

# LECTURE 5: COMPLEXITY, DATA-STRUCTURES AND SORTING

STAT 545: INTRO. TO COMPUTATIONAL STATISTICS

---

Vinayak Rao

Purdue University

September 2, 2019

Let  $x$  and  $y$  be  $N \times 1$  vectors

Let  $A$  and  $B$  be  $N \times N$  and  $N \times M$  matrices

How many additions and multiplications to calculate:

- $x^T y$
- $Ax$
- $AB$
- $A^{-1}$

The big-O notation provides an asymptotic upper bound:

$$O(g(N)) = \{f : \exists c, N_0 > 0 \text{ s.t. } f(N) \leq cg(N) \forall N > N_0\}$$

The big-O notation provides an asymptotic upper bound:

$$O(g(N)) = \{f : \exists c, N_0 > 0 \text{ s.t. } f(N) \leq cg(N) \forall N > N_0\}$$

$$2N^3 \in O(N^3)$$

$$N^2 \in O(N^3)$$

$$N^3 + N^2 \in O(N^3)$$

$$N^3 + \exp(N) \notin O(N^3)$$

The big-O notation provides an asymptotic upper bound:

$$O(g(N)) = \{f : \exists c, N_0 > 0 \text{ s.t. } f(N) \leq cg(N) \forall N > N_0\}$$

$$2N^3 \in O(N^3)$$

$$N^2 \in O(N^3)$$

$$N^3 + N^2 \in O(N^3)$$

$$N^3 + \exp(N) \notin O(N^3)$$

So is matrix multiplication  $O(N^3)$ ? Yes, but: it's also  $O(N^{2.38})$ !

The big-O notation provides an asymptotic upper bound:

$$O(g(N)) = \{f : \exists c, N_0 > 0 \text{ s.t. } f(N) \leq cg(N) \forall N > N_0\}$$

$$2N^3 \in O(N^3)$$

$$N^2 \in O(N^3)$$

$$N^3 + N^2 \in O(N^3)$$

$$N^3 + \exp(N) \notin O(N^3)$$

So is matrix multiplication  $O(N^3)$ ? Yes, but: it's also  $O(N^{2.38})$ !

Conjecture: matrix multiplication is actually  $O(N^2)$ .

Consider a set of  $N$  numbers. We want to sort them in decreasing order. What is the complexity?

Naïve algorithm:

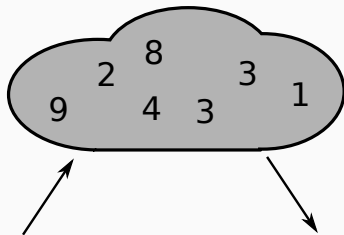
- Find smallest number. Cost?  $O(N)$
- Find next smallest number. Cost?  $O(N)$
- ...

Overall cost?  $O(N^2)$

Can we do better?

# PRIORITY QUEUE

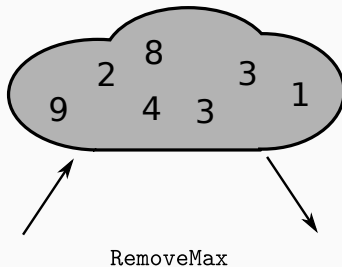
A 'bag' with three commands: Insert, FindMax and RemoveMax.





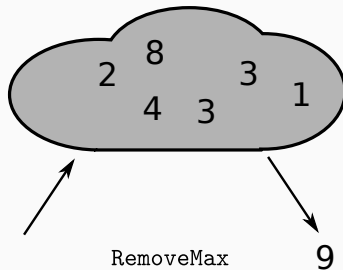
# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.



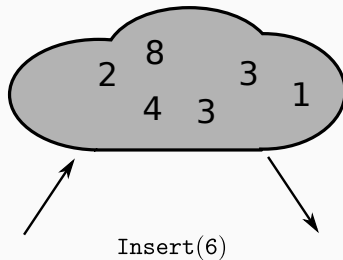
# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.



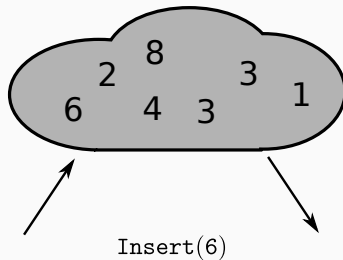
# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.



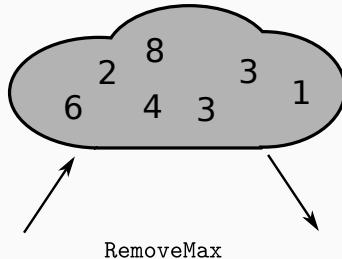
# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.



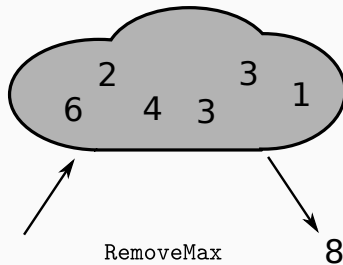
# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.



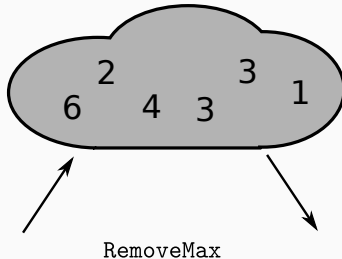
# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.



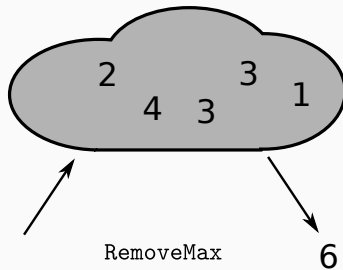
# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.



# PRIORITY QUEUE

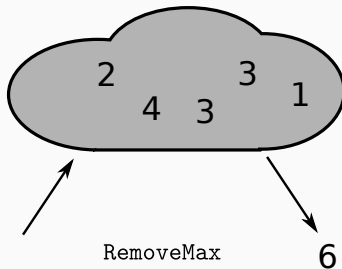
A 'bag' with three commands: Insert, FindMax and RemoveMax.





# PRIORITY QUEUE

A 'bag' with three commands: Insert, FindMax and RemoveMax.

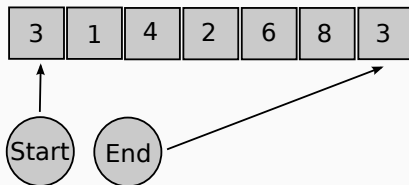


Sorting, clustering, discrete-event simulation, queuing systems  
How do we implement this?

# PRIORITY QUEUE

Naive approach 1: an unsorted array:

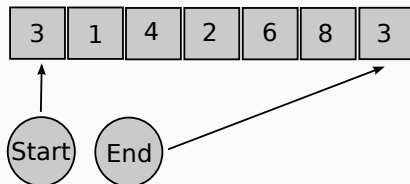
(3, 1, 4, 2, 6, 8, 3)



# PRIORITY QUEUE

Naive approach 1: an unsorted array:

(3, 1, 4, 2, 6, 8, 3)



What is the cost of **Insert**?  $O(1)$

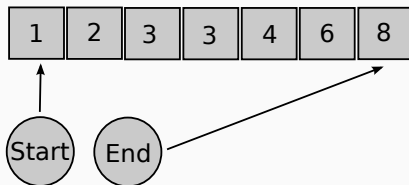
What is the cost of **FindMax**?  $O(N)$

What is the cost of **RemoveMax** (assume we've already found the maximum)?  $O(N)$

# PRIORITY QUEUE

Naive approach 2: a sorted array:

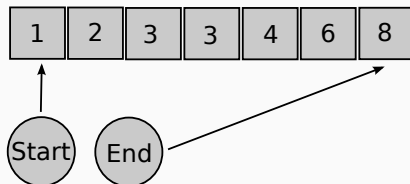
(1, 2, 3, 3, 4, 6, 8)



# PRIORITY QUEUE

Naive approach 2: a sorted array:

(1, 2, 3, 3, 4, 6, 8)



Cost of FindMax?  $O(1)$

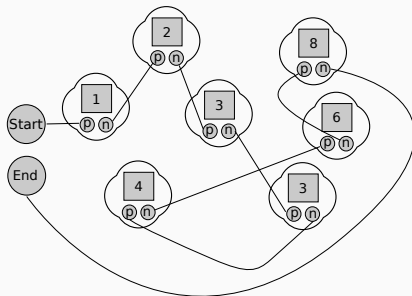
Cost of RemoveMax?  $O(1)$

Cost of Insert.FindPosition?  $O(\log(N))$  (binary search)

Cost of Insert.Insert?  $O(N)$

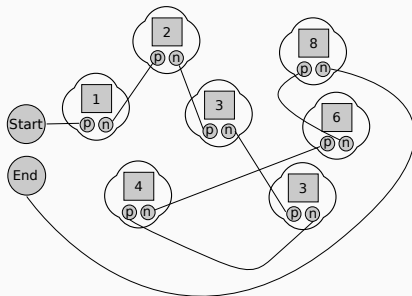
# PRIORITY QUEUE

Naive approach 3: a sorted doubly linked-list:



# PRIORITY QUEUE

Naive approach 3: a sorted doubly linked-list:



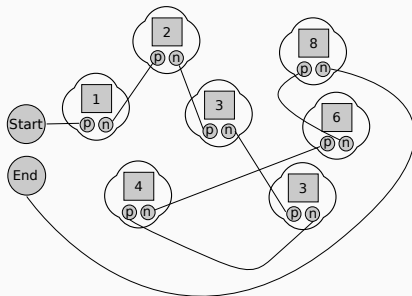
What is the cost of FindMax?  $O(1)$

What is the cost of RemoveMax?  $O(1)$

What is the cost of Insert?  $O(N)$

# PRIORITY QUEUE

Naive approach 3: a sorted doubly linked-list:



What is the cost of **FindMax**?  $O(1)$

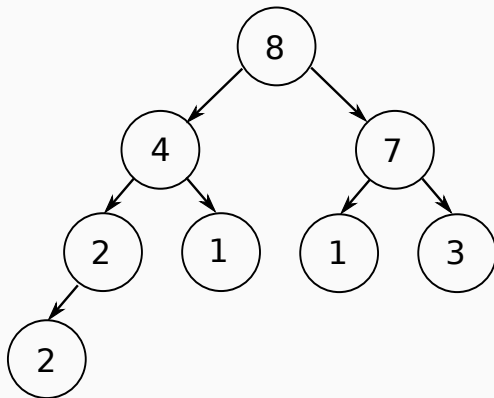
What is the cost of **RemoveMax**?  $O(1)$

What is the cost of **Insert**?  $O(N)$

Each approach solves one problem, but makes another operation  $N$ . Can we do better?



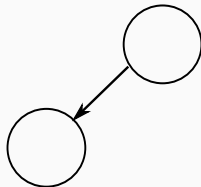
We need a more complicated data-structure: a Heap.



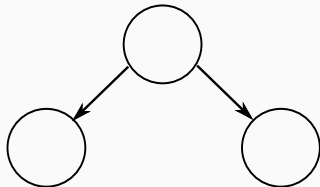
For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>



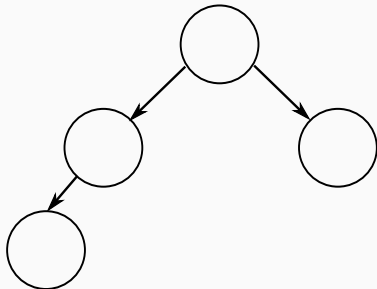
For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>



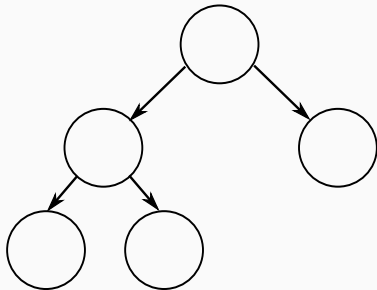
For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>



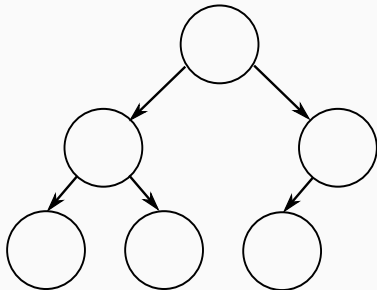
For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>



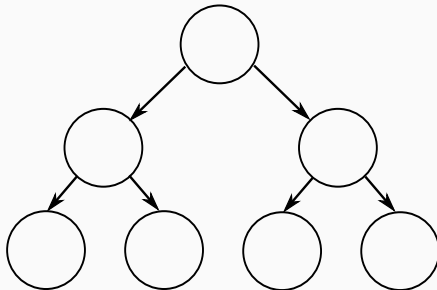
For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>



For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>

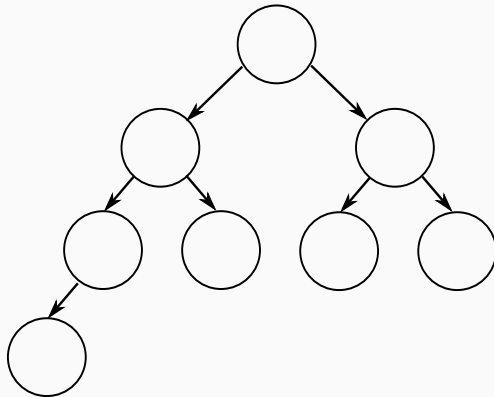


For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>

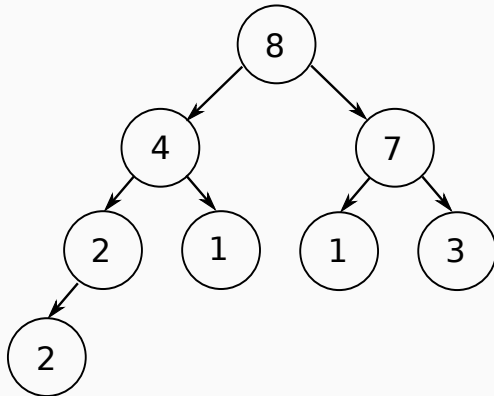


For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>

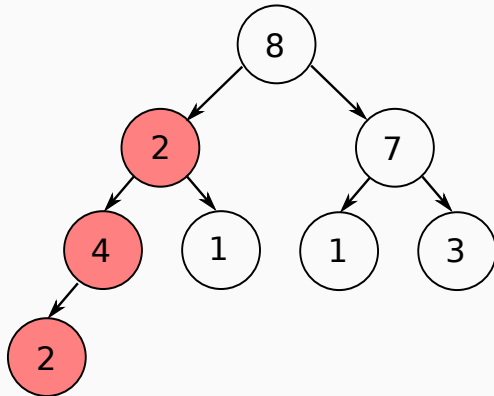




For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>

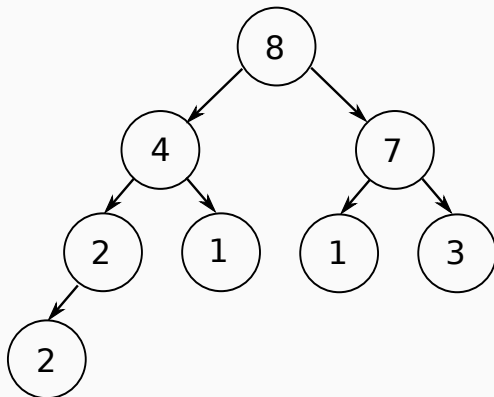


For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>

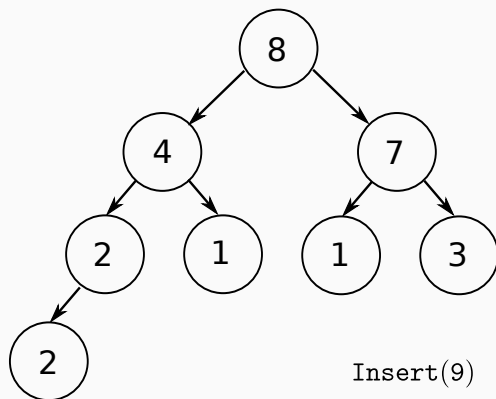


For a precise definition, see: <http://pages.cs.wisc.edu/~vernon/cs367/notes/11.PRIORITY-Q.html>

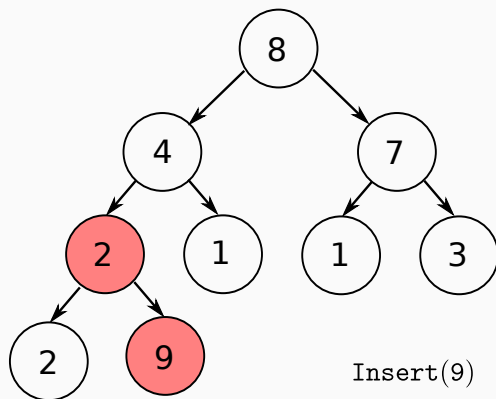
## HEAPS: Insert



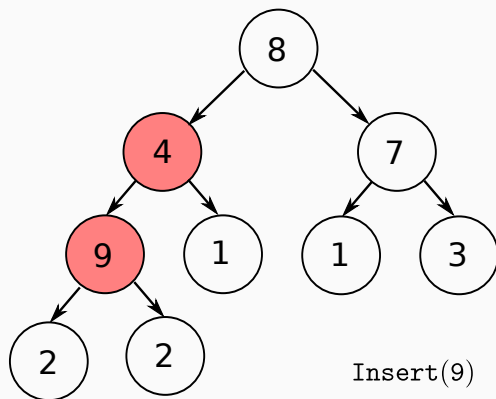
## HEAPS: Insert



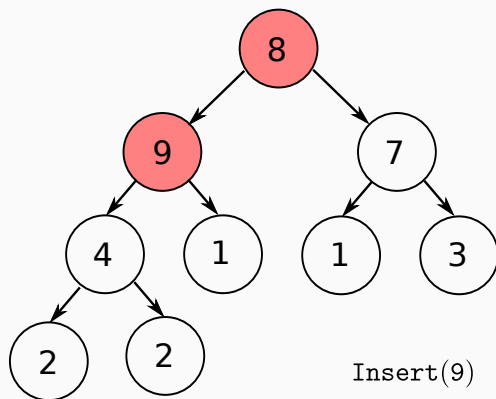
## HEAPS: Insert



## HEAPS: Insert

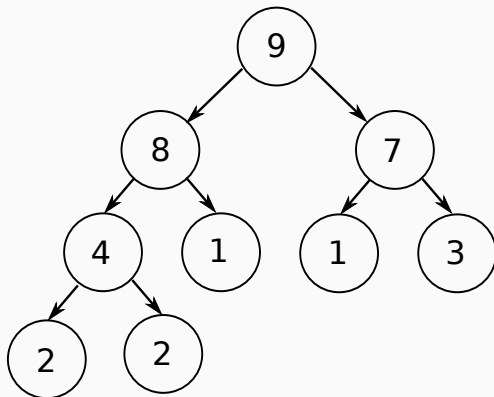


## HEAPS: Insert

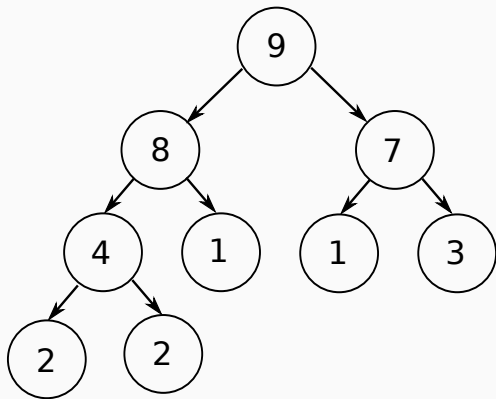




## HEAPS: Insert



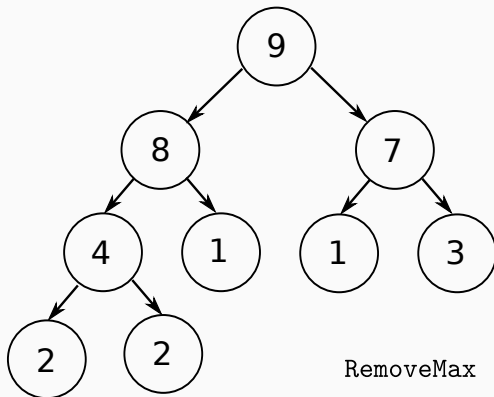
## HEAPS: Insert



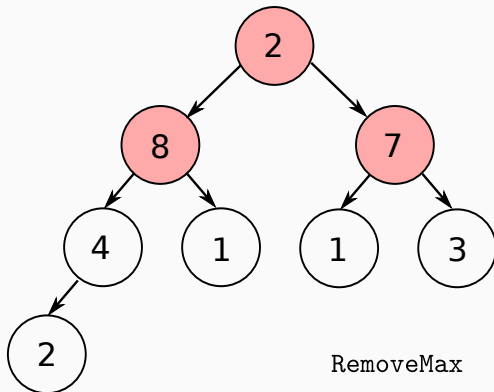
Cost?

$O(\log(N))$

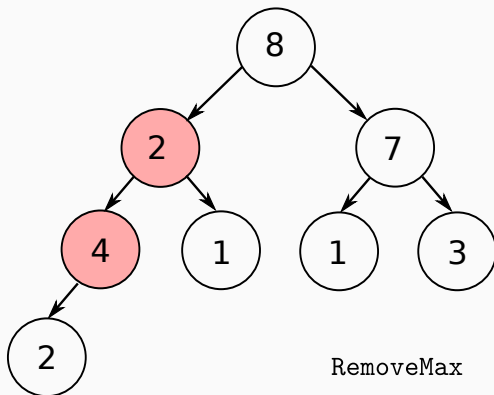
## HEAPS: RemoveMax



## HEAPS: RemoveMax

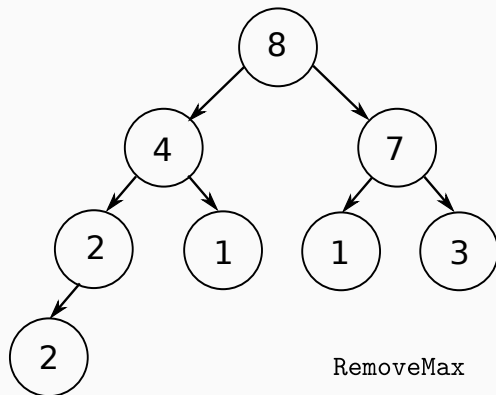


## HEAPS: RemoveMax

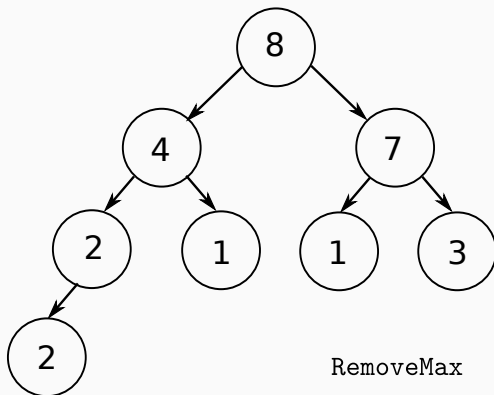


Swap with larger child

## HEAPS: RemoveMax



# HEAPS: RemoveMax



Cost?

$O(\log(N))$

# HEAPSORT

Consider a set of  $N$  numbers. Want to sort in decreasing order.

Grow a priority queue, sequentially adding elements

- Cost of each step?  $O(\log(N))$
- Overall cost?  $O(N \log(N))$

Sequentially remove the maximum element

- Cost of each step?  $O(\log(N))$
- Overall cost?  $O(N \log(N))$

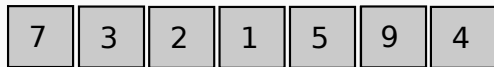
Cost of overall algorithm?  $O(N \log(N))$



# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

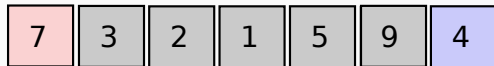
- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half



# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half

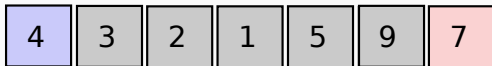


- Pivot: #7 (Blue)
- Start: #1 (Red)
- End: #6

# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half

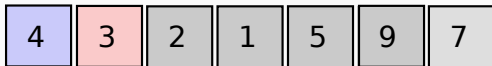


- Pivot: #1
- Start: #2
- End: #6

# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half

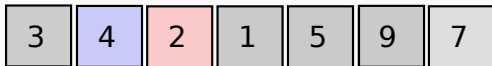


- Pivot: #1
- Start: #2
- End: #6

# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half

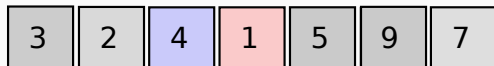


- Pivot: #2
- Start: #3
- End: #6

# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half



- Pivot: #3
- Start: #4
- End: #6

# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half



Recurse for each half

# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half



Recurse for each half



# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half



Recurse for each half

# QUICKSORT

See <http://me.dt.in.th/page/Quicksort/> for a nicer display

- Choose a pivot
- Ensure all elements to left of pivot are less than/equal, and all to right are greater than the pivot
- Recurse for each half



At the end, we have a sorted list

Analysis is a bit harder

What is the worst-case runtime?

What is the best-case runtime?

Average run-time is  $\Theta(n \log n)$

Average with respect to what?

*Randomized* algorithms

## FINAL (INFORMAL) COMMENTS

Class **P**: Problems of polynomial complexity. Let  $T(n)$  be running-time for input size  $n$ . Then there is a  $k$  such that:

$$T(n) = O(n^k)$$

Class **E**: Problems of exponential complexity.

$$T(n) = \exp(O(n))$$

Class **NP**: Problems of where proposed solution can be verified in polynomial time. E.g. graph isomorphism

Class **NP-hard**: at least as hard as the hardest problems in **NP** (halting problem)

Class **NP-complete**: Hardest problems in **NP** (i.e. problems in both **NP** and **NP-hard**). E.g. travelling salesman.

# $P = NP?$

A million dollar question (literally)

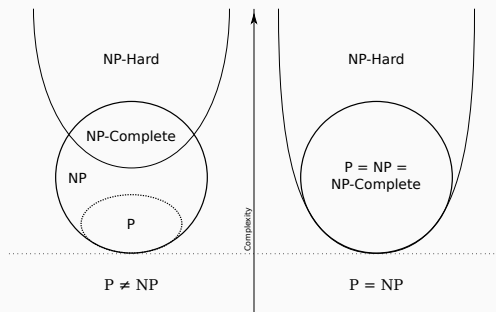


Figure: from Wikipedia