# COMP20003 Workshop Week 9

| | |
|---|---|
| **1** | Priority Queue |
| **2** | Heaps & Binary Heaps |
| **3** | Heap Sort |
| | |
| **L** | Q 8.1 |
| **A** | Programming 8.1: Natural Merge Sort |
| **B** | |

# ADT: Queue & Priority Queue

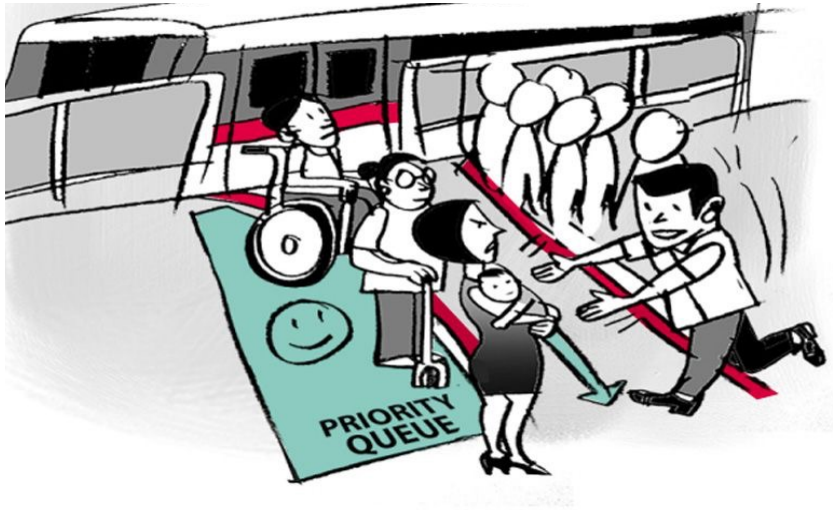

Remember queue?

enqueue, dequeue?

What is a PQ?

Is it similar to a queue (LIFO)?

What's priority? highest priority?




*'Can I borrow your baby?...'*

# Real-life Examples

- a list of to-do works
- a hospital queue where the patient with the most critical situation would be the first in the queue (each patient would be issued a number representing the critical level)

# Yet Another ADT: Priority Queue

PQ: queue, where each element is associated with a *priority* (or *weight*), and the elements will be *dequeued* following the order of priority.


*'Can I borrow your baby?...'*

Main operations:

- *create*: creates an empty `PQ: makePQ()`
- ***enqueue***: inserts (element, weight) into `PQ: enPQ(PQ, item)`
- ***dequeue***: removes & returns the heaviest element of `PQ:`
  `dePQ(PQ)`, or **`deleteMax(PQ)`**, or **`deleteMin(PQ)`**
- check for being empty**: `isEmptyPQ(PQ)`**
- *changeWeight*: change the weight of a particular element of a queue
- *peek/frontier*: returns the heaviest element without removing

# unreasonable, but possible, concrete data structures for PQ

| DS | complexity of construction a PQ of n elements | complexity of dePQ | complexity of peek |
|---|---|---|---|
| unsorted arrays or linked list | | | |
| sorted arrays or linked lists | | | |

Example: priority= max

Unsorted: 9 2 7 5 6 8 3

Sorted : 2 3 5 6 7 8 9

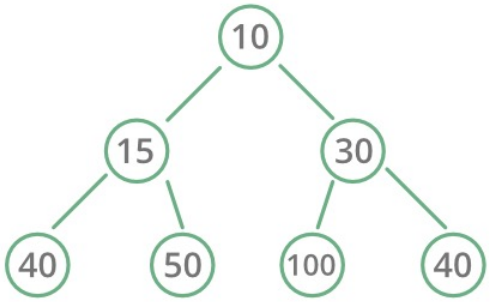
# unreasonable, but possible, concrete data structures for PQ

| DS | complexity of construction a PQ of n elements | complexity of dePQ | complexity of peek |
|---|---|---|---|
| unsorted arrays or linked list | O(n) | O(n) | O(n) |
| sorted arrays or linked lists | $O(n^2)$ or O(nlogn) | O(1) | O(1) |

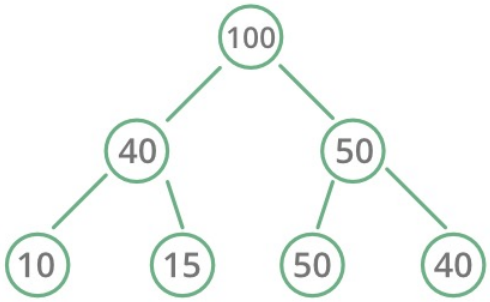
# Binary Heap

Binary heap is a priority queue

For simple keys, we can have min-heap or max-heap

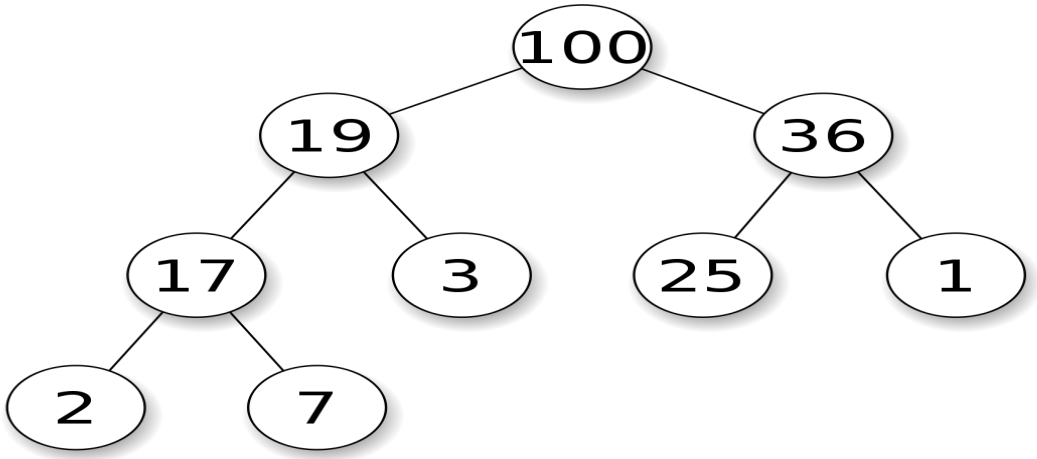



Min Heap



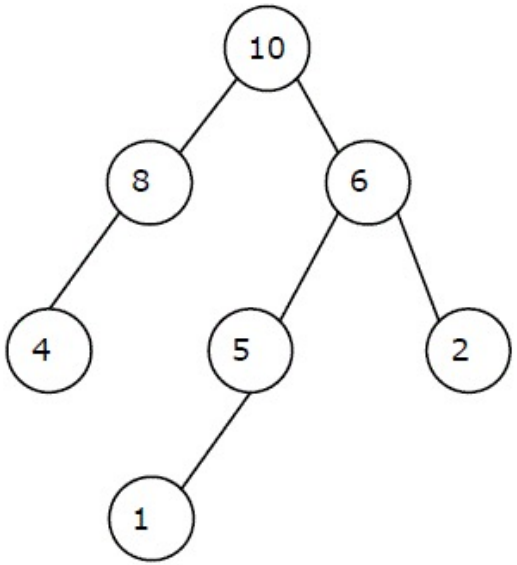

Max Heap

# Example of PQ: Binary Max/Min Heap

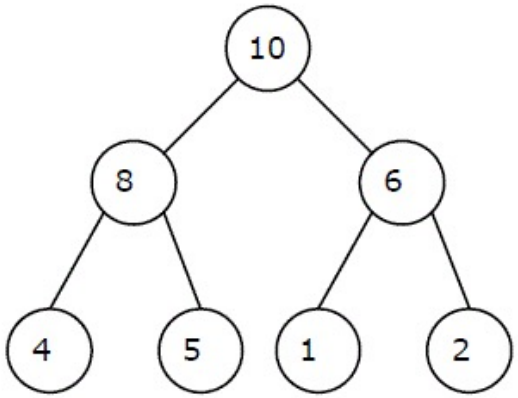
| Example | Conditions |
| --- | --- |
|   | 1. The tree is complete.<br>2. *The heap property*: each node has a higher priority (here, is larger) than any of its descendants (or equivalently, just its children). |

# which one is a binary heap? send me a letter

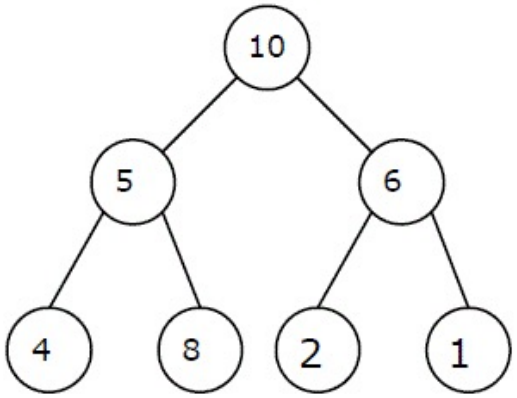
(A)



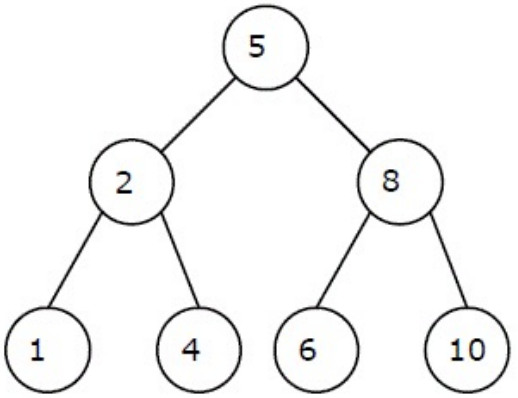

(B)




(C)

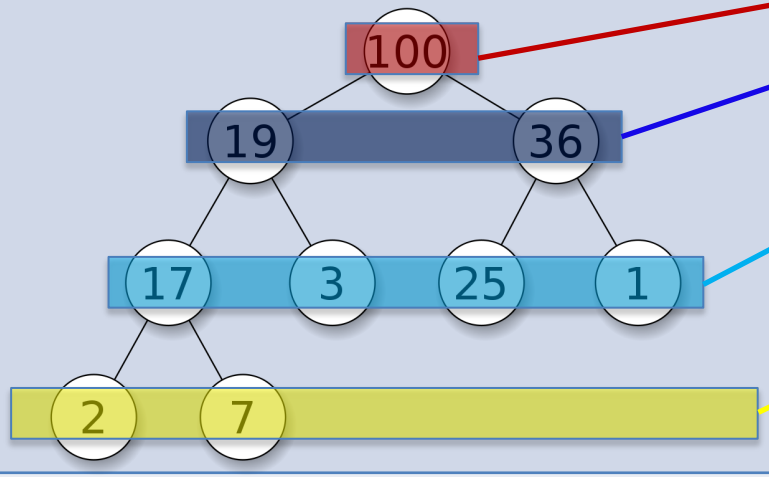



(D)



# Binary Heap is implemented as an array!

**Visualisation: as a complete binary tree**




**Implementation: using arrays**

| idx | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| val | × | 100 | 19 | 36 | 17 | 3 | 25 | 1 | 2 | 7 |

*note:* `H[1]` is for the root
`H[0]` not used

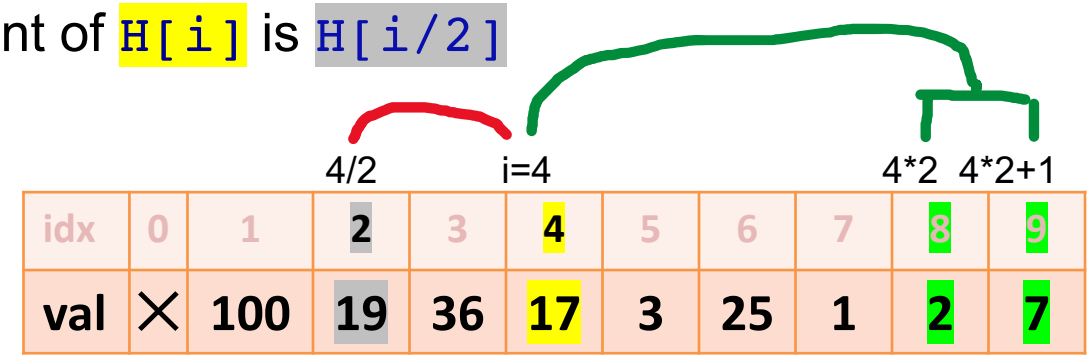
Heap is `H[1..n]`

- level `i` occupies $2^i$ cells in array `H[1..n]` from idx $2^i$ to $2^{i+1}-1$

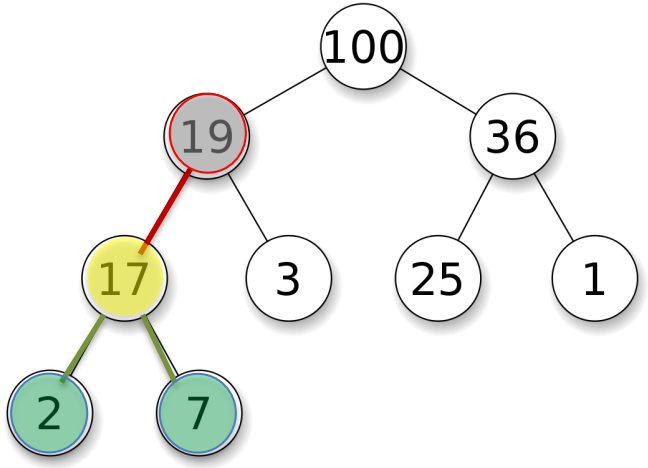
# Binary Heap is implemented as an array!

- left child of `H[i]` is `H[2*i]`
- right child of `H[i]` is `H[2*i+1]`

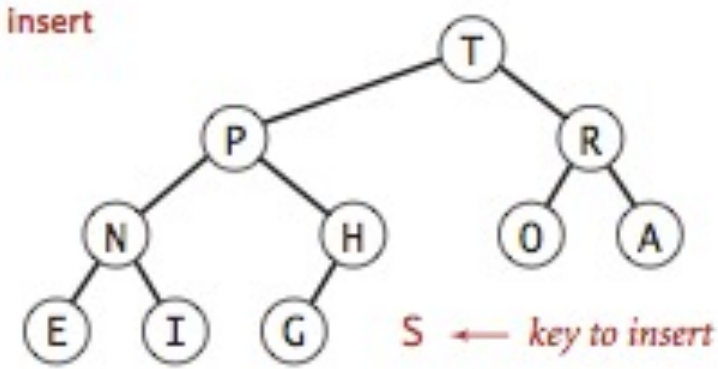
parent of `H[i]` is `H[i/2]`

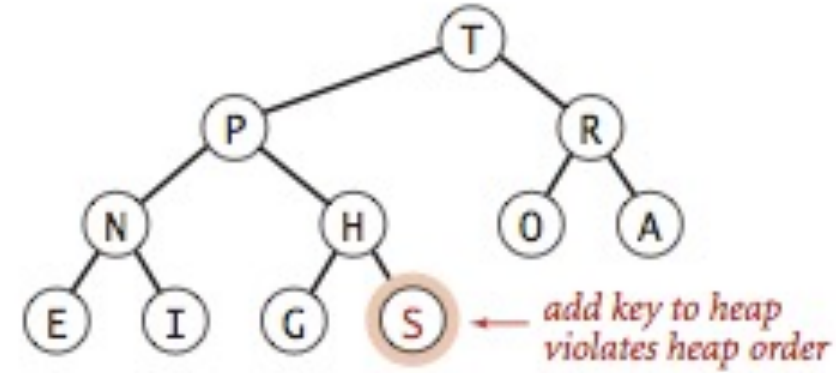


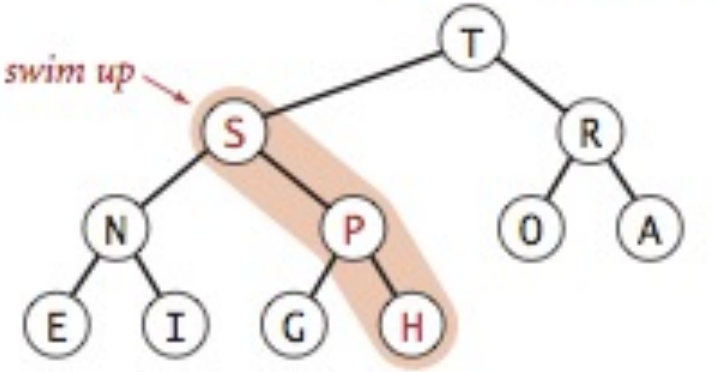

| idx | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|---|
| val | × | 100 | 19 | 36 | 17 | 3 | 25 | 1 | 2 | 7 |



# Insert a new elem into a heap. Complexity= ?




1 2 3 4 5 6 7 8 9 10 11

H= [T,S,R,N,P,O,A,E,I,G,H]

H has 10 elements

Insert S

Just added H[11]= S

5=11/2 11

[T,P,R,N,H,O,A,E,I,C,S]

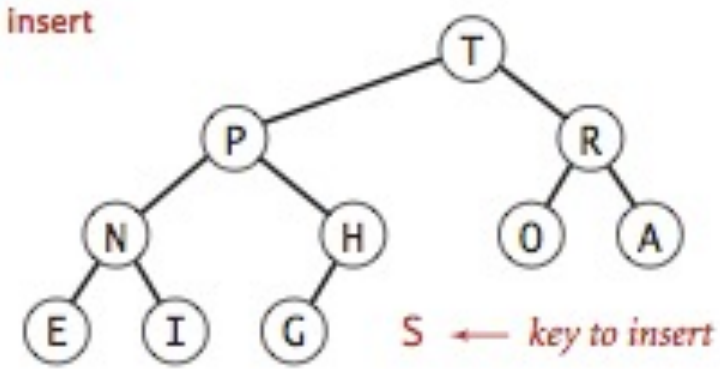
that will likely violate heap order

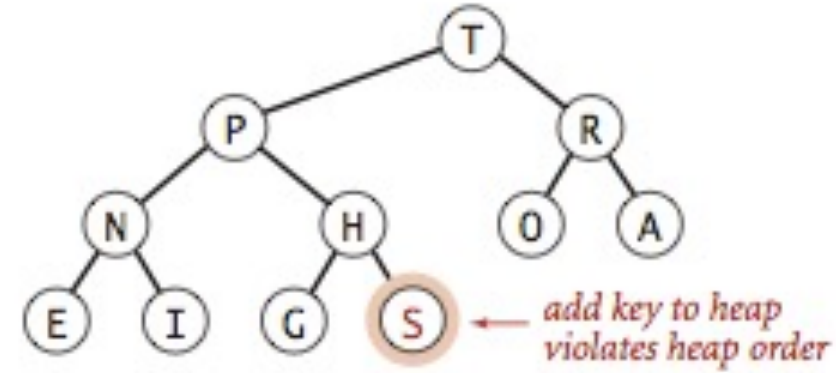
Need to promote H[11] up using upheap(h,i=11), which repeatedly swap node i with its parent.

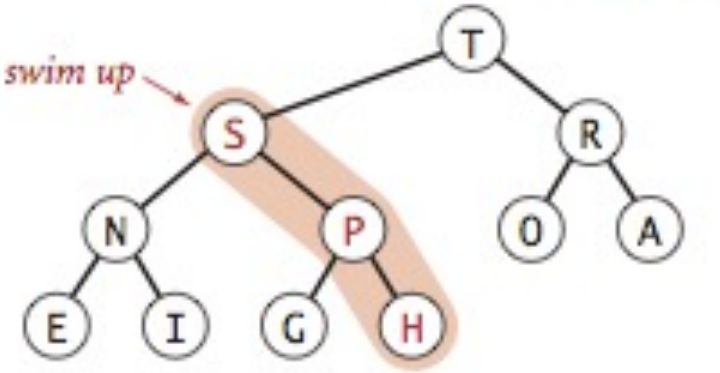
2=5/2 5=11/2 11

[T,P,R,N,H,O,A,E,I,C,S]

# Insert a new elem into a heap (enPQ). Complexity= ?




Complexity= O(log n) not Theta

Need to promote `H[11]` up using `upheap(h,i=11)`, which repeatedly swap node `i` with its parent.

2/2=1 2=5/2 5=11/2 11

`[T,P,R,N,H,O,A,E,I,C,S]`

S H

S P

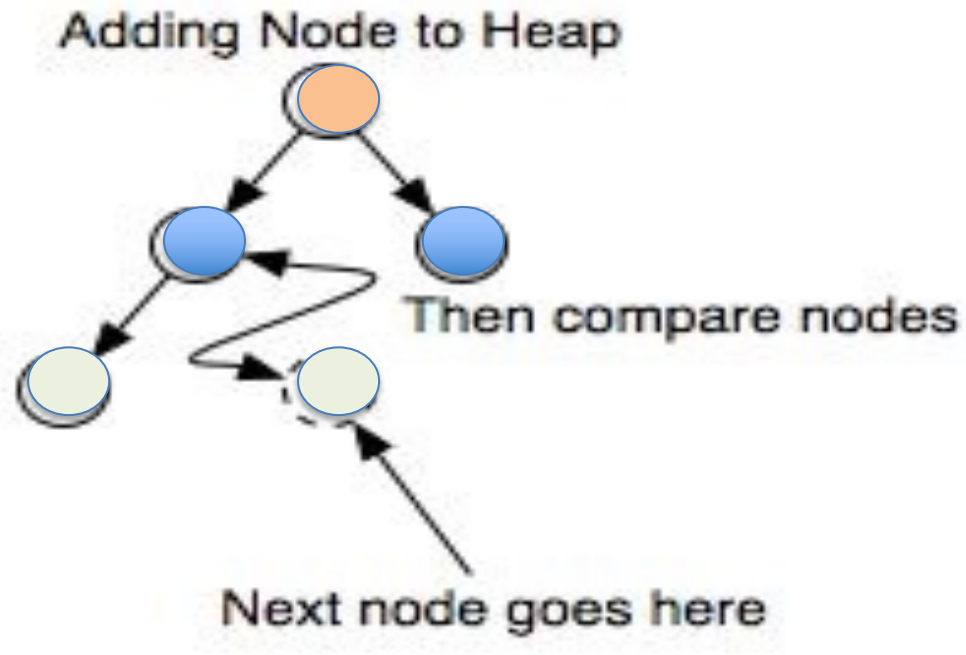
S

# upheap – basic operation for inserting into heaps

Problem: The last node of (e.g. just inserted into) a heap, and only it, might violate the heap property. Need to repair the heap.

index= 1 2 3 4 5

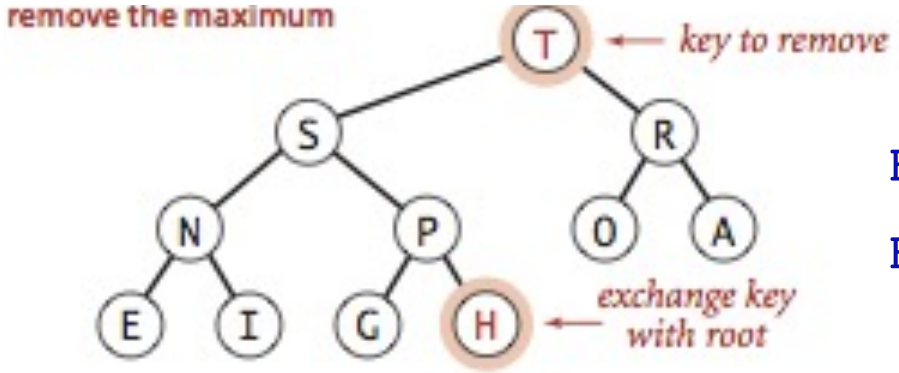
Operation: `upheap(h,node)`

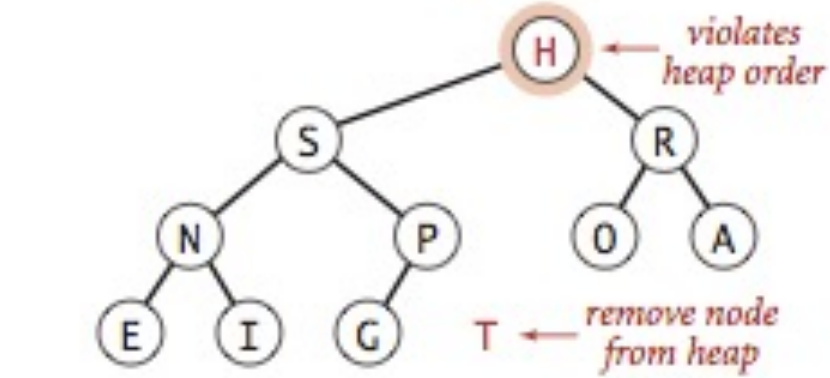



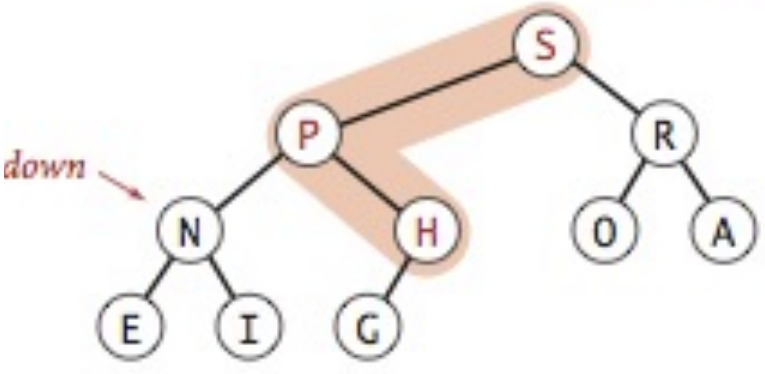

```
loop while(not_heap) {
   swap(node, parent)
}
```

Complexity= ?

# deletemax: delete (and returns) the heaviest. Complexity=




```
  1  2  3  4  5  6  7  8  9  10 11
H= [S,P,R,N,H,O,A,E,I,G],T]
```

H has 11 elements

exchange and remove the last -->

```
[H,S,R,N,P,O,A,E,I,G]
```

n=11, heap order violated

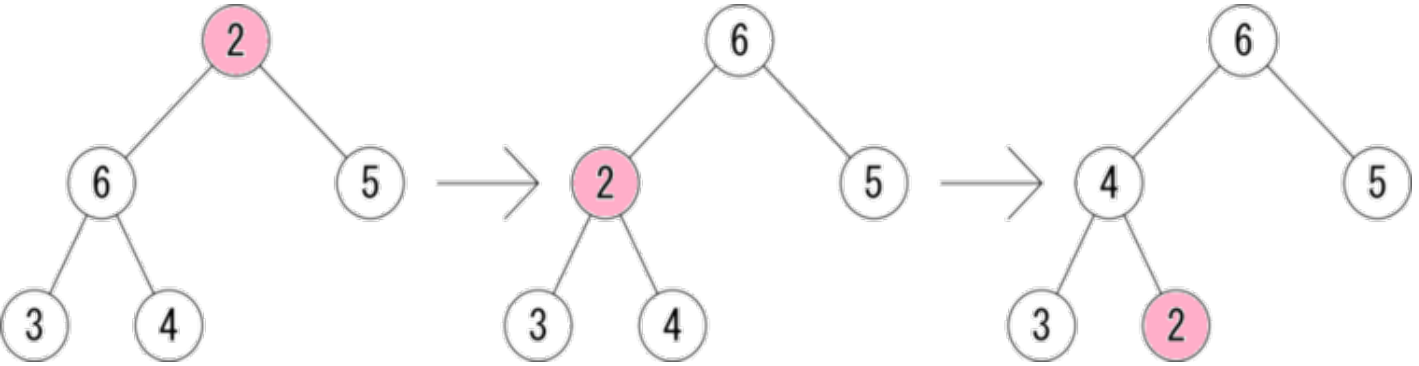
Need to push `H[1]` down using `downheap(h,i=1),` which repeatedly swap node `i` with its heaviest child.

```
[H,S,R,N,P,O,A,E,I,G] cs= 2,3
[S,H,R,N,P,O,A,E,I,G] cs= 4,5
[S,P,R,N,H,O,A,E,I,G] cs= 10
```

# downheap – basic operation deletion from heap

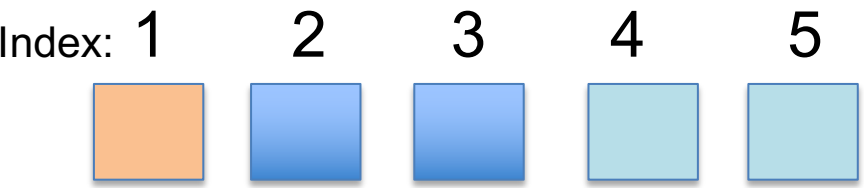
Problem after replacing `H[1]`: The root (and only the root) might violate the heap property. Need to repair the heap.

Operation: `downheap(h,node)` `Complexity=`




maxchild(i)= ?

```
loop while (not_heap) {
    swap (node, maxchild)
}
```



# Notes& Complexity

Notes: `upheap(H, node)` and `downheap` can be performed for any node of the heap. Example: changing the priority of a node in heap.

# Heapsort=

How?

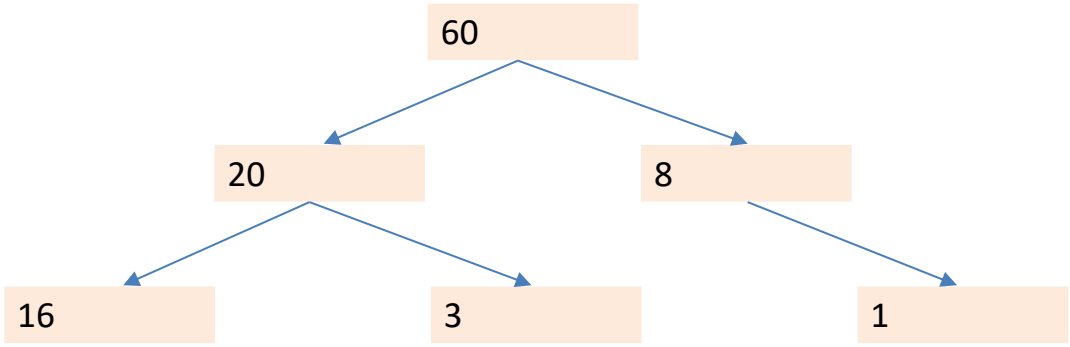
# Heapsort example (Grady's slides)

Sort the keys: 20,3,60, 8,1,16

First make them into  heap.

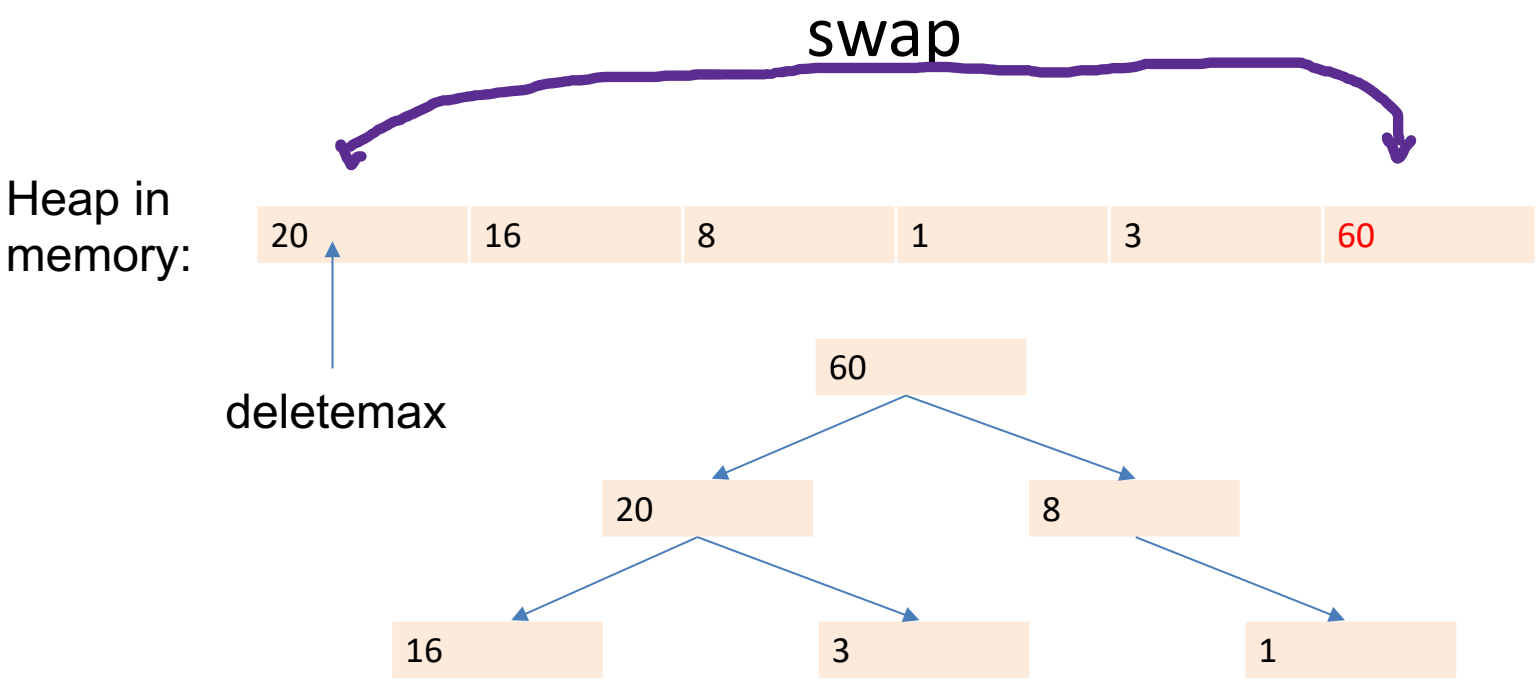
Heap in memory:

| 60 | 20 | 8 | 16 | 3 | 1 |
|---|---|---|---|---|---|

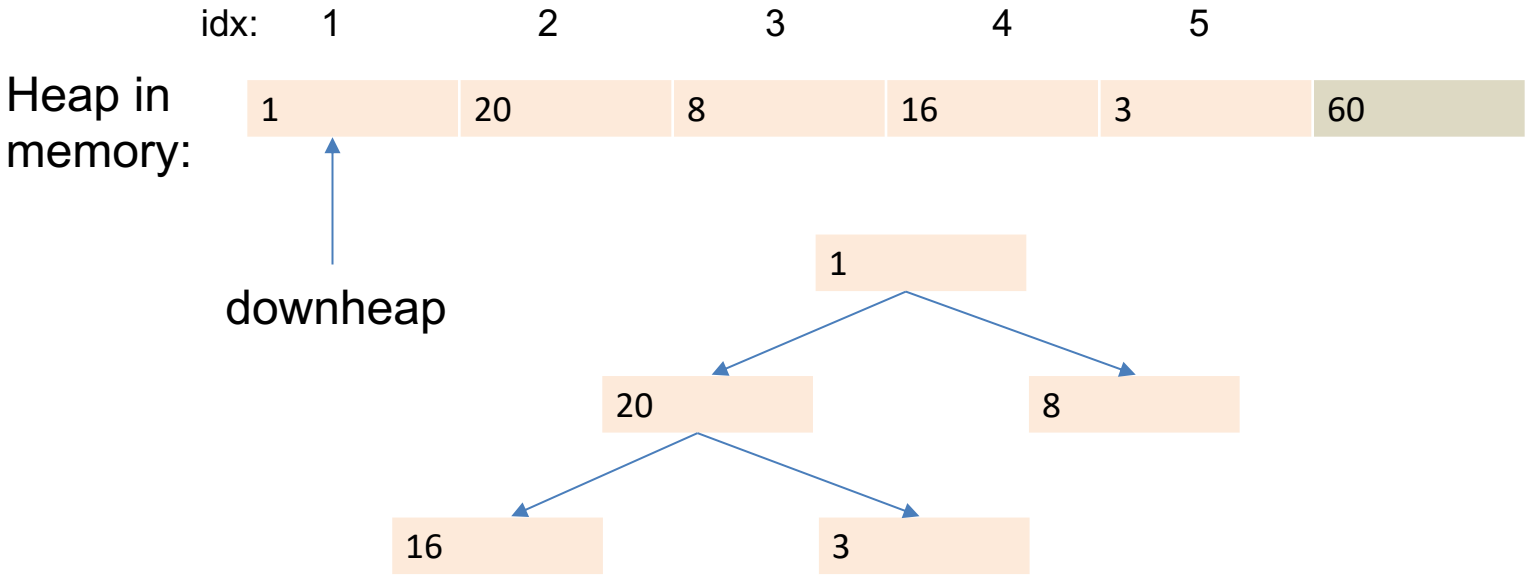


# Heapsort

Then, repeatedly deletemax by a) swapping it with the end, then b) downheap

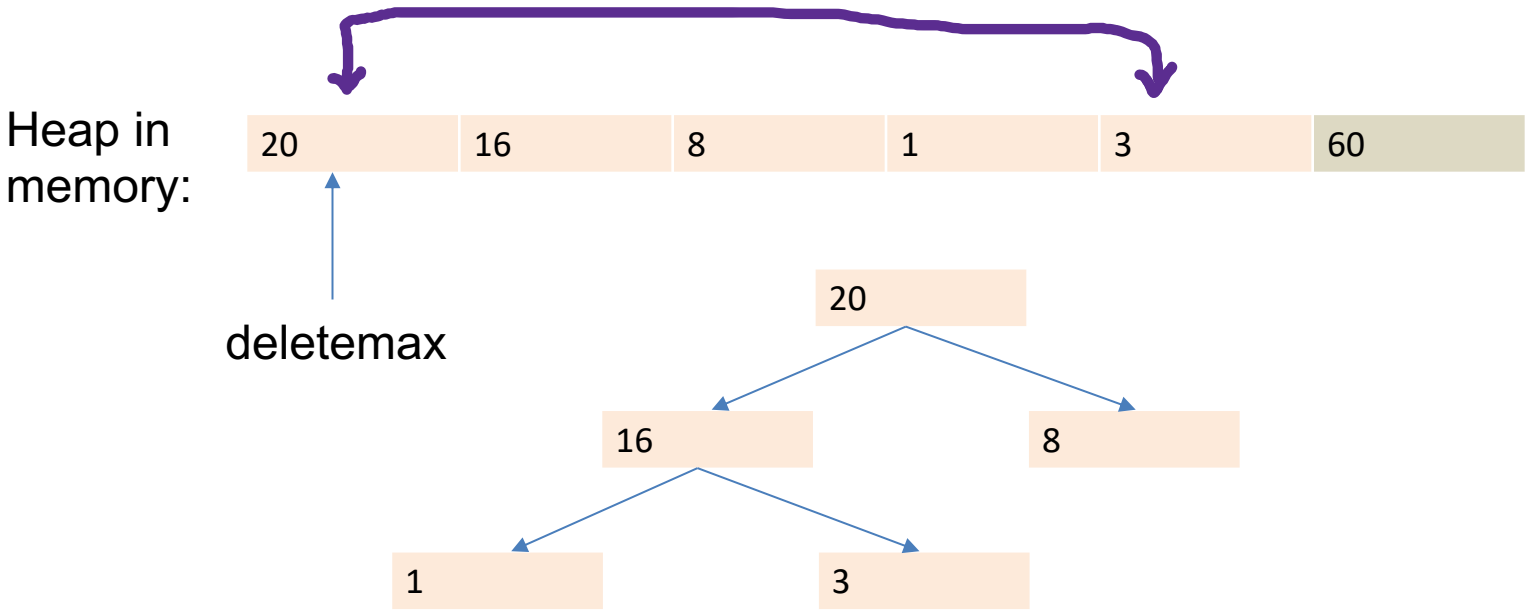


# Heapsort

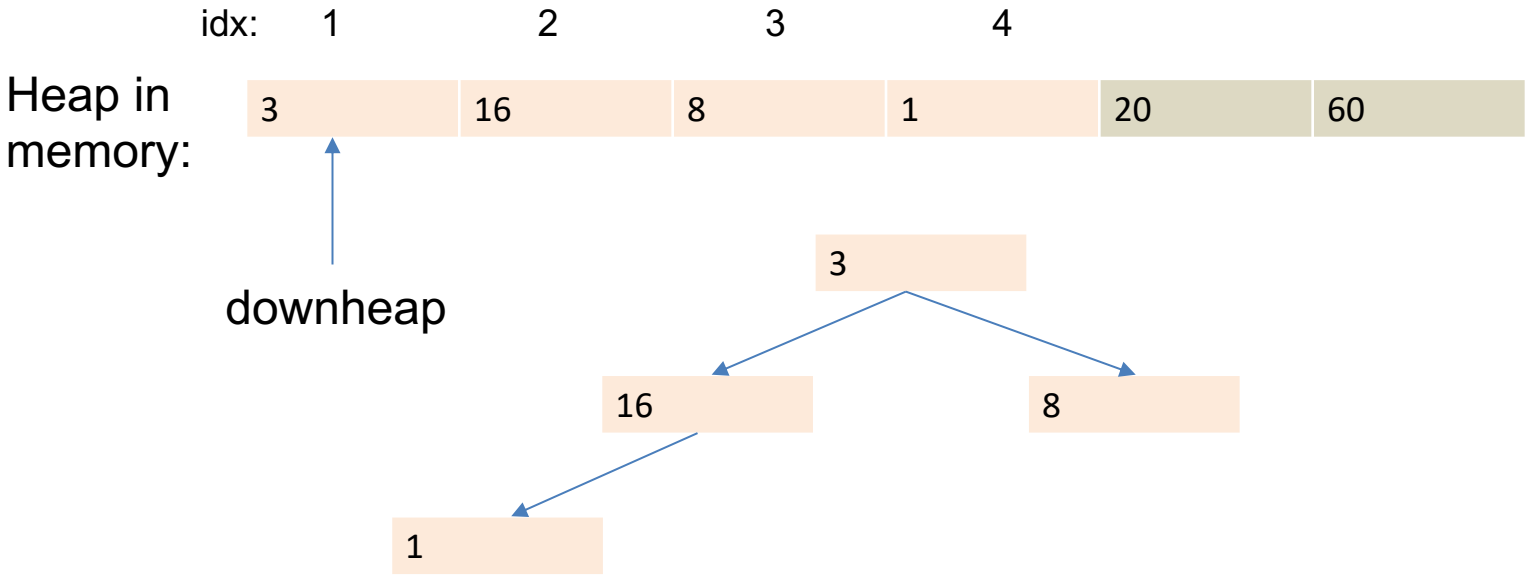


# Heapsort

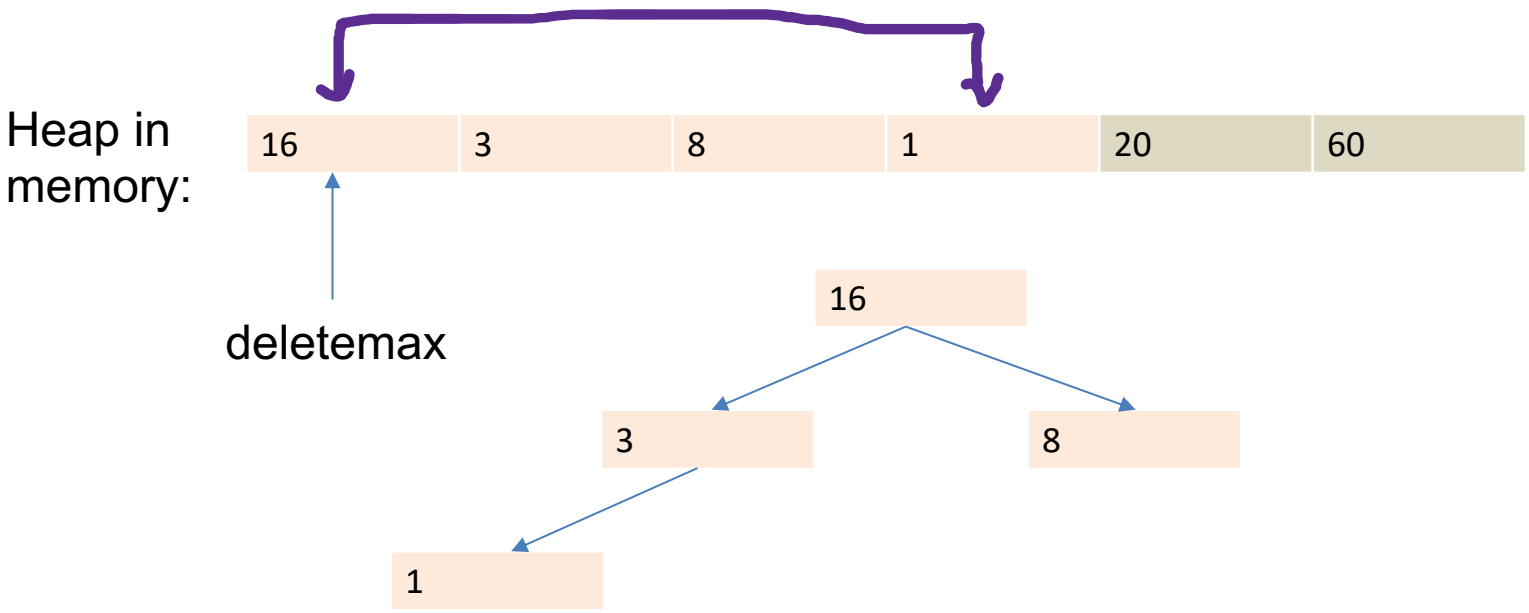
Repeat the process until heap containing a single item



# Heapsort



# Heapsort



# Heapsort

… done when heap just has 1 element

idx: 1

Heap in memory:


1

# HeapSort summary

- Construct a max heap of n elements.
- Swap root (max) with the (current) last element.
- Remove last element from further consideration, *i.e.* decrease size of heap by 1.
- Fix heap using....
  ... `downheap(for node i=1)`
- Repeat until finished (ie heap just has 1 element).
- Complexity= max( complexity of constructing a heap, n logn)
- = O(n+nlogn) = O(nlogn)

# How to efficiently build a heap with n elements?

- Solution 1: insert each element into the (initially empty) heap, and do `upheap` after each insertion.

  Complexity: O(n logn)

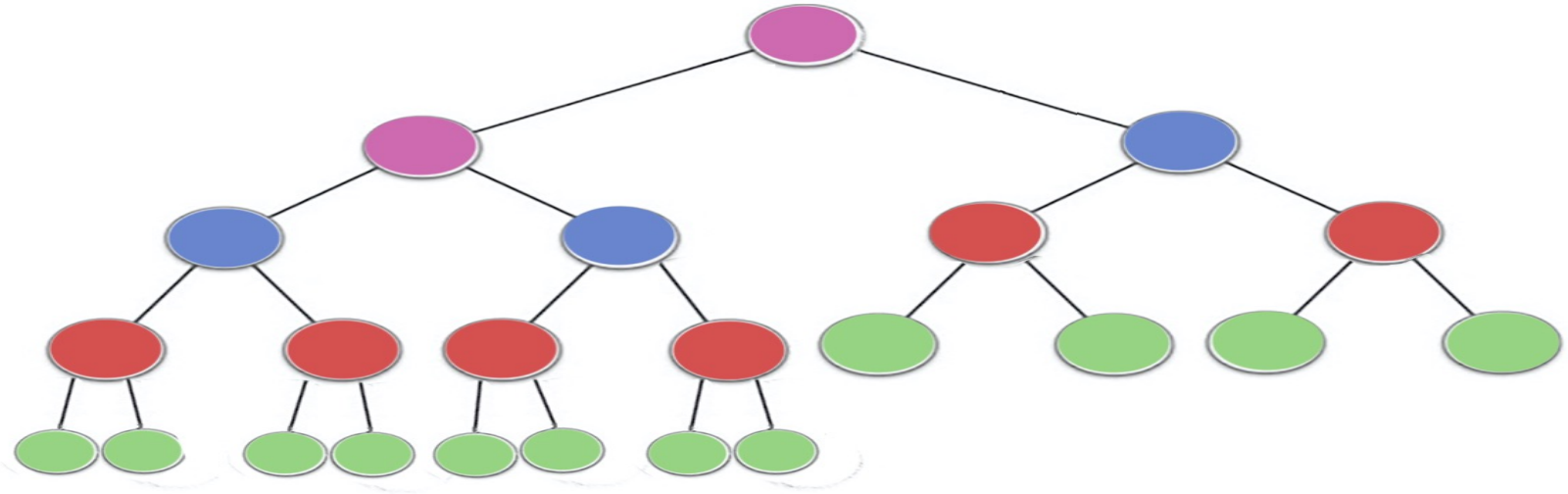
# How to efficiently build a heap with n elements? heapify

**Solution 2:** populate the heap array with n elements in the input order, then turn the array to a heap (ie make it to satisfy the heap condition). Algorithm:

```
for (i=n/2; i>0; i--) {
// for i from last parent to fisrt parent
    downheap(h, i);
   time= n/4 + n/8 * 2  + n/16*3 + = sum ( n*i/ 2^i)=
n x sum( i/2^i)
```

= **Θ(n)** (see lectures and/or ask Google for a proof)

The operation is aka. **Heapify**/Makeheap/ Bottom-Up Heap Construction



# Heap & Heap Sort: Complexity

Heap operations:

- `upheap`:
- `downheap`:
- insert 1 element:
- `deleteMax`:
- heapify: (turning array into a heap)= Theta(n)
- `heapsort`:

# Q 8.1 in break-out room

Construct a max binary heap from the following keys:

8 7 16 10 17

a) Construct a max binary heap using the up-heap, inserting one number at a time.

b) Now construct a max binary heap from the same keys, using downheap (ie convert the original array into a heap).

c) What is the complexity of each method?

# a) by inserting one-by-one

Construct a max binary heap from the following keys:

8 7 16 10 17

# b) by heapify

Construct a max binary heap from the following keys:

8 7 16 10 17

# MST Questions

If you have any questions from the MST you'd like to go over, let us know

# Demonstration – Adaptive Merge Sort (Grady's slides)

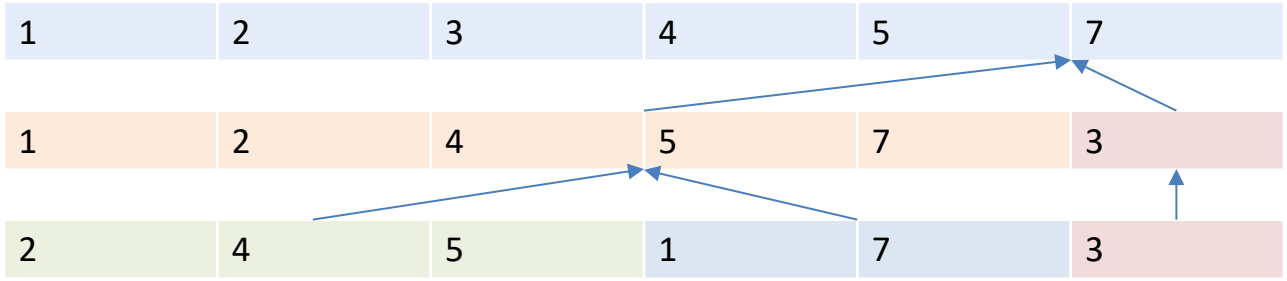
Bottom-up merge sort improvement

- Monotonic increasing runs already sorted
- Insert monotonic runs into queue instead of singletons

# Demonstration – Adaptive Merge Sort

Bottom-up merge sort improvement

- Best Case: Θ(n)
- Worst Case: Θ(n log n)



# Lab: in break-out rooms

- Q 8.1: For the following keys.

  **8 7 16 10 5 13 5 11 15 12 1 17**

  - Construct a max binary heap using the top-down approach (adding items to heap one-at-a-time).
  - Now construct a max binary heap from the same keys, using the bottom-up "heapify" method.
  - What is the complexity of each method? Did the time it took you to do the exercise on paper correlate (roughly) with the theoretical complexity?

- Programming 8.1: Work in breakout rooms through writing adaptive merge sort program