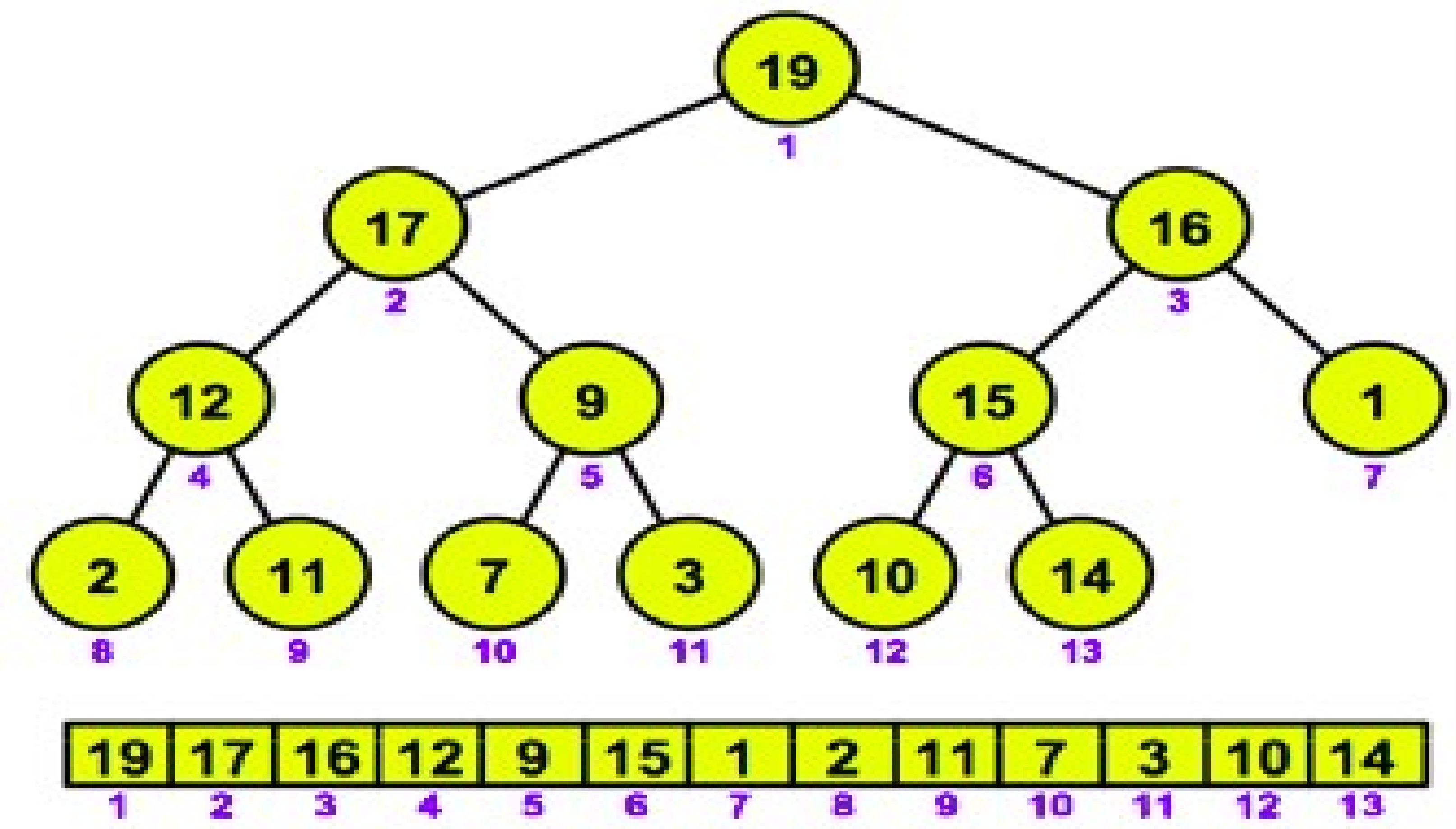
Heapsort



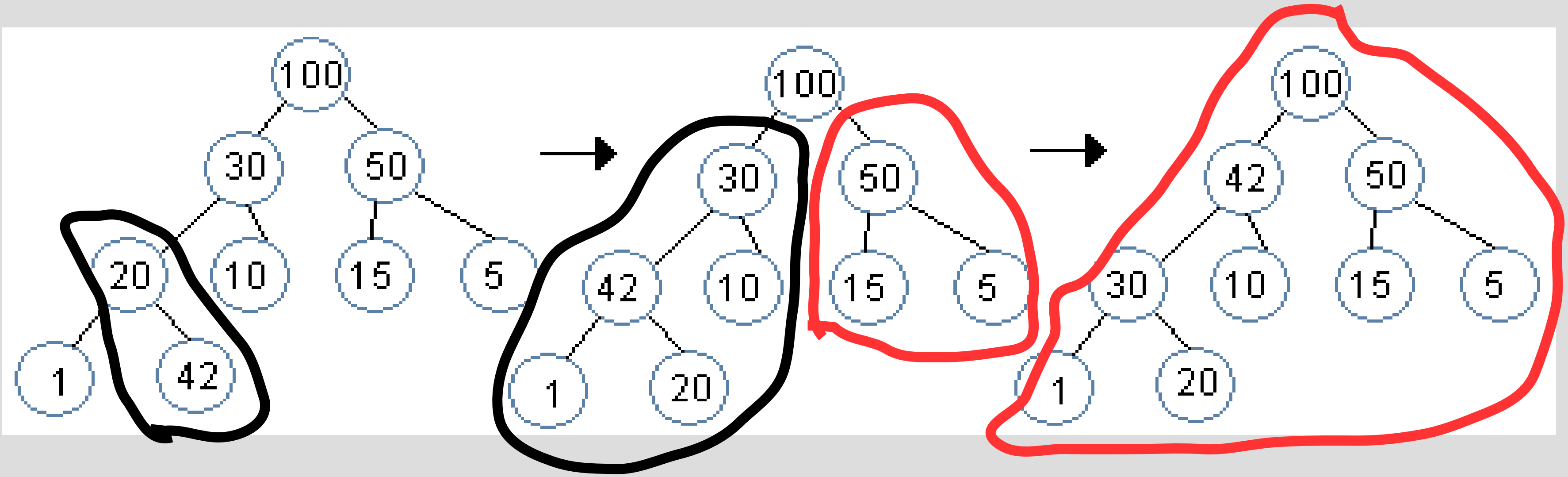
19
1
17
2
16
3
12
4
9
5
15
6
1
7
2
8
11
9
7
10
3
11
10
12
14
13
19 17 16 12 9 15 1 2 11 7 3 10 14
1 2 3 4 5 6 7 8 9 10 11 12 13

Build-Max-Heap

Next we build a full heap from an unsorted sequence

```
Build-Max-Heap(A)
for i = floor( A.length/2 ) to 1
   Max-Heapify(A, i)
```

Build-Max-Heap



Red part is already Heapified

Build-Max-Heap

Correctness:

Base: Each alone leaf is a max-heap

Step: if A[i] to A[n] are in a heap, then Heapify(A, i-1) will make i-1 a heap as well

Termination: loop ends at i=1, which is the root (so all heap)

Build-Max-Heap

Runtime?

Build-Max-Heap

Runtime?

O(n lg n) is obvious, but we can get a better bound...

Show ceiling($n/2^{h+1}$) nodes at any level 'h', with h=0 as bottom

Build-Max-Heap

Heapify from height 'h' takes O(h)

$$\sum_{h=0}^{\lceil \ln n \rceil} \lceil n/2^{h+1} \cdot O(h) \rceil$$

$$= O(n \cdot \sum_{h=0}^{\lceil \ln n \rceil} \lceil h/2^{h+1} \rceil)$$

$$\leq O(n \cdot \sum_{h=0}^{\infty} h/2^h)$$

$$Note : \sum_{k=0}^{\infty} k \cdot c^k = c/(1-c)^2 \ldots \text{ for us } c = \frac{1}{2}$$

$$= O(n \cdot 0.5/(1-0.5)^2) = O(2n) = O(n)$$

Heapsort

```
Heapsort(A):
Build-Max-Heap(A)
for i = A.length to 2
   Swap A[ 1 ], A[ i ]
   A.heapsize = A.heapsize – 1
   Max-Heapify(A, 1)
```

Heapsort

You try it!

Sort: A = [1, 6, 8, 4, 7, 3, 4]

Heapsort

First, build the heap starting here

A = [1, 6, 8, 4, 7, 3, 4]

A = [1, **6**, 8, 4, **7**, 3, 4]

A = [**1**, 7, **8**, 4, 6, 3, 4]

A = [8, 7, **1**, 4, 6, 3, **4**] - recursive

A = [8, 7, 4, 4, 6, 3, 1] - done

Heapsort

Move first to end, then re-heapify

A = [**8**, 7, 4, 4, 6, 3, **1**], move end

A = [<u>**1**</u>, <u>**7**</u>, <u>4</u>, 4, 6, 3, 8], heapify

A = [7, <u>**1**</u>, 4, <u>4</u>, <u>**6**</u>, 3, 8], rec. heap

A = [**7**, 6, 4, 4, 1, **3**, 8], move end

A = [<u>**3**</u>, <u>**6**</u>, <u>4</u>, 4, 1, 7, 8], heapify

A = [6, <u>**3**</u>, 4, <u>**4**</u>, <u>1</u>, 7, 8], rec. heap

A = [6, 4, 4, 3, 1, 7, 8], next slide..

Heapsort

A = [**6**, 4, 4, 3, **1**, 7, 8], move end

A = [**1**, 4, **4**, 3, 6, 7, 8], heapify

A = [**4**, 4, 1, **3**, 6, 7, 8], move end

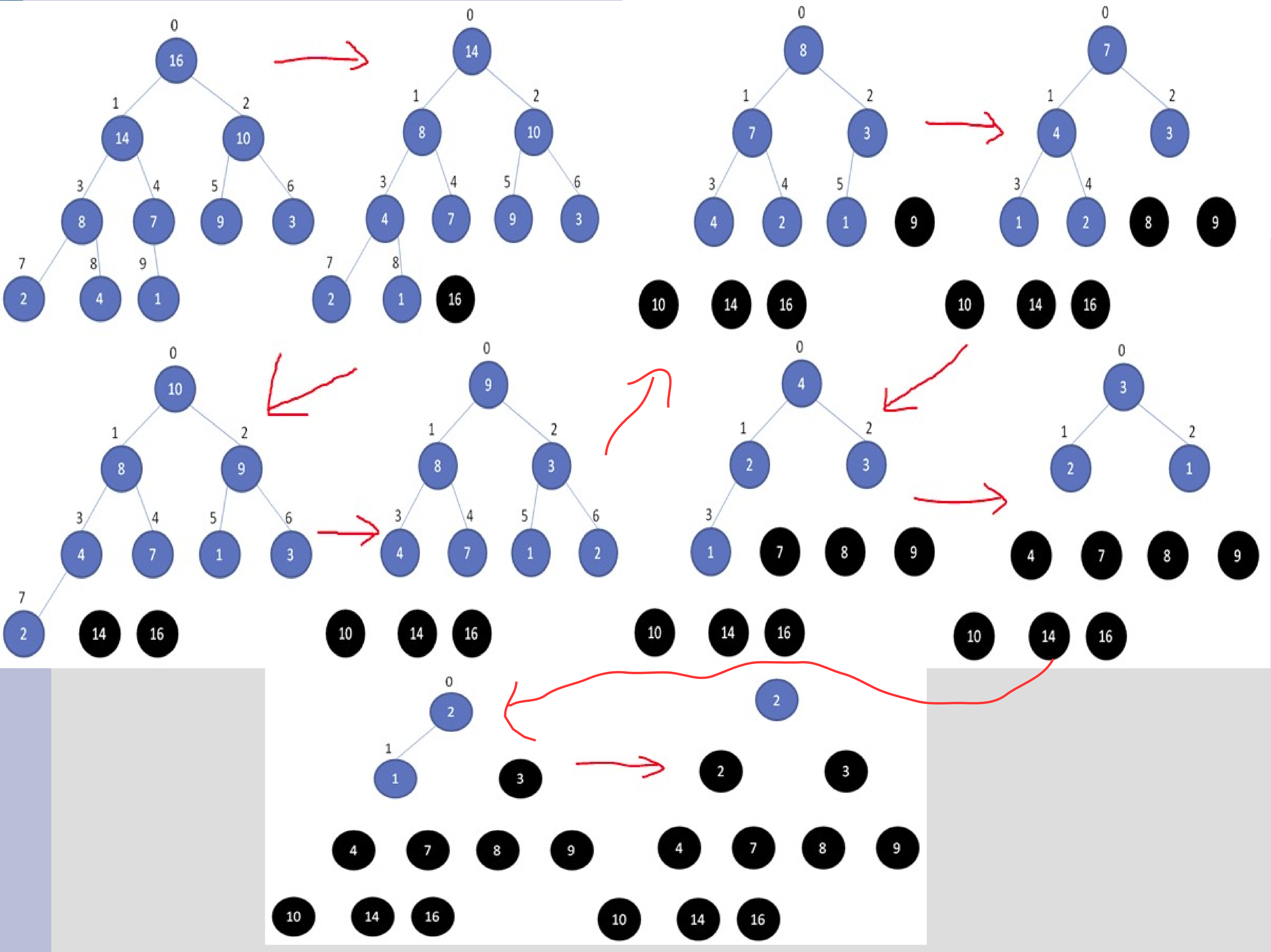
A = [**3**, **4**, 1, 4, 6, 7, 8], heapify

A = [**4**, 3, **1**, 4, 6, 7, 8], move end

A = [**1**, **3**, 4, 4, 6, 7, 8], heapify

A = [**3**, **1**, 4, 4, 6, 7, 8], move end

A = [1, 3, 4, 4, 6, 7, 8], done



Heapsort

Runtime?

Heapsort

Runtime?

Run Max-Heapify O(n) times
So... O(n lg n)

Priority queues

Heaps can also be used to implement priority queues (i.e. airplane boarding lines)

Operations supported are: Insert, Maximum, Extract-Max and Increase-key

Priority queues

Maximum(A):
  return A[1]

Extract-Max(A):
max = A[1]
A[1] = A.heapsize
A.heapsize = A.heapsize – 1
Max-Heapify(A, 1),   return max

Priority queues

```
Increase-key(A, i, key):
A[ i ] = key
while ( i>1 and A [floor(i/2)] < A[i])
   swap A[ i ], A [floor(i/2)]
   i = floor(i/2)
```

Opposite of Max-Heapify... move high keys up instead of low down

Priority queues

Insert(A, key):

A.heapsize = A.heapsize + 1

A [A.heapsize] = -∞

Increase-key(A, A.heapsize, key)

Priority queues

Runtime?

Maximum =

Extract-Max =

Increase-Key =

Insert =

Priority queues

Runtime?

Maximum = O(1)
Extract-Max = O(lg n)
Increase-Key = O(lg n)
Insert = O(lg n)

Sorting comparisons:

s=stable, p=parallelizable, i=in-place

Name	Average	Worst-case
Insertion[s,i]	$O(n^2)$	$O(n^2)$
Merge[s,p]	$O(n \lg n)$	$O(n \lg n)$
Heap[i]	$O(n \lg n)$	$O(n \lg n)$
Quick[p]	$O(n \lg n)$	$O(n^2)$
Counting[s]	$O(n + k)$	$O(n + k)$
Radix[s]	$O(d(n+k))$	$O(d(n+k))$
Bucket[s,p]	$O(n)$	$O(n^2)$

Sorting comparisons:

https://www.youtube.com/watch?v=kPRA0W1kECg