CS473 - Algorithms I

Lecture 8

Heapsort

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort

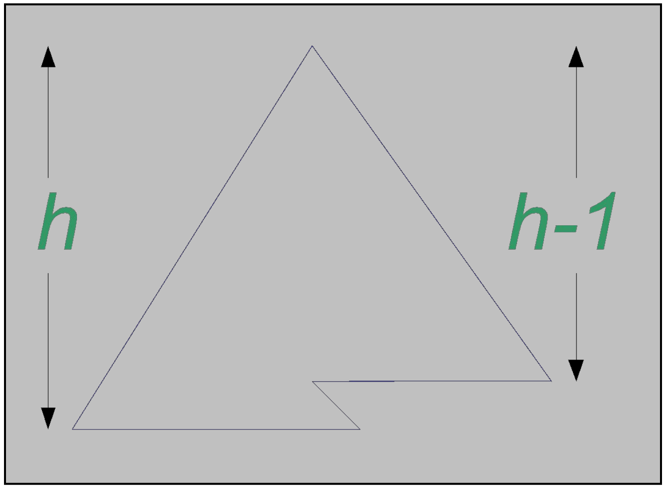
- Worst-case runtime: O(nlgn)
- Sorts in-place
- Uses a special data structure (heap) to manage information during execution of the algorithm
  - Another design paradigm

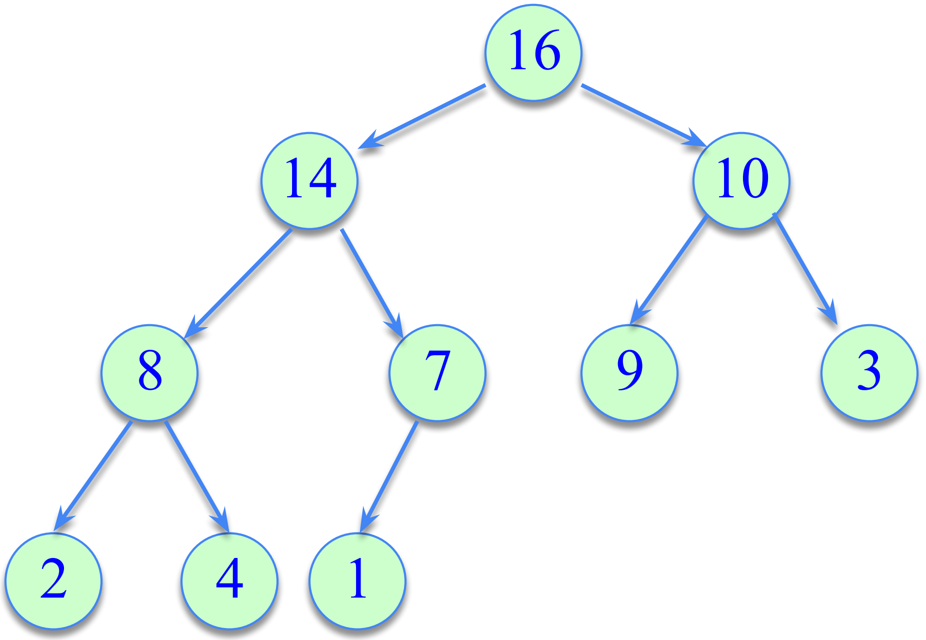
Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heap Data Structure

Refer to Data Structures slides for heap details!

The largest element in any subtree is the root element in a max-heap





Nearly complete binary tree

- Completely filled on all levels except possibly the lowest level

Assume max-heaps

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heap Operations

- Max(A, n) → O(1)
- Extract(A, n) → O(lg n)
    - Heapify(A, i, n) → O(lg n)
- Insert(A, key, n) → O(lg n)
- Build-Heap(A, n) → O(n) [why not O(n lg n)?]

- Min(A, n) → O(n)
- Search(A, key) → O(n)
- Heap-Increase-Key(A, i, key) → O(lg n)
- Heap-Decrease-Key(A, i, key) → O(lg n)

Summary: Max Heap

Max(A, n)

Returns the max element of the heap (no modification)

Runtime: O(1)

Heapify(A, i, n)

Works when both child subtrees of node i are heaps

"*Floats down*" node i to satisfy the heap property

Runtime: O(lgn)

Extract(A, n)

Returns and removes the max element of the heap

Fills the gap in A[1] with A[n], then calls Heapify(A,1)

Runtime: O(lgn)

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Summary: Max Heap

Build-Heap(A, n)

Given an arbitrary array, builds a heap from scratch

Runtime: O(n)

Min(A, n)

How to return the min element in a *max-heap*?

Worst case runtime: O(n)

because ~half of the heap elements are leaf nodes

Instead, use a *min-heap* for efficient min operations

Search(A, key)

For an arbitrary x value, the worst-case runtime: O(n)

Use a sorted array instead for efficient search operations

Summary: Max Heap

Increase-Key(A, i, x)

Increase the key of node i (from A[i] to x)

"Float up" x until heap property is satisfied

Runtime: O(lg n)

Decrease-Key(A, i, x)

Decrease the key of node i (from A[i] to x)

Call Heapify(A, i)

Runtime: O(lg n)

Heap Increase Key

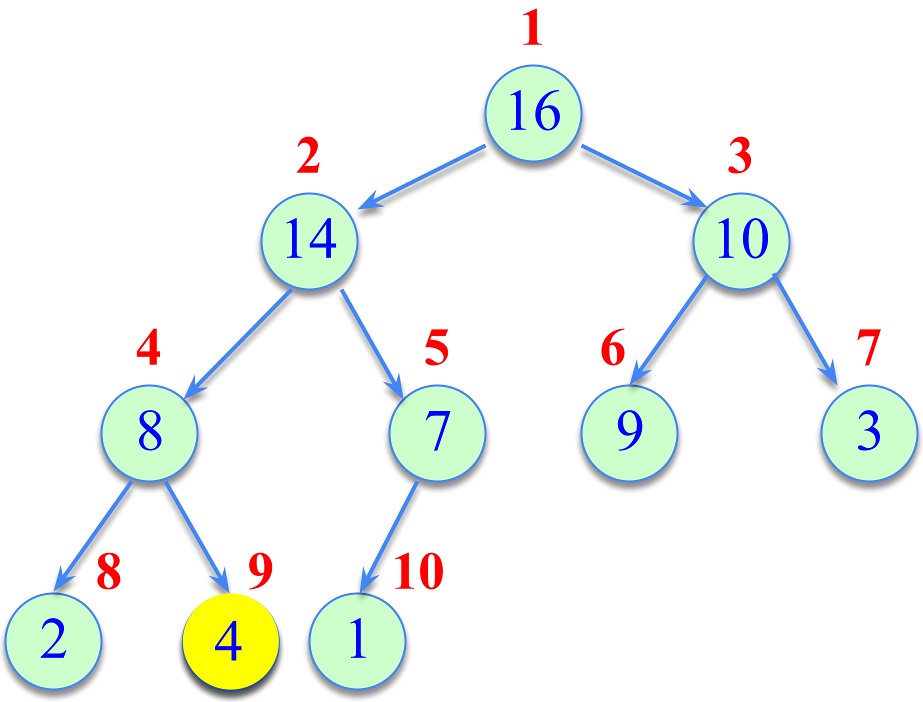
- Key value of i-th element of heap is increased from A[i] to *key*

HEAP-INCREASE-KEY(A, i, *key*)
    if *key* < A[i] **then**
        return error
    while $i > 1$ **and** A[$\lfloor i/2 \rfloor$] < key **do**
        A[i] $\leftarrow$ A[$\lfloor i/2 \rfloor$]
        $i \leftarrow \lfloor i/2 \rfloor$
    A[i] $\leftarrow$ *key*

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Example: HEAP-INCREASE-KEY(A, 9, 15)

HEAP-INCREASE-KEY(A, *i*, *key*)
    if *key* < A[*i*] **then**
        return error
    while *i* >1 **and** A[⌊*i*/2⌋] < key **do**
        A[*i*] ← A[⌊*i*/2⌋]
        i ← ⌊*i*/2⌋
    A[*i*] ← *key*



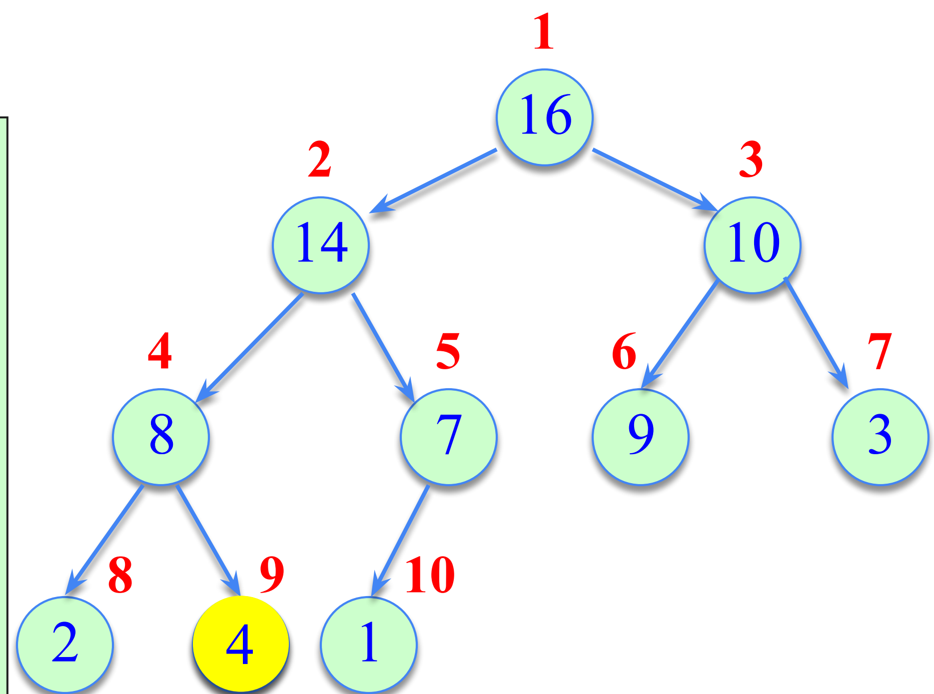
key = 15

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Example: HEAP-INCREASE-KEY(A, 9, 15)

HEAP-INCREASE-KEY(A, *i*, *key*)

    if *key* < A[*i*] **then**
        return error
    while *i* > 1 **and** A[⌊*i*/2⌋] < key **do**
        A[*i*] ← A[⌊*i*/2⌋]
        i ← ⌊*i*/2⌋
    A[*i*] ← *key*

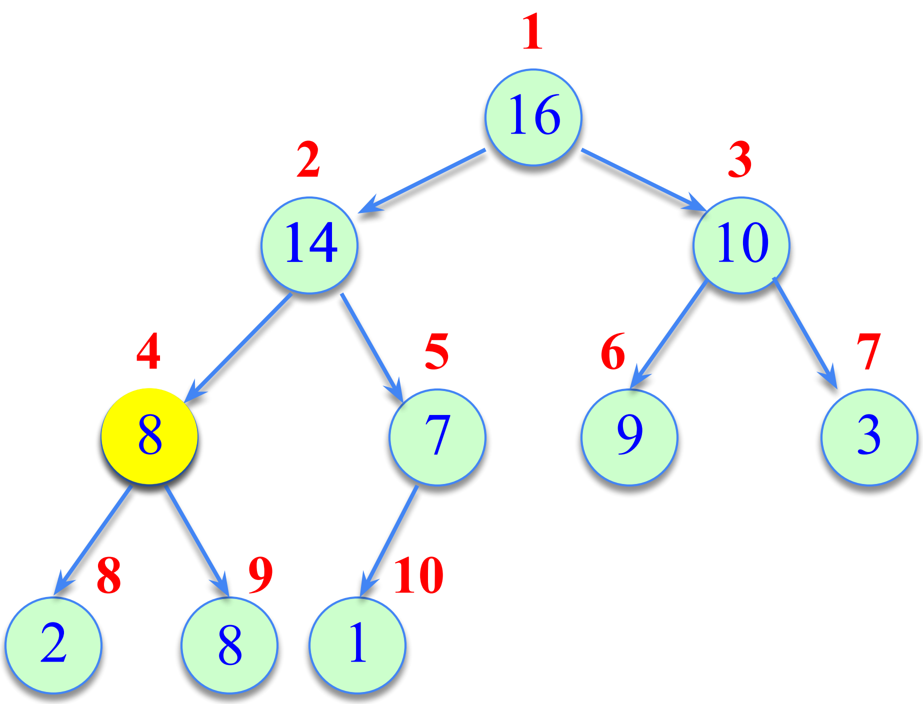


key = 15

Example: HEAP-INCREASE-KEY(A, 9, 15)

HEAP-INCREASE-KEY(A, *i*, *key*)

```
if key < A[i] then
    return error
while i > 1 and A[⌊i/2⌋] < key do
    A[i] ← A[⌊i/2⌋]
    i ← ⌊i/2⌋
A[i] ← key
```



key = 15

Example: HEAP-INCREASE-KEY(A, 9, 15)

HEAP-INCREASE-KEY(A, *i*, *key*)

    if *key* < A[*i*] **then**

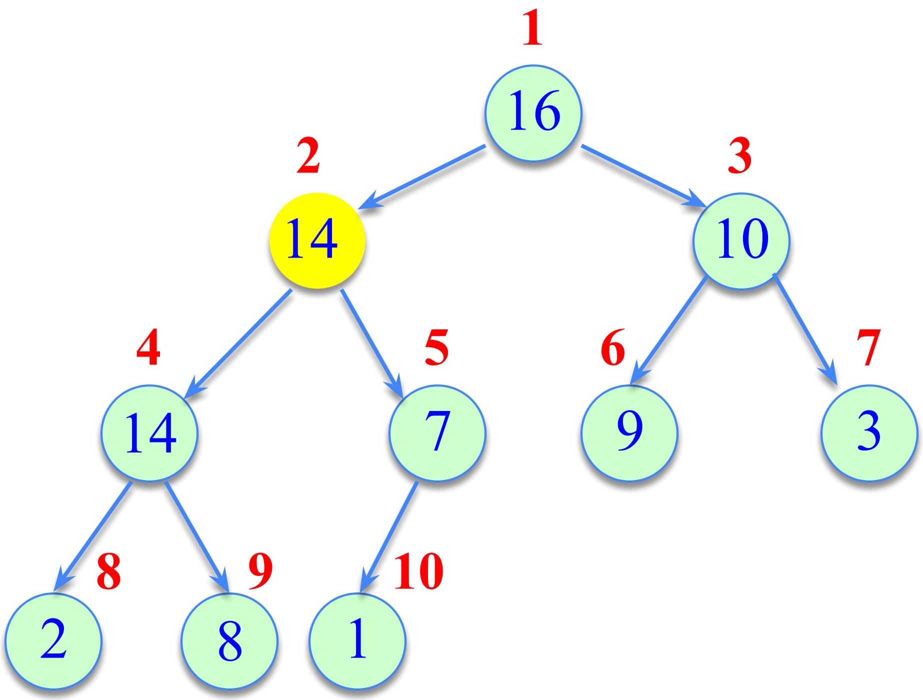
        return error

    while *i* >1 **and** A[⌊*i*/2⌋] < key **do**

        A[*i*] ← A[⌊*i*/2⌋]

        i ← ⌊*i*/2⌋

    A[*i*] ← *key*



key = 15

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Example: HEAP-INCREASE-KEY(A, 9, 15)

HEAP-INCREASE-KEY(A, *i*, *key*)

    if *key* < A[*i*] **then**

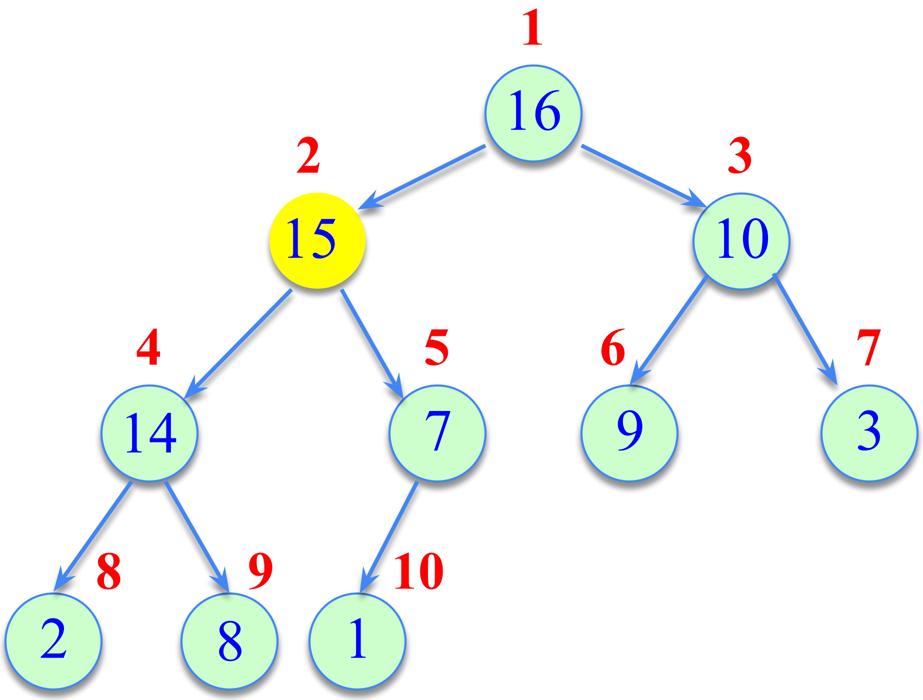
        return error

    while *i* >1 **and** A[⌊*i*/2⌋] < key **do**

        A[*i*] ← A[⌊*i*/2⌋]

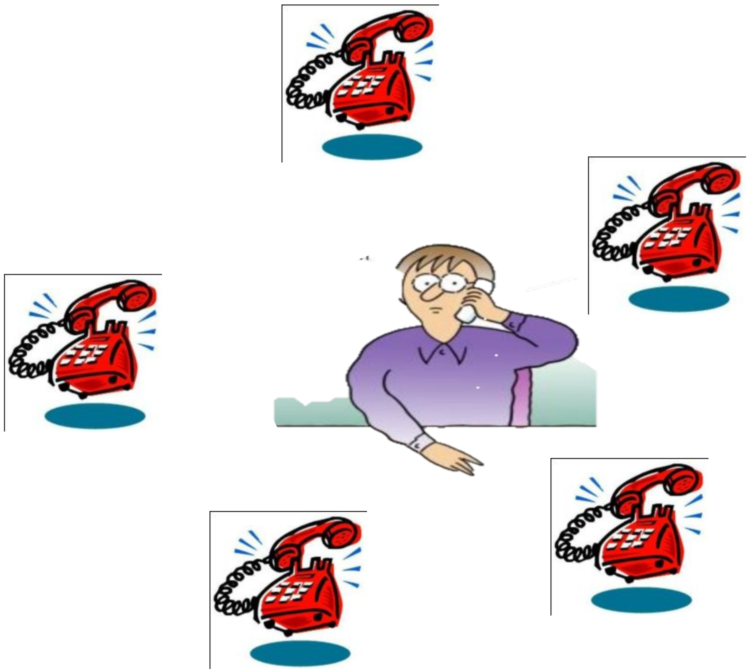
        i ← ⌊*i*/2⌋

    A[*i*] ← *key*



key = 15

Example Application: Phone Operator



A phone operator answering n phones

Each phone i has x_i people waiting in line for their calls to be answered.

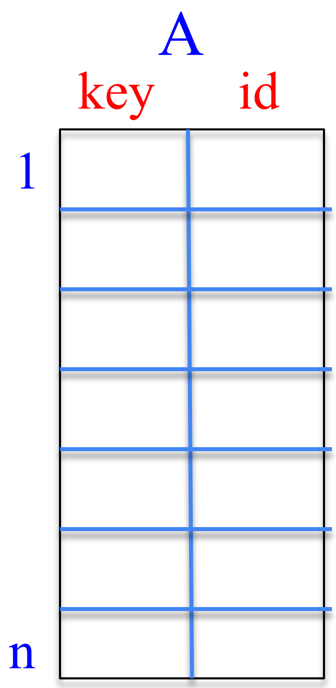
Phone operator needs to answer the phone with the largest number of people waiting in line.

New calls come continuously, and some people hang up after waiting.

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Solution

Step 1: Define the following array:



A[i]: the i^{th} element in heap

A[i].id: the index of the corresponding phone

A[i].key: # of people waiting in line for phone with index A[i].id

Solution

Step 2: Build-Max-Heap (A, n)

Execution:

When the operator wants to answer a phone:

id = A[1].id

Decrease-Key(A, 1, A[1].key-1)

answer phone with index id

When a new call comes in to phone i:

Increase-Key(A, i, A[i].key+1)

When a call drops from phone i:

Decrease-Key(A, i, A[i].key-1)

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

(1) Build a heap on array A[1…n] by calling BUILD-HEAP(A, n)

(2) The largest element is stored at the root A[1]

Put it into its correct final position A[n] by A[1] ↔ A[n]

(3) Discard node n from the heap

(4) Subtrees (S_2 & S_3) rooted at children of root remain as heaps

but the new root element may violate the heap property

Make A[1…$n-1$] a heap by calling HEAPIFY(A, 1, $n-1$)

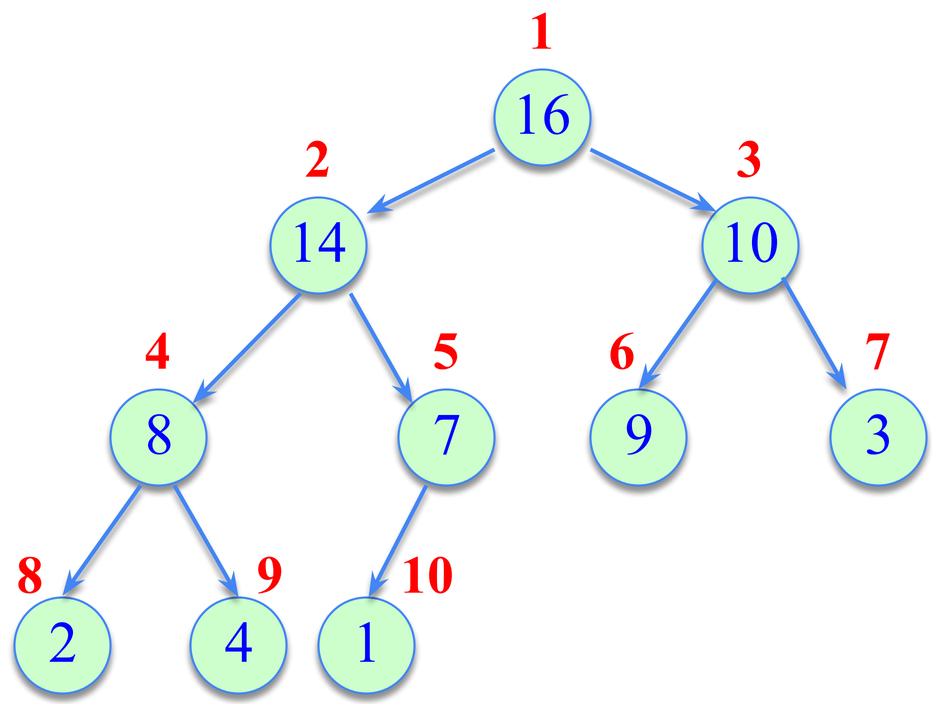
(5) $n \leftarrow n - 1$

(6) Repeat steps 2−4 until $n = 2$

Heapsort Algorithm

HEAPSORT(A, n)

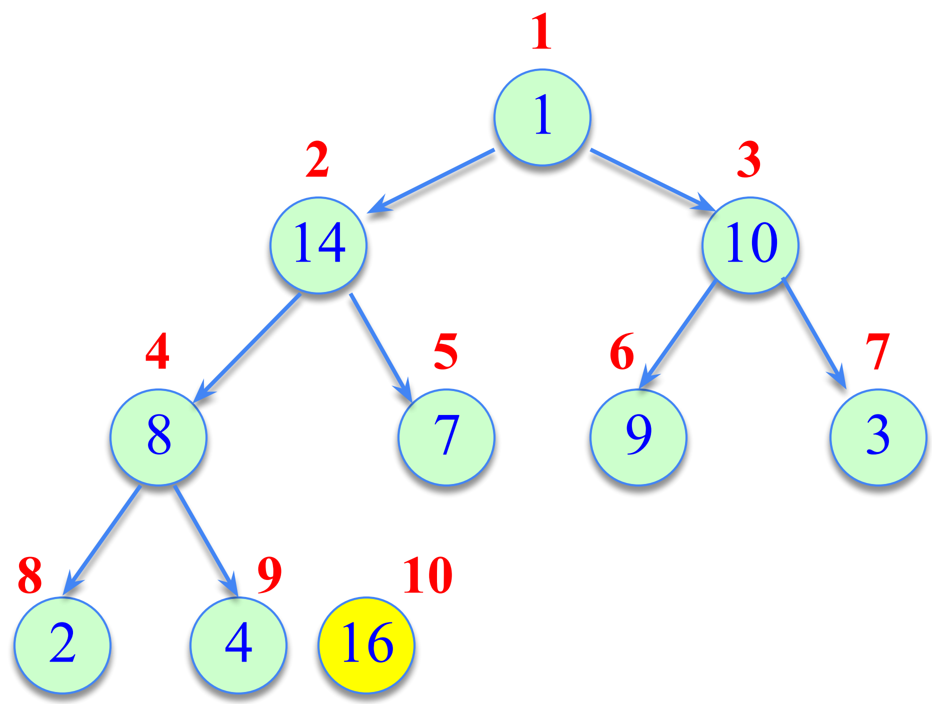
```
    BUILD-HEAP(A, n)
    for i ← n downto 2 do
        exchange A[1] ↔ A[i]
        HEAPIFY(A, 1, i −1)
```



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

Heapsort Algorithm

HEAPSORT(A, n)
BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
  exchange A[1] $\leftrightarrow$ A[i]
  HEAPIFY(A, 1, i −1)



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

Heapsort Algorithm

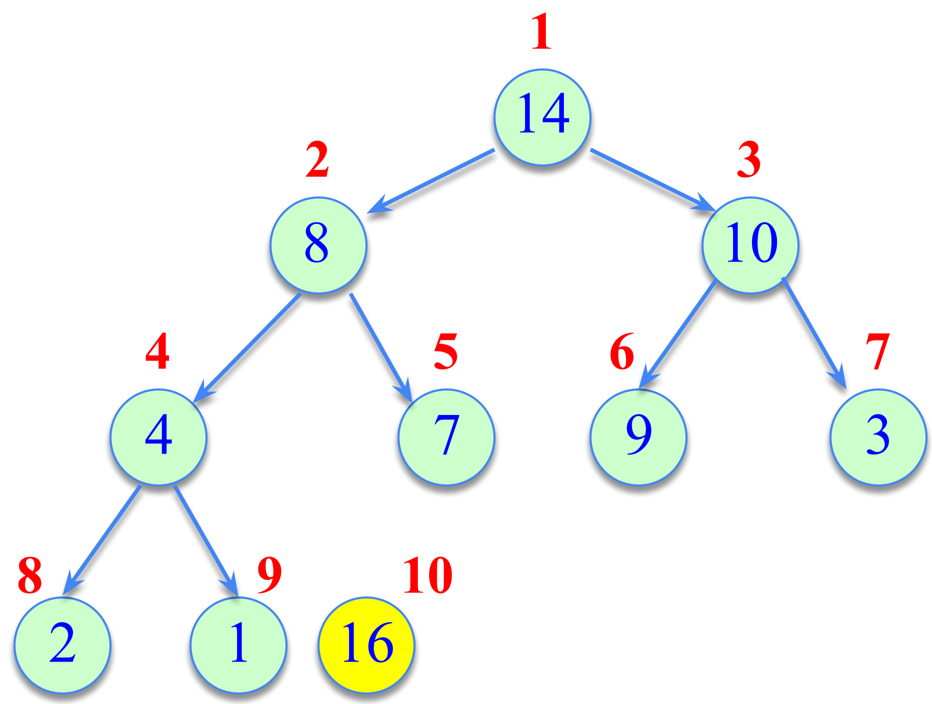
HEAPSORT(A, n)

BUILD-HEAP(A, n)

for $i \leftarrow n$ **downto** 2 **do**

exchange A[1] ↔ A[i]

HEAPIFY(A, 1, i −1)



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

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

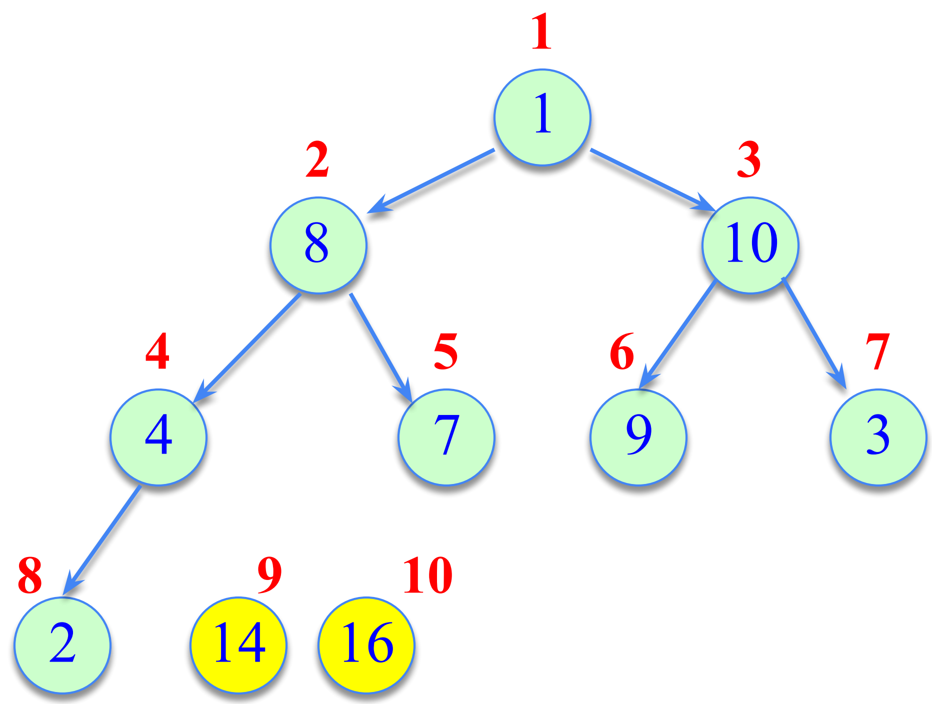
HEAPSORT(A, n)

BUILD-HEAP(A, n)

for $i \leftarrow n$ **downto** 2 **do**

exchange A[1] ↔ A[i]

HEAPIFY(A, 1, i −1)



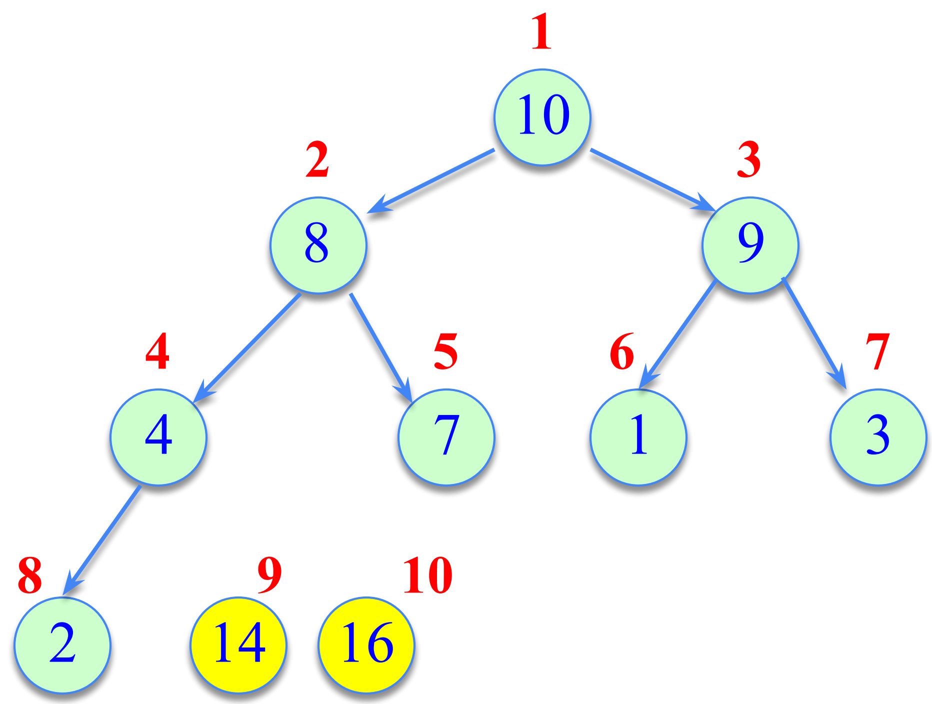
	1	2	3	4	5	6	7	8	9	10
A	1	8	10	4	7	9	3	2	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
  exchange A[1] ↔ A[i]
  HEAPIFY(A, 1, i −1)



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

Heapsort Algorithm

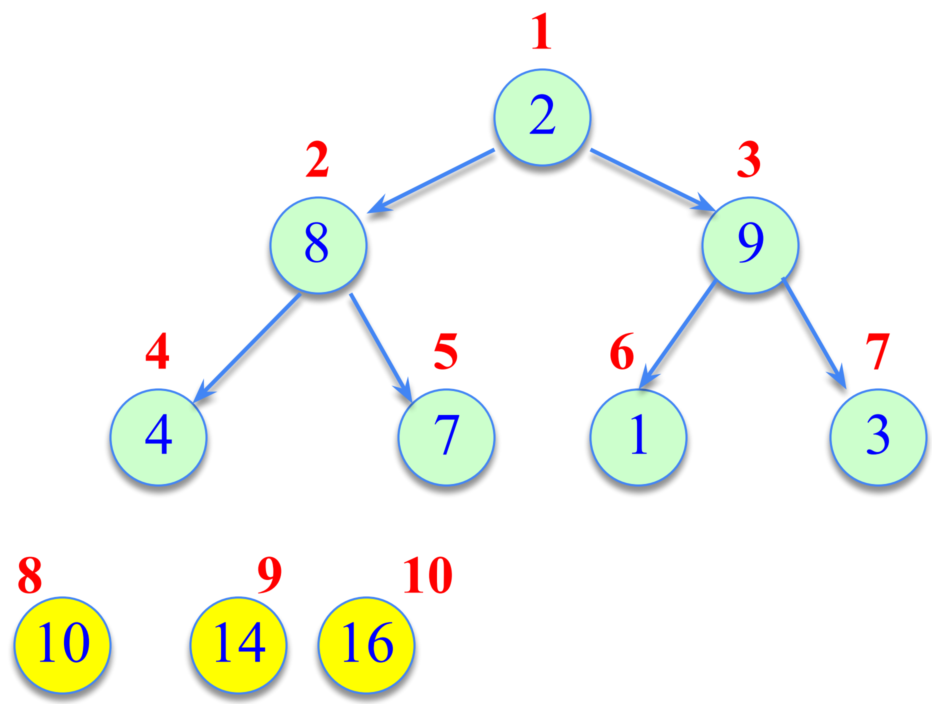
HEAPSORT(A, n)

BUILD-HEAP(A, n)

for $i \leftarrow n$ **downto** 2 **do**

exchange A[1] $\leftrightarrow$ A[i]

HEAPIFY(A, 1, i −1)



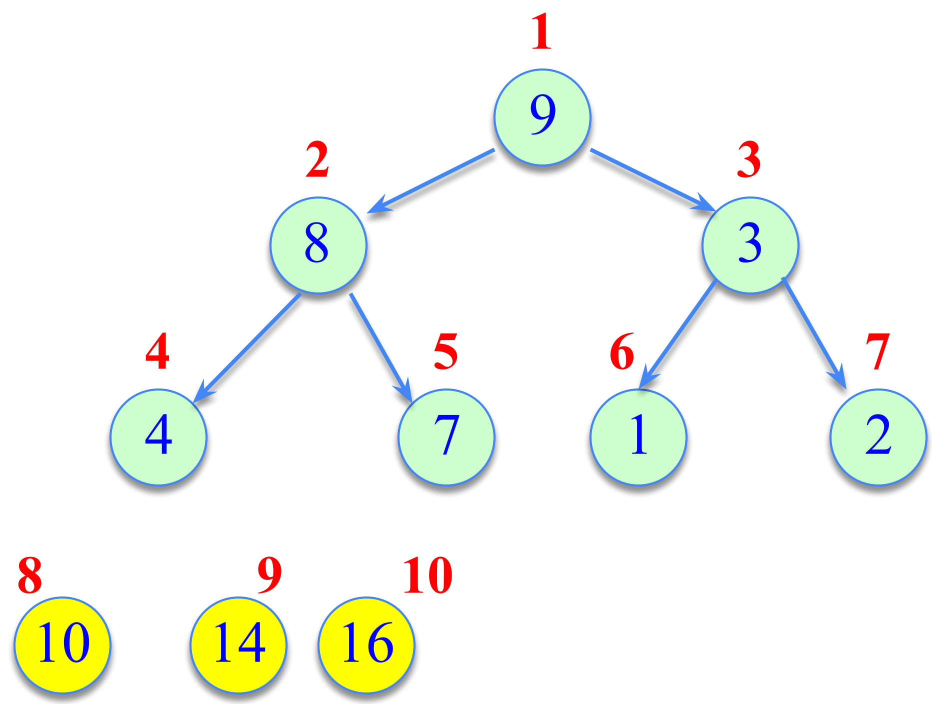
	1	2	3	4	5	6	7	8	9	10
A	2	8	9	4	7	1	3	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
  exchange A[1] ↔ A[i]
  HEAPIFY(A, 1, i −1)

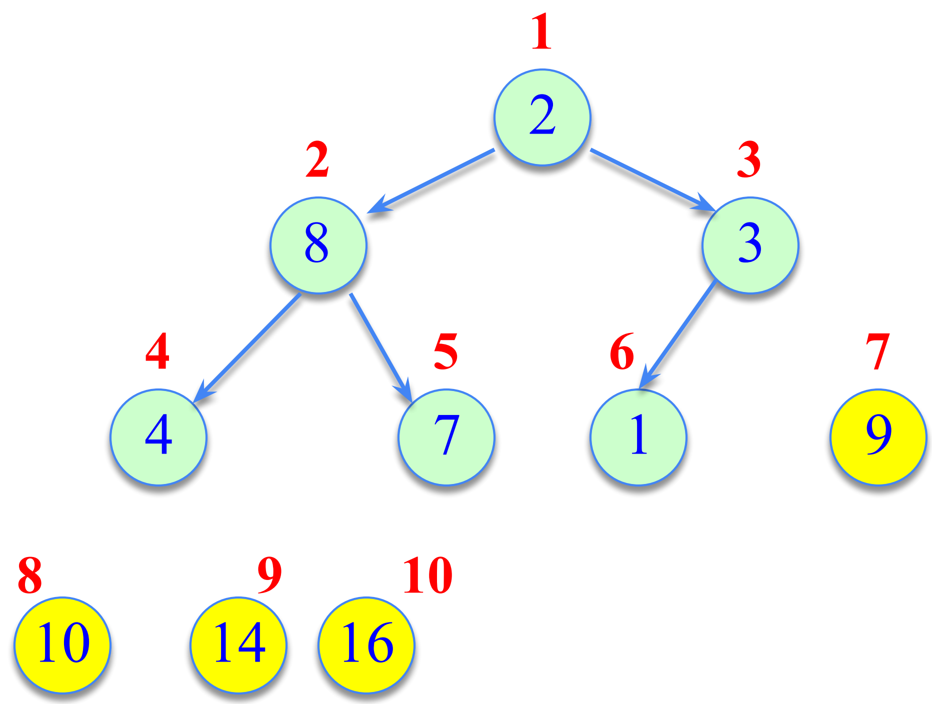


	1	2	3	4	5	6	7	8	9	10
A	9	8	3	4	7	1	2	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)
BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
exchange A[1] ↔ A[i]
HEAPIFY(A, 1, $i - 1$)



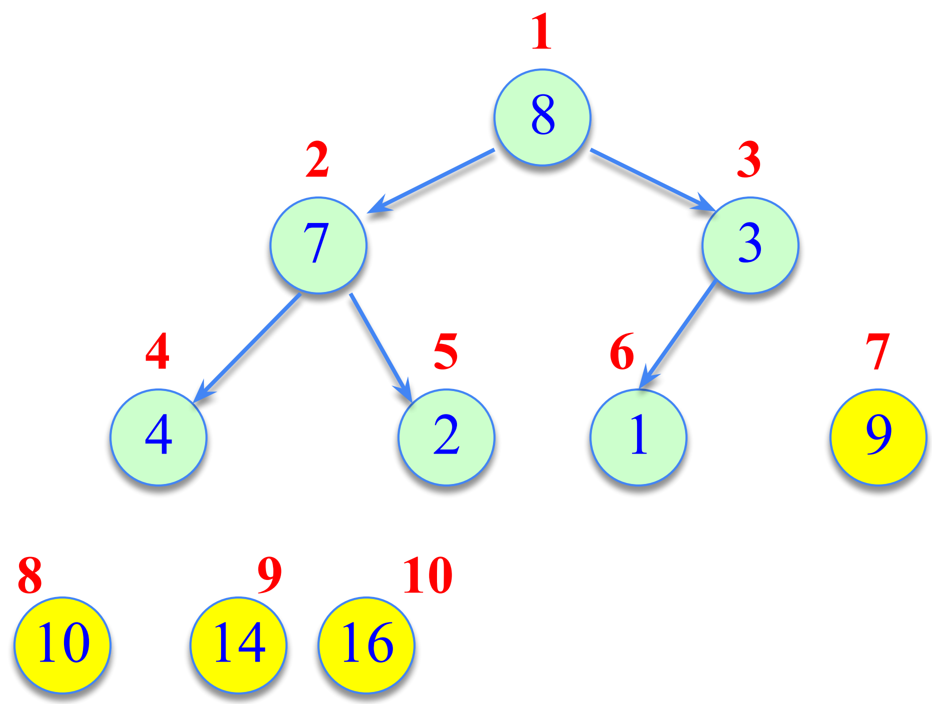
	1	2	3	4	5	6	7	8	9	10
A	2	8	3	4	7	1	9	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

```
BUILD-HEAP(A, n)
for i ← n downto 2 do
    exchange A[1] ↔ A[i]
    HEAPIFY(A, 1, i −1)
```



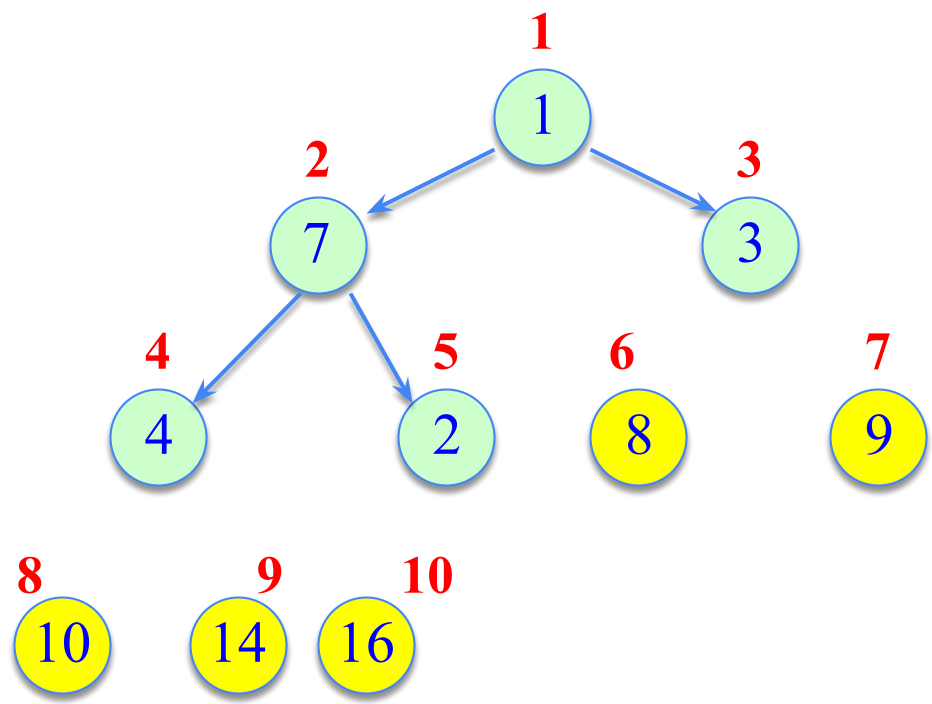
	1	2	3	4	5	6	7	8	9	10
A	8	7	3	4	2	1	9	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
  exchange A[1] ↔ A[i]
  HEAPIFY(A, 1, i −1)

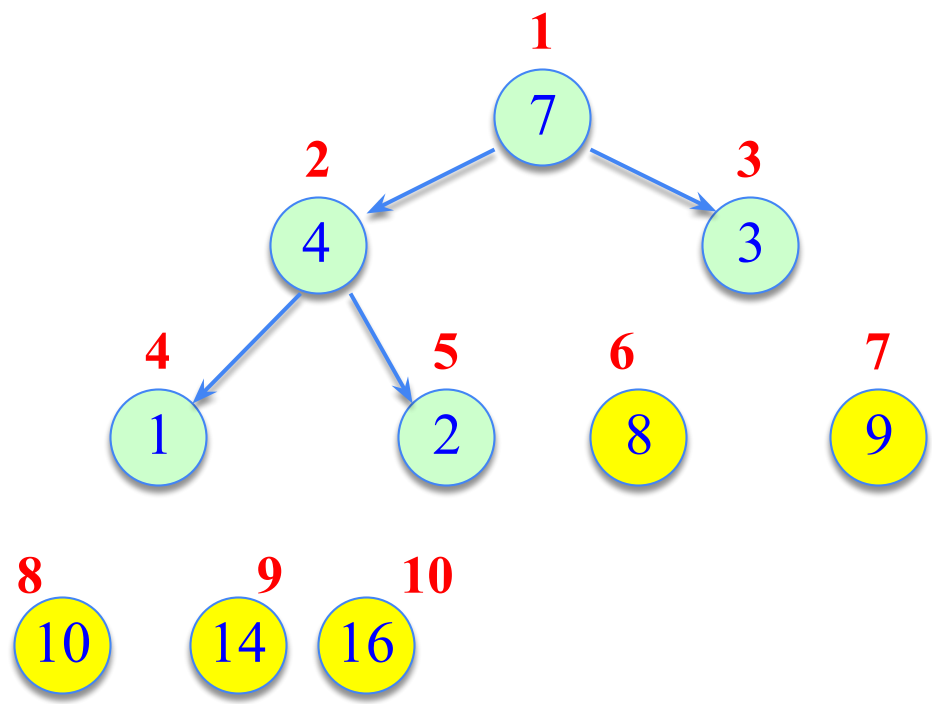


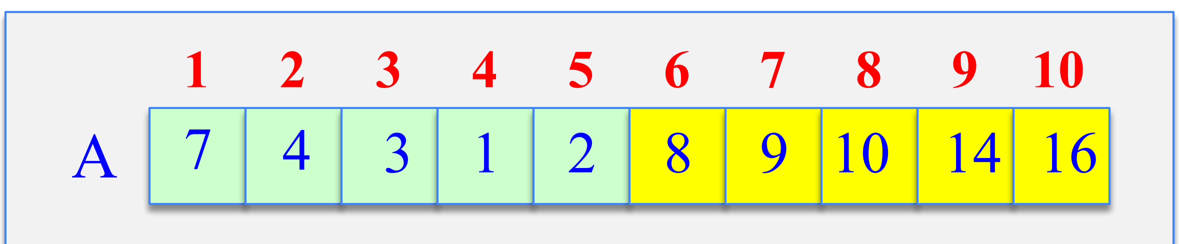
	1	2	3	4	5	6	7	8	9	10
A	1	7	3	4	2	8	9	10	14	16

Heapsort Algorithm

HEAPSORT(A, n)

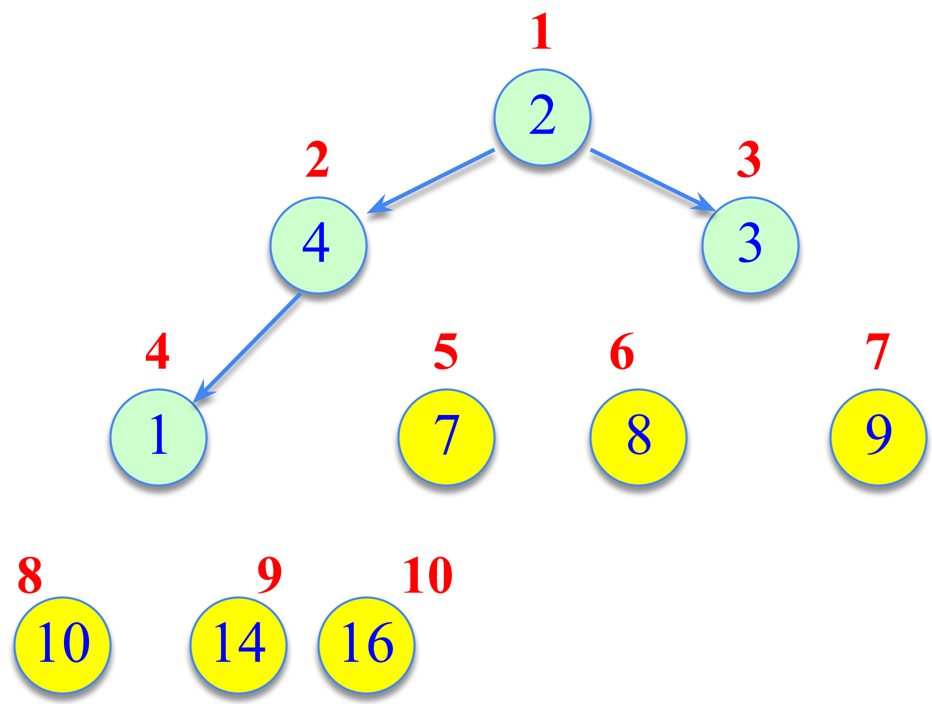
BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
exchange A[1] ↔ A[i]
HEAPIFY(A, 1, i −1)





Heapsort Algorithm

HEAPSORT(A, n)
BUILD-HEAP(A, *n*)
for *i* ← *n* **downto** 2 **do**
exchange A[1] ↔ A[*i*]
HEAPIFY(A, 1, *i* −1)



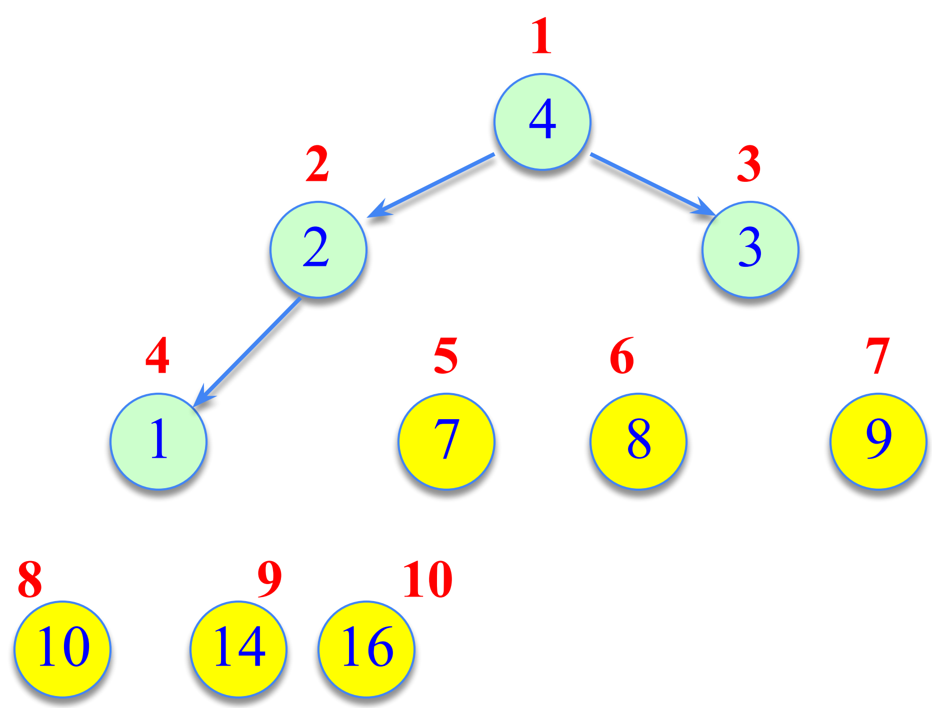
	1	2	3	4	5	6	7	8	9	10
A	2	4	3	1	7	8	9	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
  exchange A[1] ↔ A[i]
  HEAPIFY(A, 1, i −1)



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

Heapsort Algorithm

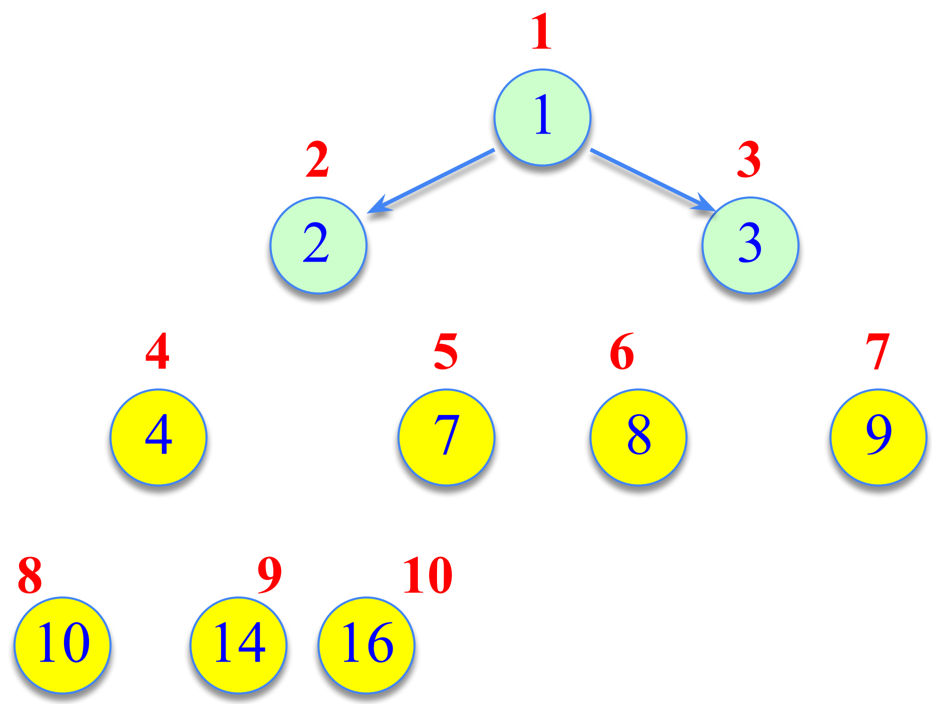
HEAPSORT(A, n)

BUILD-HEAP(A, n)

for $i \leftarrow n$ **downto** 2 **do**

exchange A[1] ↔ A[i]

HEAPIFY(A, 1, i −1)



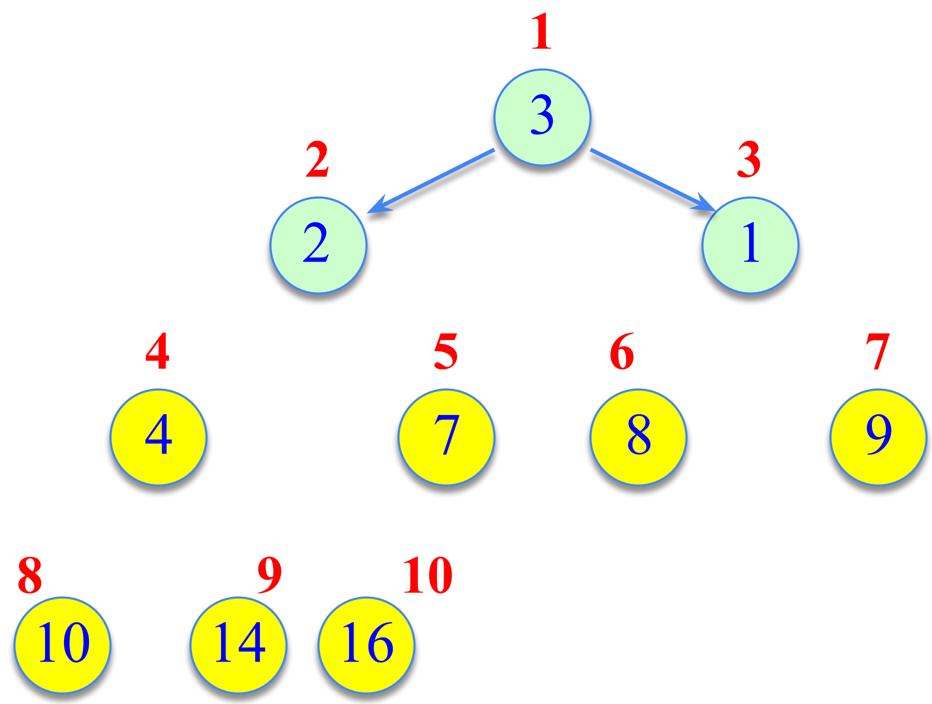
	1	2	3	4	5	6	7	8	9	10
A	1	2	3	4	7	8	9	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
  exchange A[1] ↔ A[i]
  HEAPIFY(A, 1, i −1)



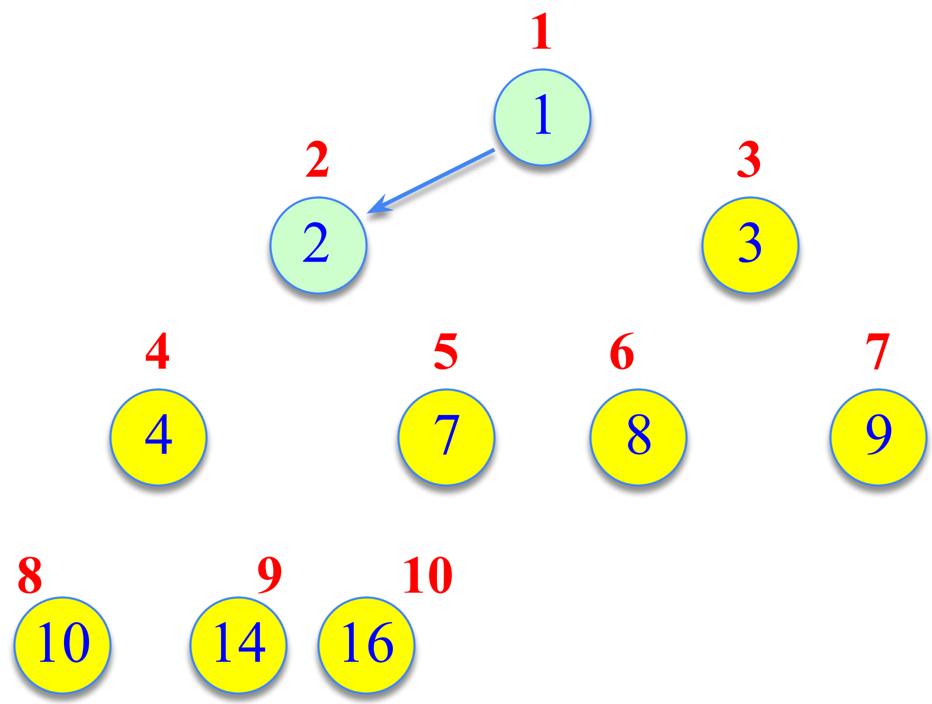
	1	2	3	4	5	6	7	8	9	10
A	3	2	1	4	7	8	9	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

```
BUILD-HEAP(A, n)
for i ← n downto 2 do
    exchange A[1] ↔ A[i]
    HEAPIFY(A, 1, i −1)
```



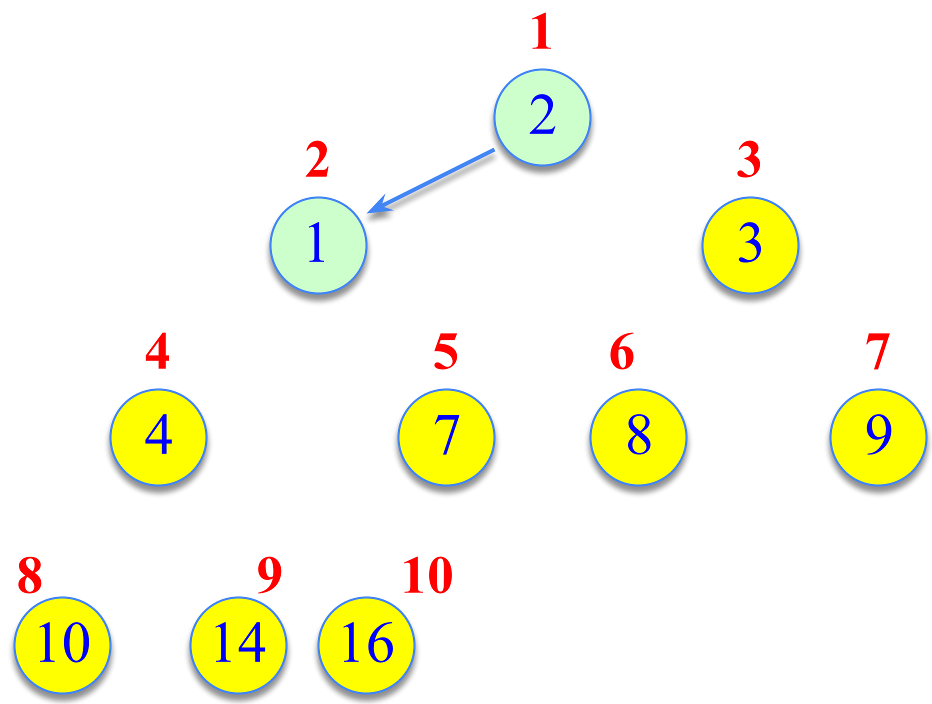
	1	2	3	4	5	6	7	8	9	10
A	1	2	3	4	7	8	9	10	14	16

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm

HEAPSORT(A, n)

BUILD-HEAP(A, n)
for $i \leftarrow n$ **downto** 2 **do**
  exchange A[1] ↔ A[i]
  HEAPIFY(A, 1, $i-1$)



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

Heapsort Algorithm

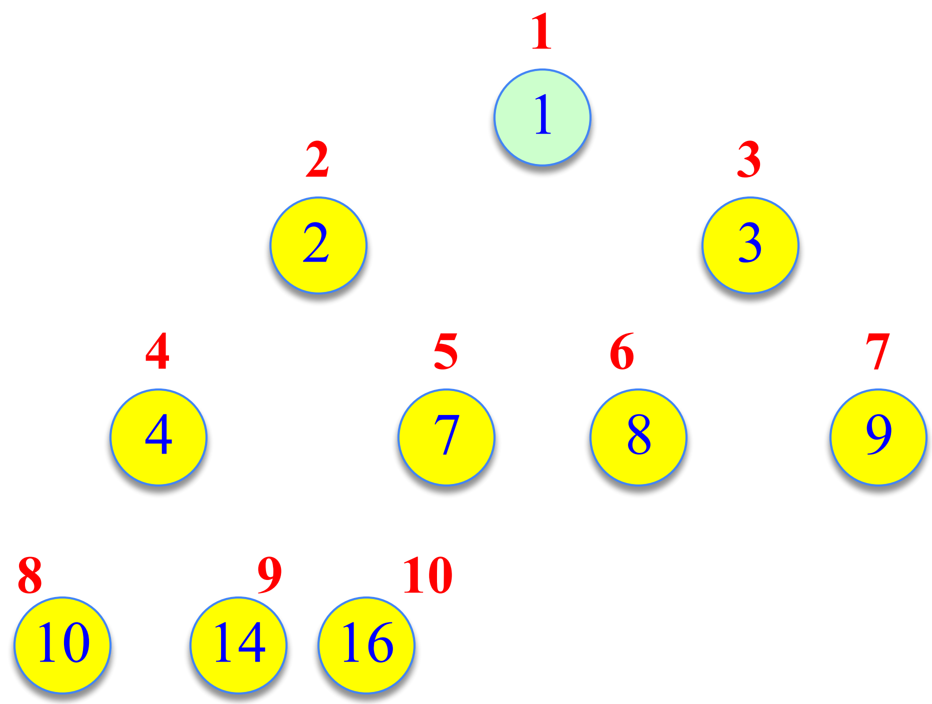
HEAPSORT(A, n)

BUILD-HEAP(A, n)

for $i \leftarrow n$ **downto** 2 **do**

exchange A[1] $\leftrightarrow$ A[i]

HEAPIFY(A, 1, i −1)

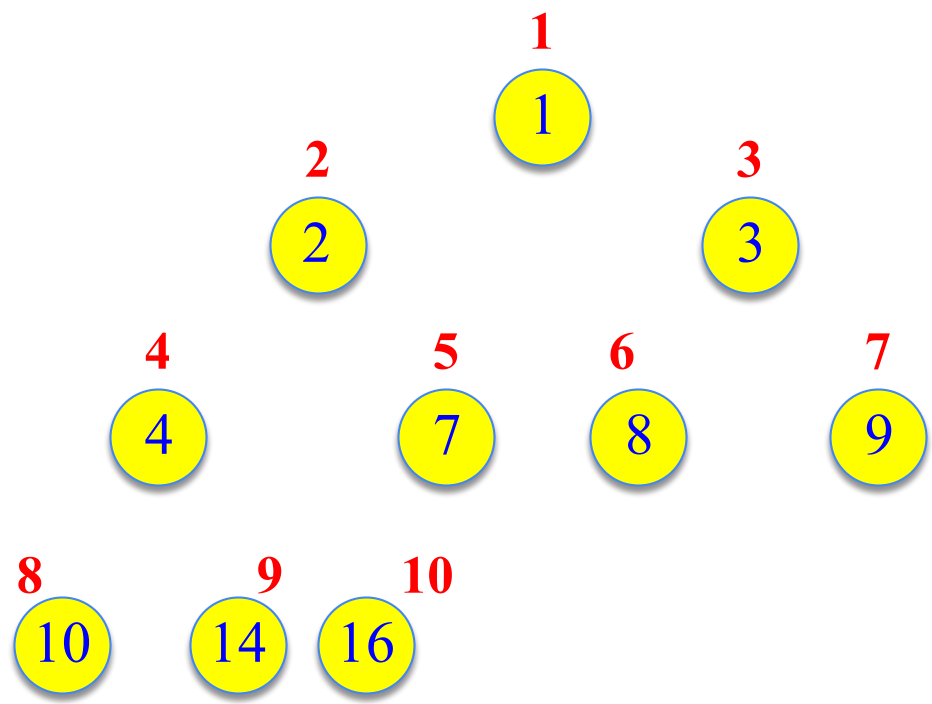


	1	2	3	4	5	6	7	8	9	10
A	1	2	3	4	7	8	9	10	14	16

Heapsort Algorithm

HEAPSORT(A, n)

    BUILD-HEAP(A, n)
    for $i \leftarrow n$ downto 2 do
        exchange A[1] ↔ A[i]
        HEAPIFY(A, 1, i −1)



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

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm: Runtime Analysis

HEAPSORT(A, n)

BUILD-HEAP(A, n) ---------------- $\Theta(n)$

for $i \leftarrow n$ **downto** 2 **do**

exchange A[1] $\leftrightarrow$ A[i] ---------------- $\Theta(1)$

HEAPIFY(A, 1, $i-1$) ---------------- O(lg(i-1))

$$T(n) = \Theta(n) + \sum_{i=2}^{n} O(\lg i) = \Theta(n) + O\left(\sum_{i=2}^{n} O(\lg n) \right) = O(n \lg n)$$

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University

Heapsort Algorithm: Performance

- Heapsort is a very good algorithm but, a good implementation of quicksort always beats heapsort in practice
- However, heap data structure has many popular applications, and it can be efficiently used for implementing priority queues

Cevdet Aykanat and Mustafa Ozdal
Computer Engineering Department, Bilkent University