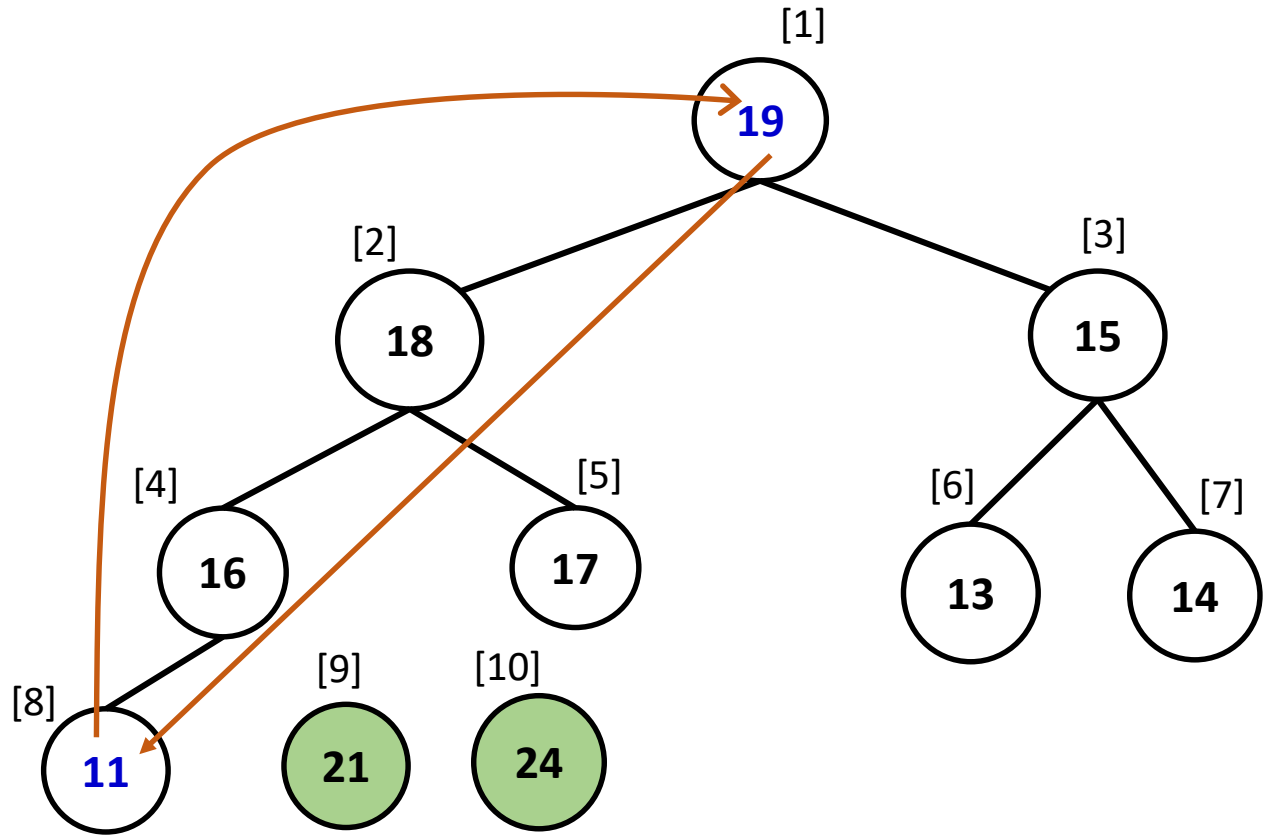
	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	19	18	15	16	17	13	14	11	21	24

End of Iteration-2

Heapsort: An Example (12/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*

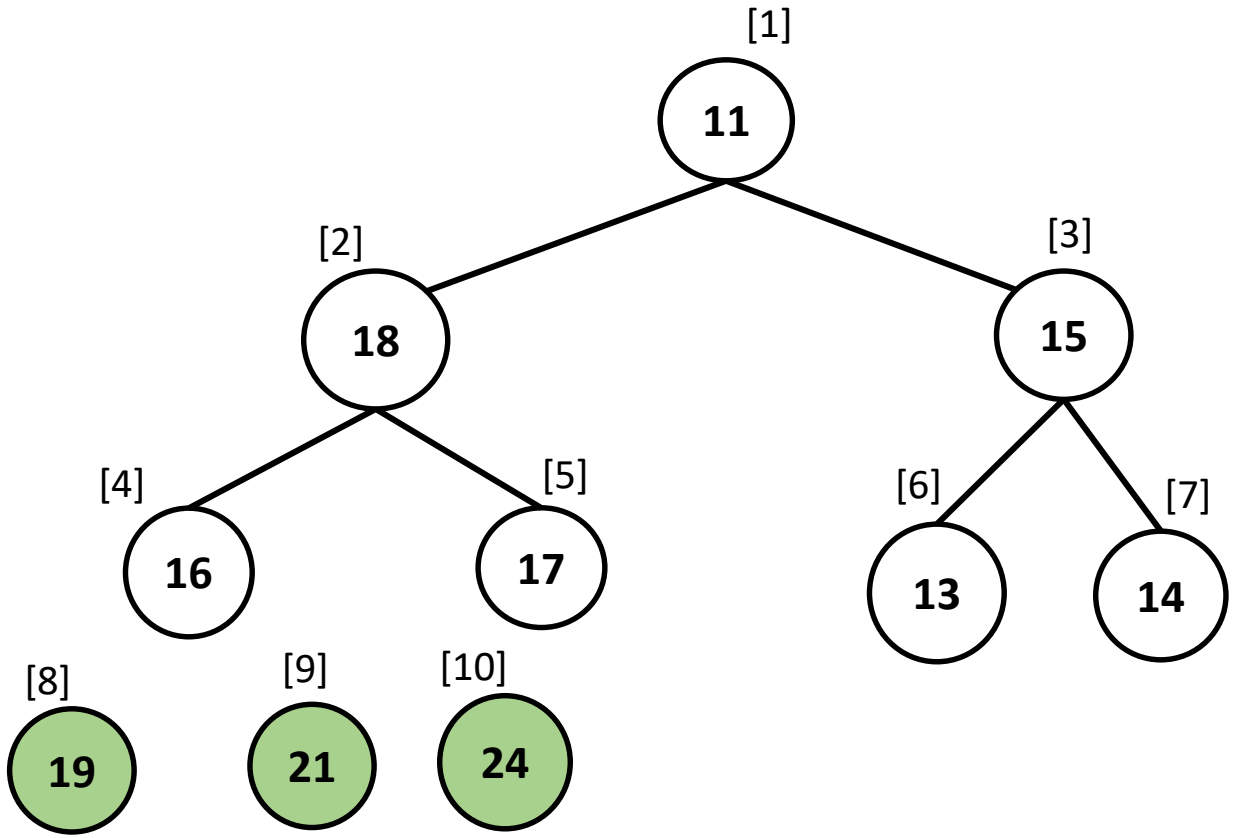


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**11**	**18**	**15**	**16**	**17**	**13**	**14**	**19**	**21**	**24**

Heapsort: An Example (13/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*

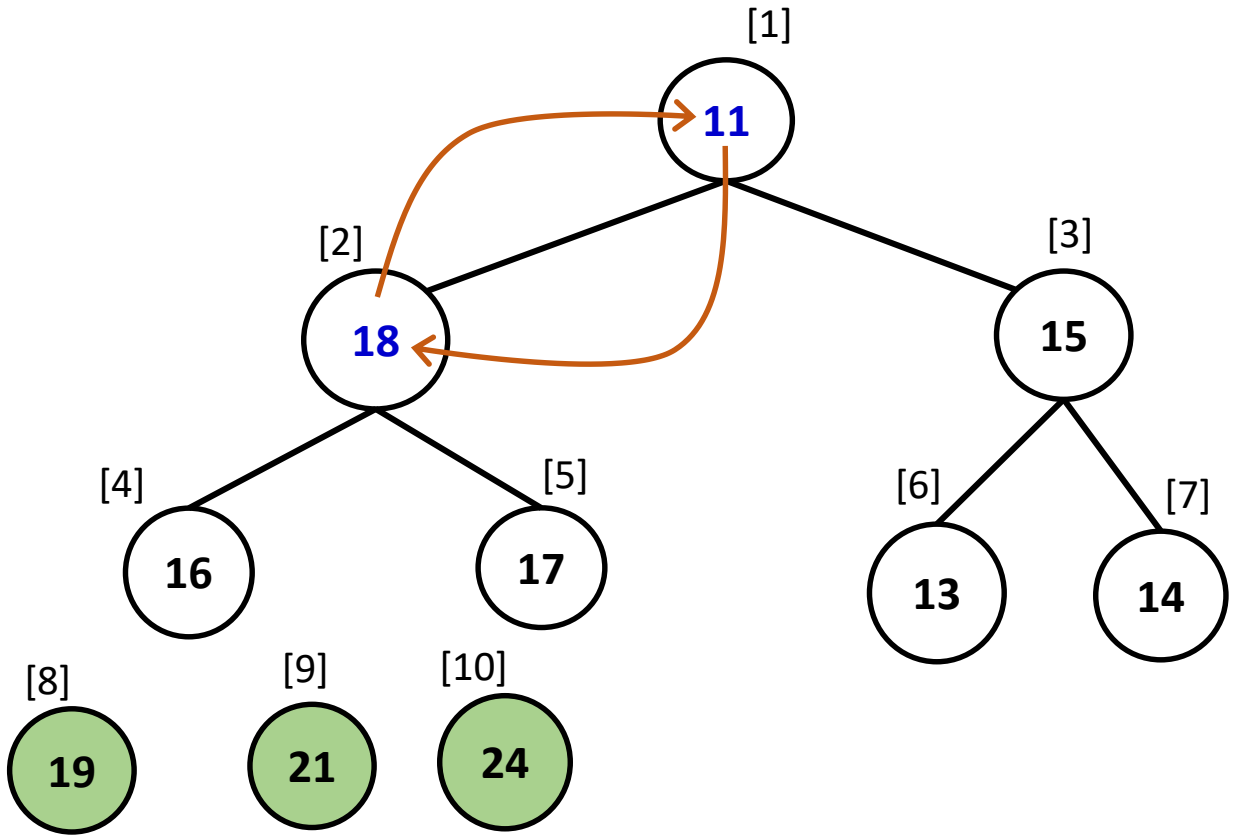


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**11**	**18**	**15**	**16**	**17**	**13**	**14**	**19**	**21**	**24**

Heapsort: An Example (14/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*

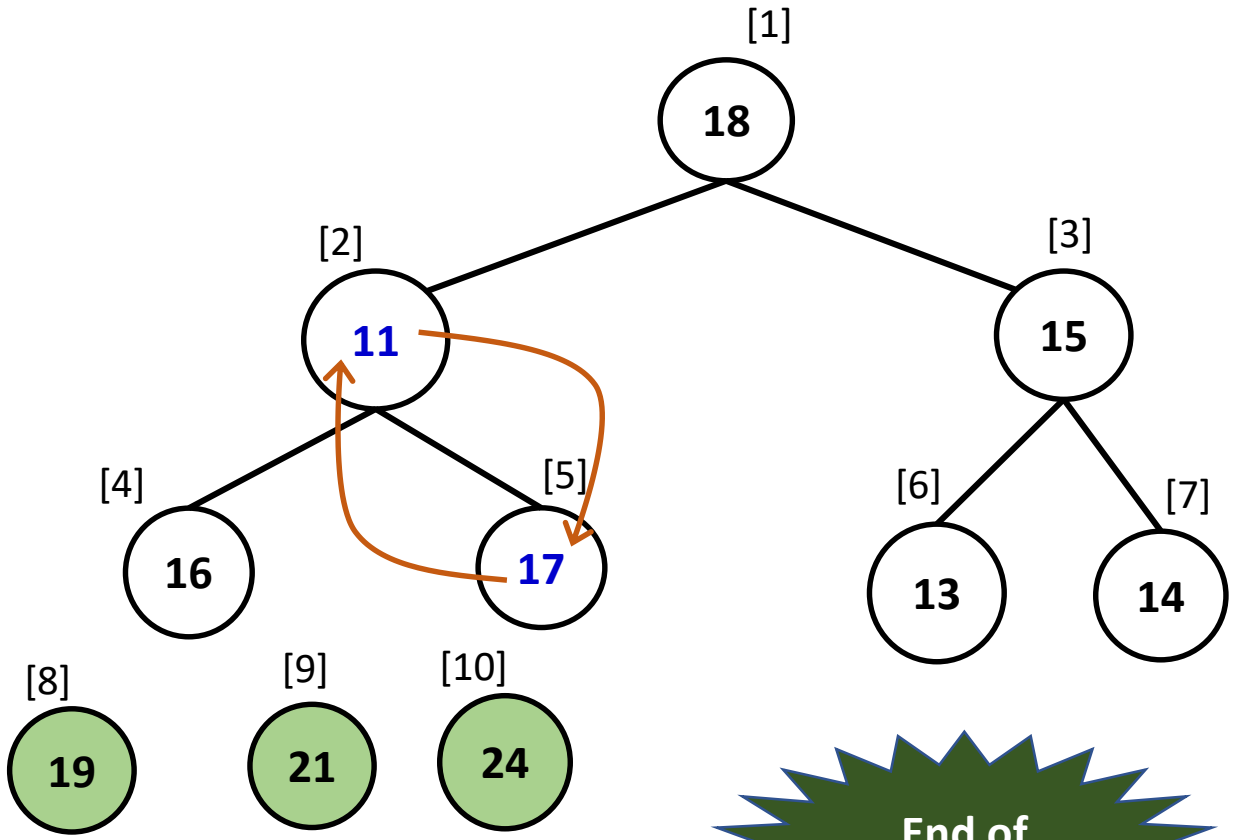


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**18**	**11**	**15**	**16**	**17**	**13**	**14**	**19**	**21**	**24**

Heapsort: An Example (15/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*



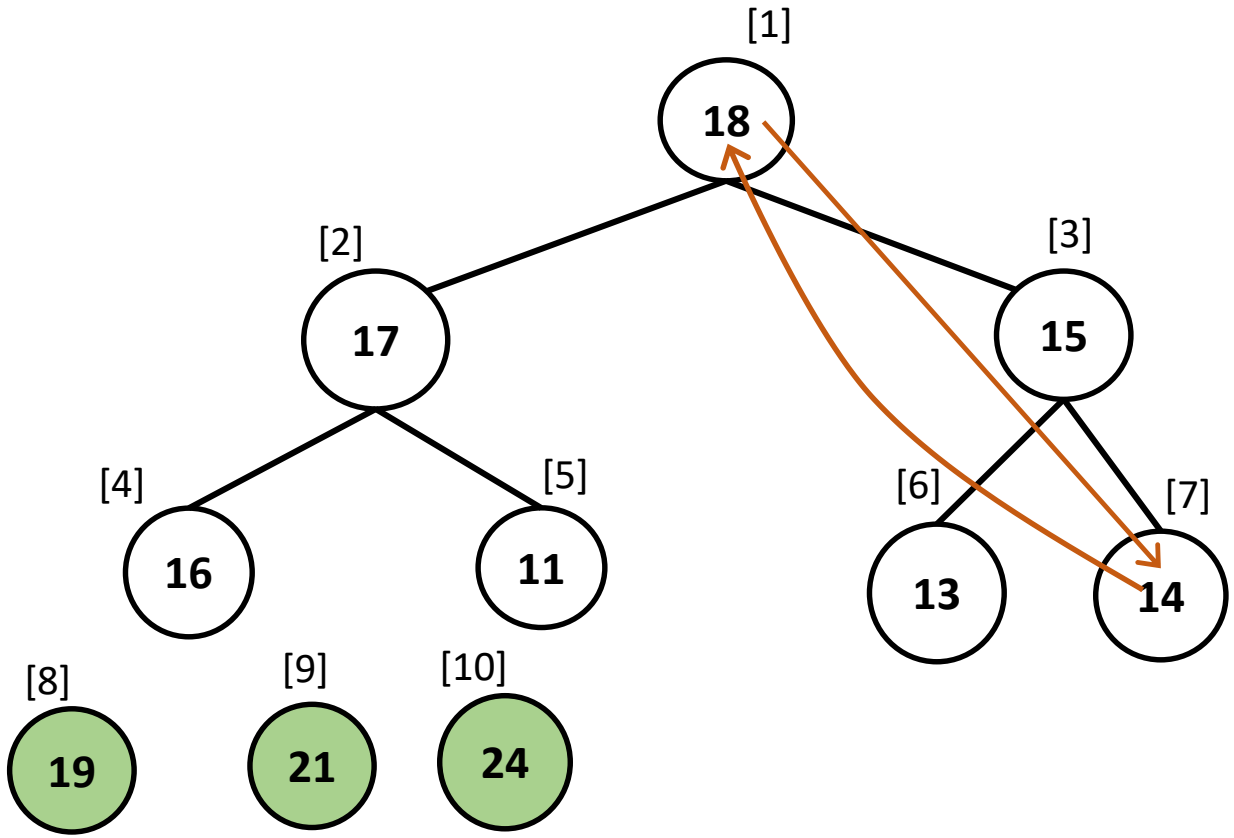
End of Iteration-3

	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	18	17	15	16	11	13	14	19	21	24

Heapsort: An Example (16/18)

HEAPSORT(A)

1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*

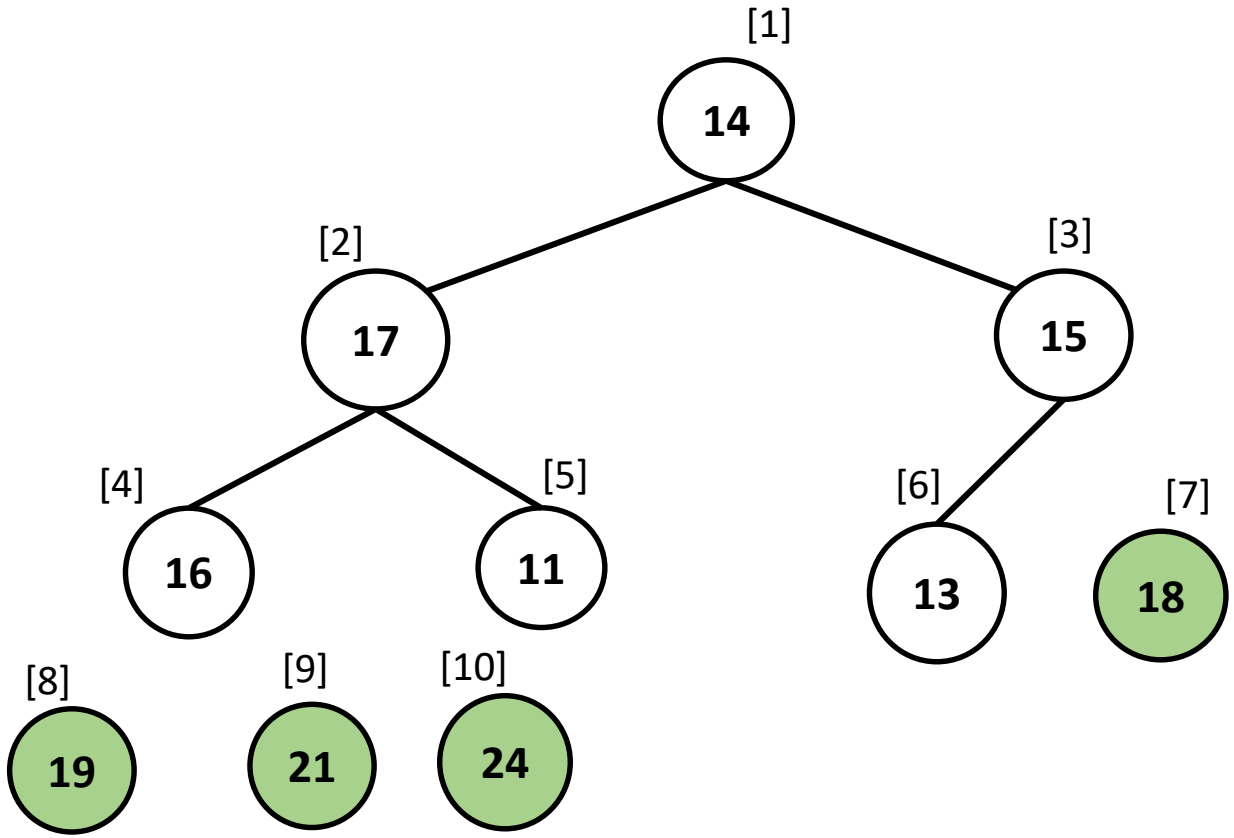


	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	**14**	**17**	**15**	**16**	**11**	**13**	**18**	**19**	**21**	**24**

Heapsort: An Example (17/18)

HEAPSORT(A)

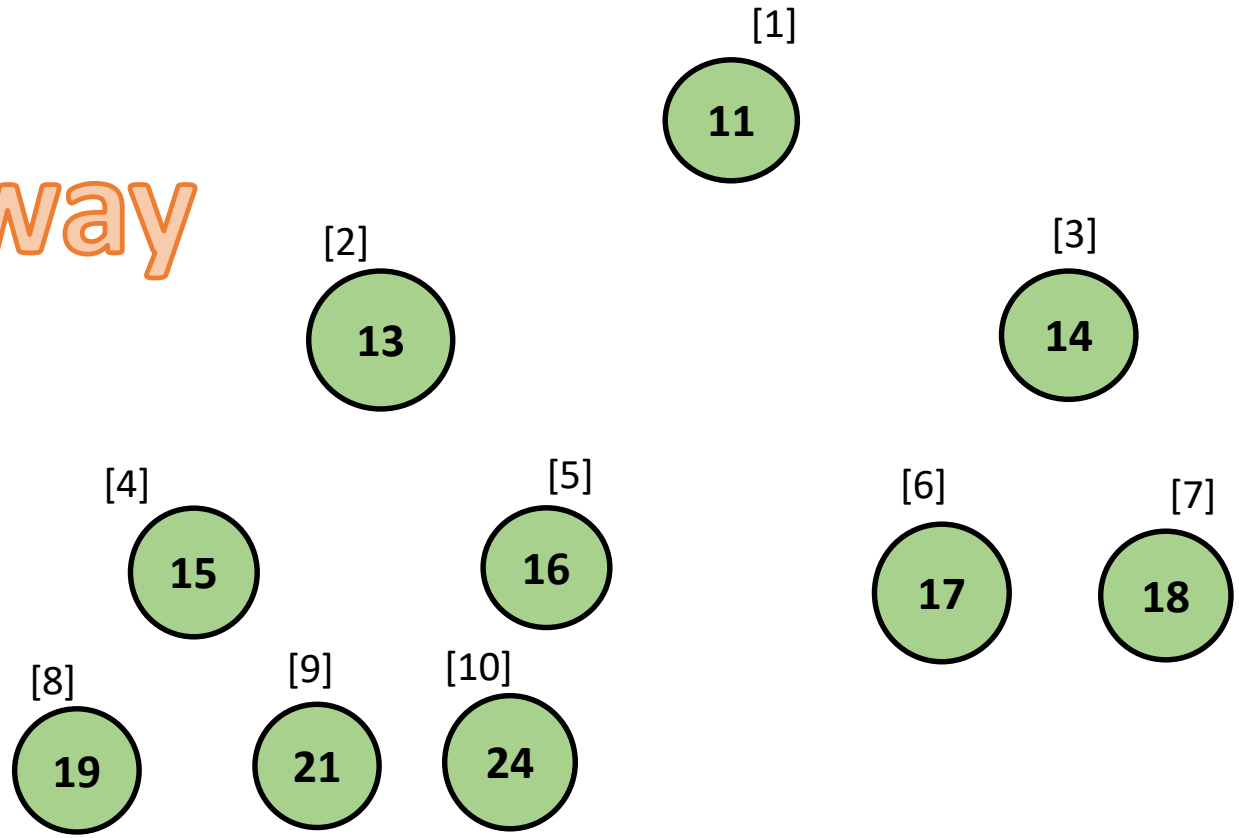
1. *BUILD-MAX-HEAP(A)*
2. *for i ← length of A down to 2*
3. *exchange A[1] ↔ A[i]*
4. *heapsize of A ← heapsize of A-1*
5. *MAX-HEAPIFY (A,1)*



	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	14	17	15	16	11	13	18	19	21	24

Heapsort: An Example (18/18)

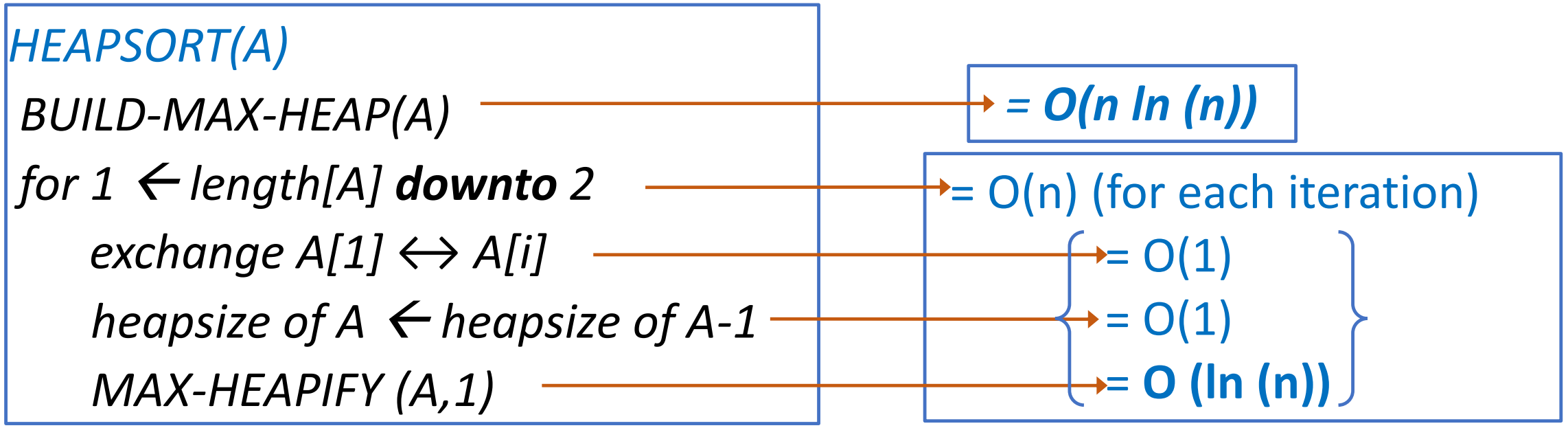
Continuing in this way



	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]	[10]
A	11	13	14	15	16	17	18	19	21	24

DONE

Heap Sort: Complexity



Overall Complexity: O(n ln (n))

One Major Use of Heap: Priority Queue

- A Heap is an efficient data structure to implement a priority queue
- Operations (A is the Max-Heap implemented in an array)
  1. INSERT (A, x)
  2. GET-MAX(A)
  3. REMOVE-MAX(A)

Priority Queue Operations: Remove and Max

```
GET-MAX(A)
    return A[1]
```

```
REMOVE-MAX(A)
1  if heapsize of A < 1 then
2      error "heap underflow"
3  max ← A[1]
4  A[1] ← A[heapsize of A]
5  heapsize of A ← heapsize of A - 1
6  MAX-HEAPIFY(A, 1)
7  return max
```

Priority Queue Operation: Insert

```
INSERT(A, x)
1    if heapsize of A < 1 then
2            A[1] ← x
3    else
4    heapsize ← heapsize of A + 1
5    A[heapsize] ← x
6    i ← heapsize
7    while (A[i] > parent(A[i]){
                swap(A[i], A[parent(A[i])])
                i ← parent(A[i])
            }
```

Heap: Applications

- **Heapsort**: One of the **best sorting methods** with in-place and with no quadratic worst-case scenarios

- **Selection algorithms**: Finding min, max, both min and max, median, or even the k^{th} largest element can be done in linear time (often constant time) using heaps

- **Graph algorithms**: By using heaps as internal traversal data structures, run time could be reduced by polynomial order. Examples: Prim's minimal spanning tree algorithm and Dijkstra's shortest path problem.

Thank you!