Heapsort
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?

$O(n \log n)$ 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 $O(h)$

$$
\sum_{h=0}^{\lfloor \ln n \rfloor} \left\lfloor \frac{n}{2^{h+1}} \cdot O(h) \right\rfloor
= O(n \cdot \sum_{h=0}^{\lfloor \ln n \rfloor} \left\lfloor \frac{h}{2^{h+1}} \right\rfloor)
\leq O(n \cdot \sum_{h=0}^{\infty} \frac{h}{2^h})
$$

Note: $\sum_{k=0}^{\infty} k \cdot c^k = c/(1 - c)^2$... 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] - recursive
A = [8, 7, 1, 4, 6, 3, 4] - done
Heapsort

Move first to end, then re-heapify
A = [8, 7, 4, 4, 6, 3, 1], move end
A = [1, 7, 4, 4, 6, 3, 8], heapify
A = [7, 1, 4, 4, 6, 3, 8], rec. heap
A = [7, 6, 4, 4, 1, 3, 8], move end
A = [3, 6, 4, 4, 1, 7, 8], heapify
A = [6, 3, 4, 4, 1, 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
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.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

<table>
<thead>
<tr>
<th>Name</th>
<th>Average</th>
<th>Worst-case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Insertion[i]</td>
<td>$O(n^2)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Merge[p]</td>
<td>$O(n \lg n)$</td>
<td>$O(n \lg n)$</td>
</tr>
<tr>
<td>Heap[i]</td>
<td>$O(n \lg n)$</td>
<td>$O(n \lg n)$</td>
</tr>
<tr>
<td>Quick[p]</td>
<td>$O(n \lg n)$</td>
<td>$O(n^2)$</td>
</tr>
<tr>
<td>Counting[s]</td>
<td>$O(n + k)$</td>
<td>$O(n + k)$</td>
</tr>
<tr>
<td>Radix[s]</td>
<td>$O(d(n+k))$</td>
<td>$O(d(n+k))$</td>
</tr>
<tr>
<td>Bucket[p]</td>
<td>$O(n)$</td>
<td>$O(n^2)$</td>
</tr>
</tbody>
</table>
Sorting comparisons:

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