COMP90038
Algorithms and Complexity

Lecture 13: Priority Queues, Heaps and Heapsort
(with thanks to Harald Søndergaard & Michael Kirley)

Andres Munoz-Acosta
munoz.m@unimelb.edu.au
Peter Hall Building G.83

Before we start

- Where and when to find me?
  - My office is Room G.83 at the Peter Hall building (http://bit.ly/2Guxh2C)
  - If you need help, consultation hours are **immediately after the lectures.**

Before we start

- About the assessment
  - This part of the subject has one assignment associated
  - It is expected to be released on **Monday September 23rd** and it is due on **Sunday October 13th at 23:59**
  - It is composed of three (**moderately challenging**) problems on search structures/algorithms, string matching and dynamic programming
  - Public questions on the assignment at the final 5 minutes of the lectures are encouraged
  - While individual work is to be submitted, you are strongly encouraged to discuss in groups
  - Be mindful that **coping-and-pasting** part or the whole of somebody else's solution (even if you worked together in the assignment) is called **collusion** and amounts to **plagiarism**

Before we start

- Some students **struggle** with my accent.
  - Please **stop me** if I am going too fast or I am being unclear.

- While I have taken care to check for typos, I may miss some.
  - Please let me know as soon as possible if you noted any issues, so I can correct them

- These lectures are based on Levitin's book, with some additions taken from Erickson's book (free at http://algorithms.wtf).
  - If you want to **challenge yourself**, there are plenty of (**hard**) exercises in this book, particularly for Dynamic Programming (Weeks 8 and 9).

Before we start

- I have organized each lecture around **two questions** to be answered through on-line pooling.
  - You will have the opportunity to answer the question **individually**, and then discuss your answer with your **classmates**
  - One question will focus on the **previous lecture material**, at the beginning of the session
  - One question will focus on the **current lecture material**, at the end of the session
  - In some lectures, there will be some simple exercises through out

Priority Queues

- A **priority queue** is a **set** of elements, each containing a **priority** value

- Elements are **ejected** according to their priority (**highest** first)

- Some uses:
  - Job scheduling in OSes
  - Dynamical systems simulations

Heaps

- Priority queues can be constructed using **sorted and unsorted arrays**, with **ejection or injection** being of **linear** complexity.

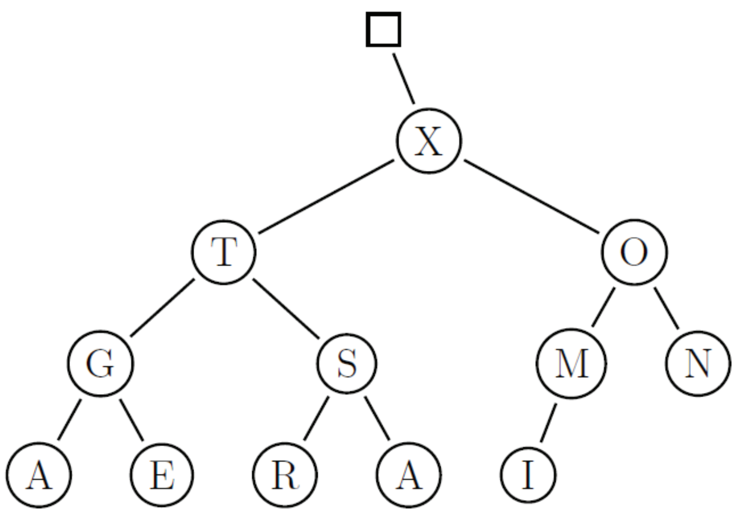
- The **heap** is a structure used to implement priority queues, with ejection and injection of **logarithmic** complexity.

- A **heap** is structured as a **complete binary tree** that satisfies the **condition**:

Each child has a priority which is no greater (lesser) than its parent's

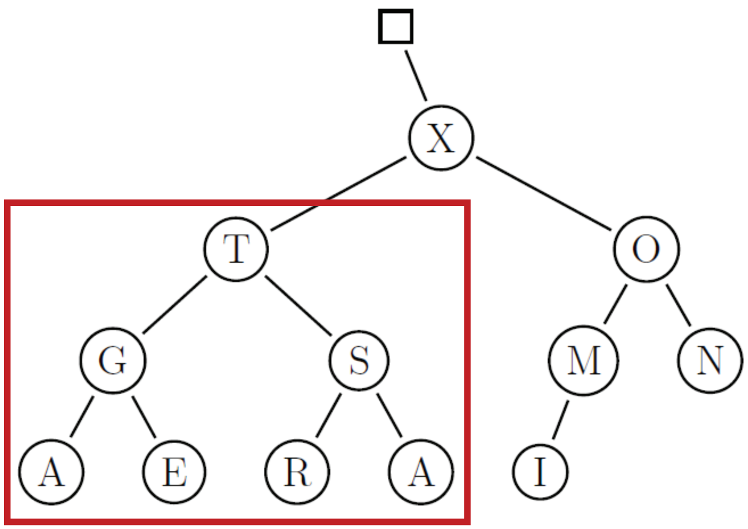
- This guarantees that the **root** of the tree is a **maximal** (minimal) element.

Implementing a heap



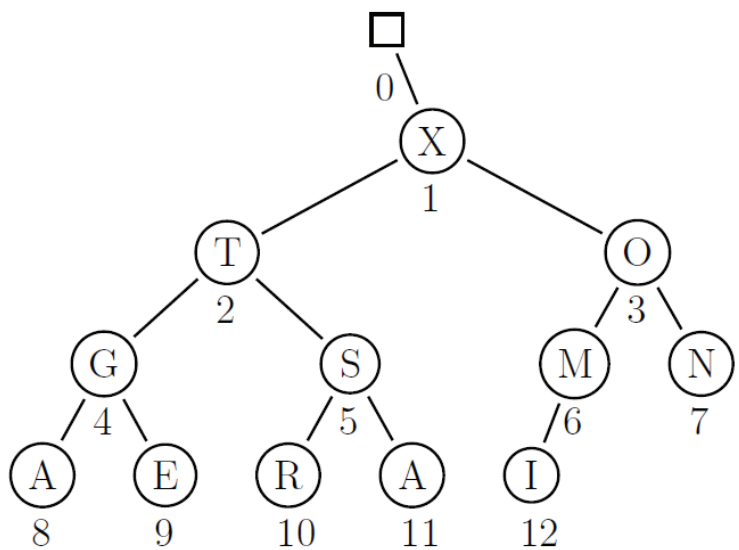
- Is this a **heap**?

Implementing a heap



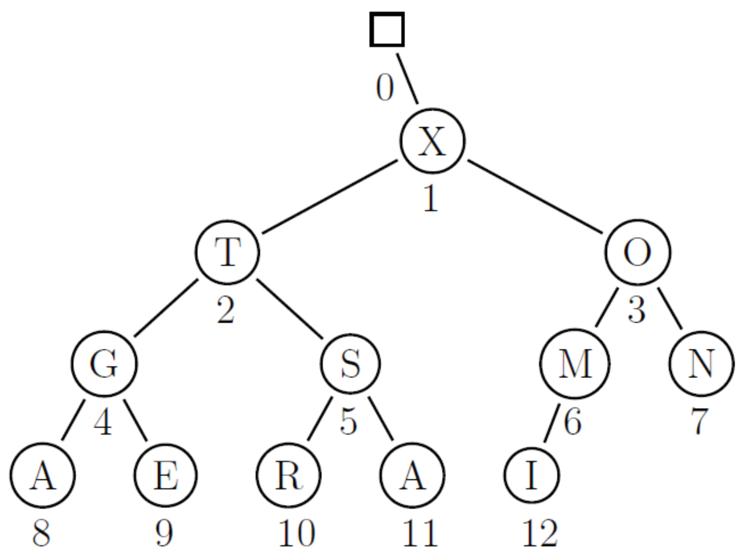
- Is this a **heap**?
    - Is a subtree a heap?

Implementing a heap



- Is this a **heap**?
  - Is a subtree a heap?
- If we label each node level by level, can you notice a pattern?

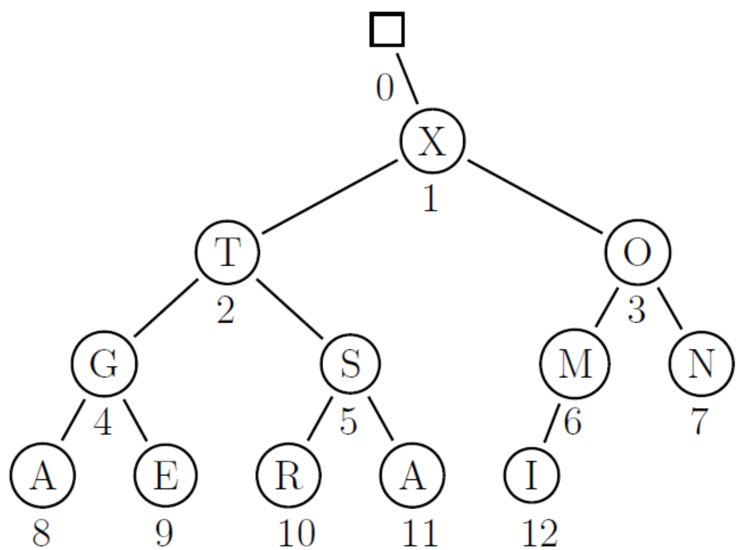
Implementing a heap



- Is this a **heap**?
  - Is a subtree a heap?

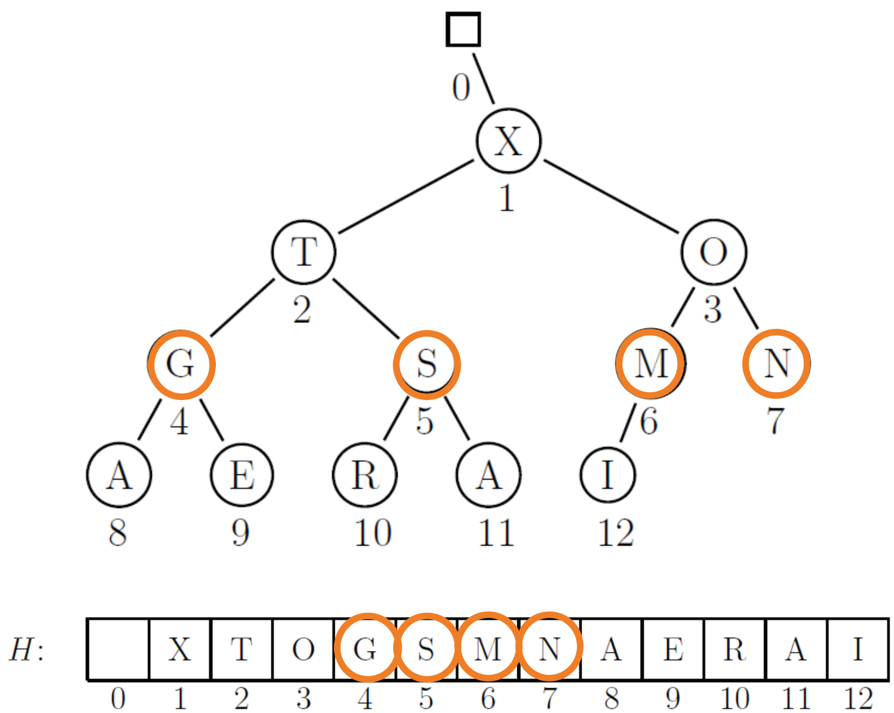
- If we label each node level by level, can you notice a pattern?
  - Did you notice that the children of node i are nodes $2i$ and $2i + 1$?

Implementing a heap



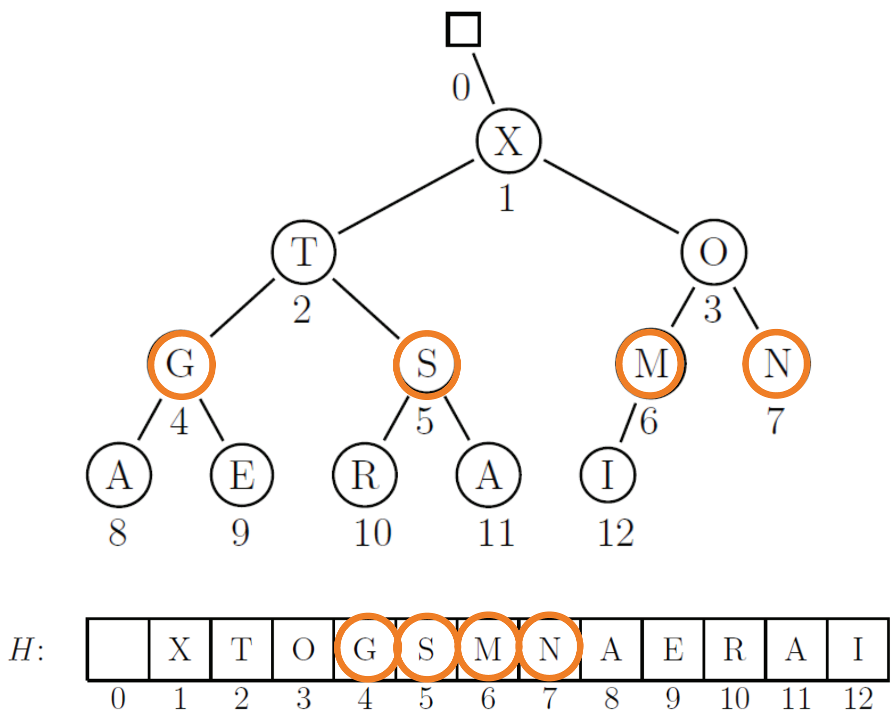
- Is this a **heap**?
  - Is a subtree a heap?
- If we label each node level by level, can you notice a pattern?
  - Did you notice that the children of node i are nodes $2i$ and $2i + 1$?
- Can we reorder the data in other way?

Implementing a heap



- Is this a **heap**?
  - Is a subtree a heap?
- If we label each node level by level, can you notice a pattern?
  - Did you notice that the children of node i are nodes $2i$ and $2i + 1$?
- Can we reorder the data in other way?
  - An array is a powerful yet simple way to **implement** a heap.

Implementing a heap

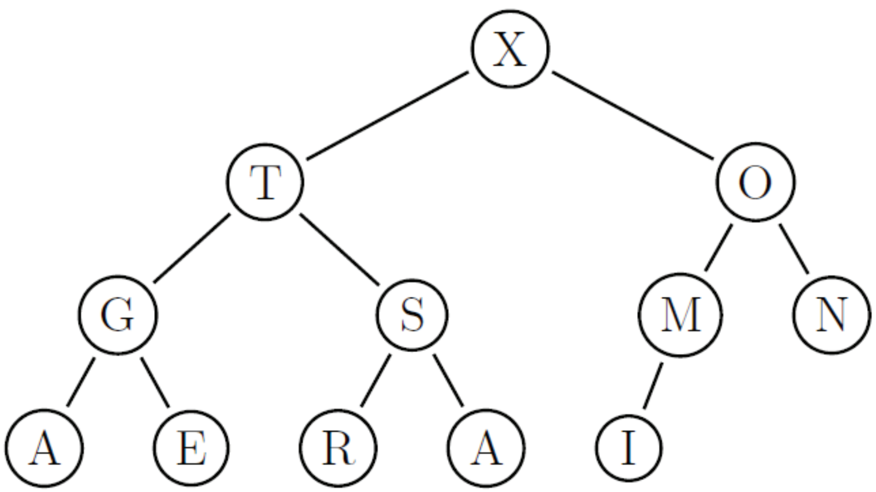


- Is this a **heap**?
  - Is a subtree a heap?
- If we label each node level by level, can you notice a pattern?
  - Did you notice that the children of node i are nodes $2i$ and $2i + 1$?
- Can we reorder the data in other way?
  - An array is a powerful yet simple way to **implement** a heap.
  - Testing the heap condition is as simple as testing $H[i] \leq H[i/2]$ for all i.

Properties of a heap

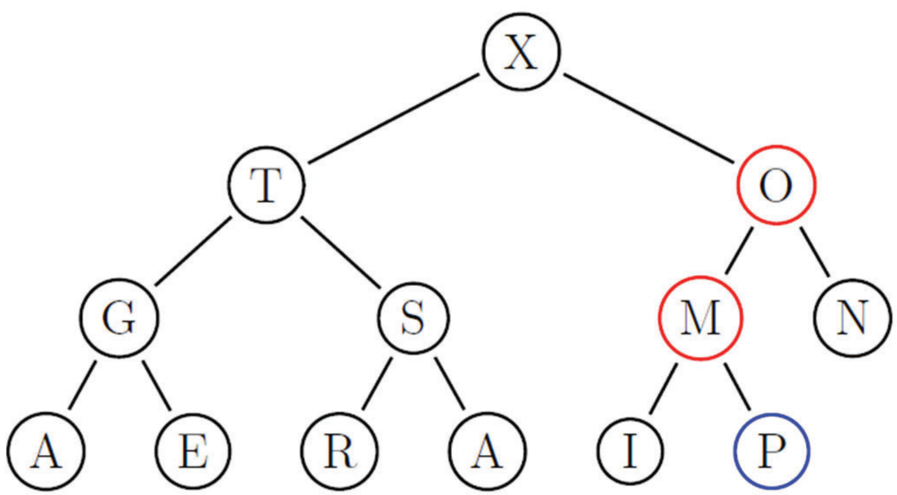
- The **height** of the heap is $\lfloor \log_2 n \rfloor$.
- Each subtree is also a heap.
- The nodes which happen to be parents are in array positions 1 to $\lfloor n/2 \rfloor$.
- The **root of the tree** $H[1]$ holds the maximal item; the cost of ejecting an element is $O(1)$ plus time to restore the heap.
  - What would happen if we eject **systematically** the maximal item?

How to inject a new item?



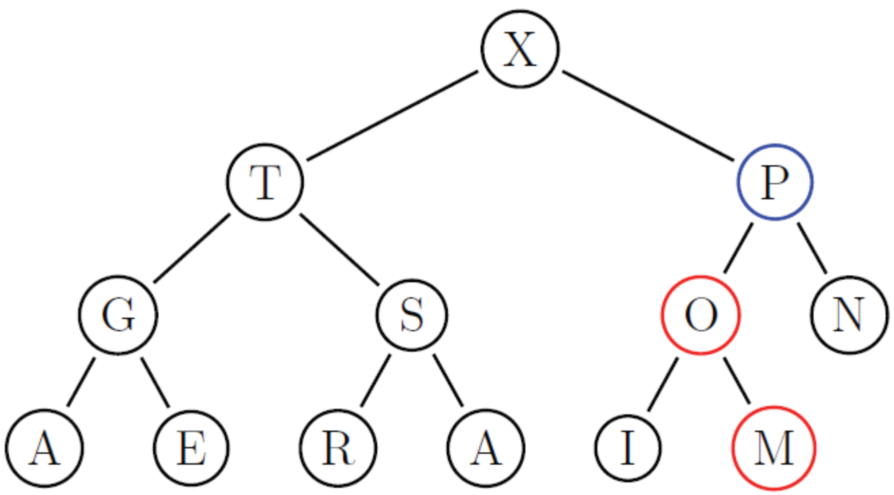
- We want to inject “P”

How to inject a new item?



- We want to inject “P”
- We place “P” at **the end**

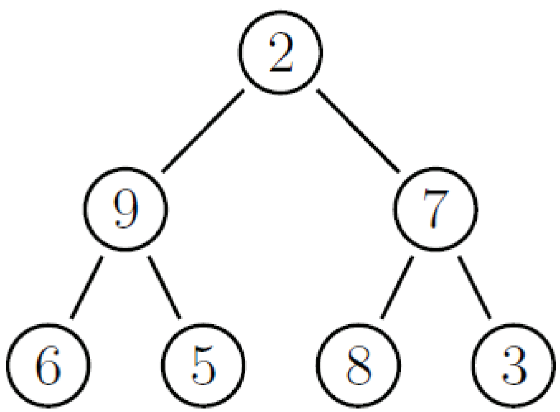
How to inject a new item?



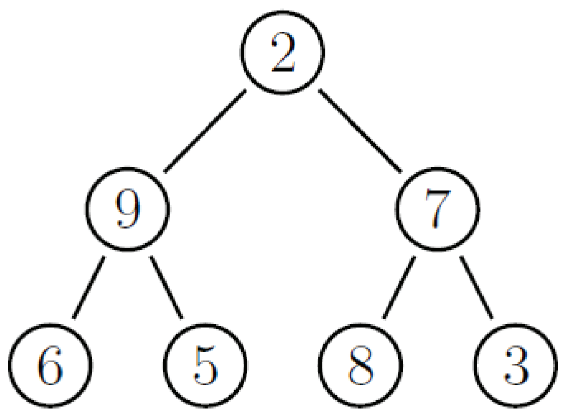
- We want to inject “P”
- We place “P” at **the end**
- We let it "climb up", **swapping** with smaller parents (“M” and “O”).
- This process has $O(\log n)$ complexity.

How to build a heap?

- Let’s assume some random data [2 9 7 6 5 8 3]

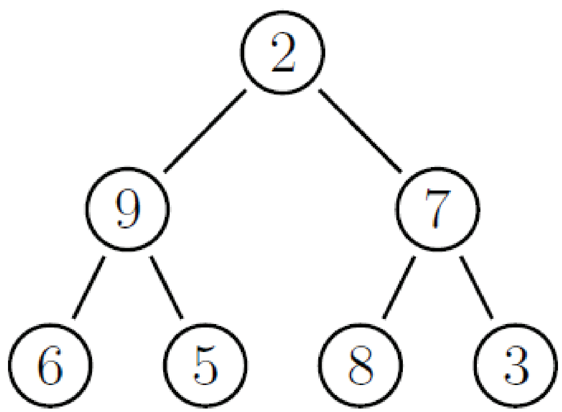


How to build a heap?



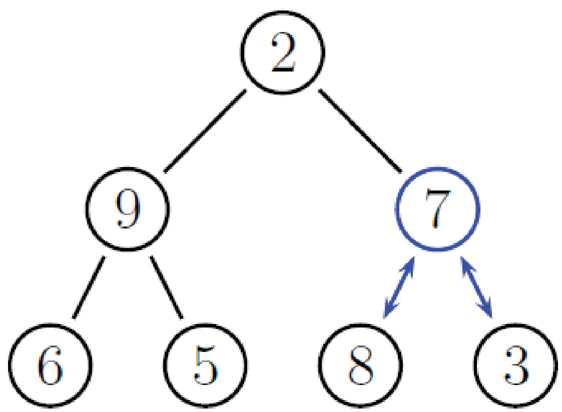
- Let's assume some random data [2 9 7 6 5 8 3]
- We could use the inject operation repeatedly

How to build a heap?



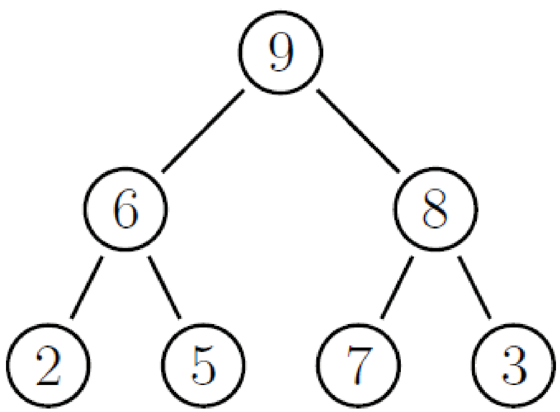
- Let's assume some random data $[2\ 9\ 7\ 6\ 5\ 8\ 3]$
- We could use the inject operation repeatedly
  - The downside is that its complexity is $O(n\log n)$

How to build a heap?



- Let’s assume some random data [2 9 7 6 5 8 3]
- We could use the inject operation repeatedly
  - The downside is that its complexity is $O(n\log n)$
- A better choice is to:
  - Start with the last parent and move backwards, in level-order.

How to build a heap?



- Let’s assume some random data $[2\ 9\ 7\ 6\ 5\ 8\ 3]$
- We could use the inject operation repeatedly
  - The downside is that its complexity is $O(n \log n)$
- A better choice is to:
  - Start with the last parent and move backwards, in level-order.
  - For parent, **if the largest child is larger**, swap it with the parent.
  - Whenever a parent is out of order, let it **sift down** until both children are smaller
  - This **bottom-down** algorithm is $O(n)$

How to build a heap?

```
for i ← ⌊n/2⌋ downto 1 do
    k ← i
    v ← H[k]
    HEAP ← FALSE
    while not HEAP and 2 × k ≤ n do
        j ← 2 × k                                  ▷ j is k's left child
        if j < n then
            if H[j] < H[j + 1] then
                j ← j + 1                          ▷ j is k's largest child
        if v ≥ H[j] then
            HEAP ← TRUE
        else                                       ▷ Promote H[j]
            H[k] ← H[j]
            k ← j
    H[k] ← v
```

How to build a heap?

```
for i ← ⌊n/2⌋ downto 1 do
    k ← i
    v ← H[k]
    HEAP ← FALSE
1 → while not HEAP and 2 × k ≤ n do
        j ← 2 × k
        if j < n then
            if H[j] < H[j + 1] then
                j ← j + 1
        if v ≥ H[j] then
            HEAP ← TRUE
        else
            H[k] ← H[j]
            k ← j
    H[k] ← v
```

Memory state (1)	
$H[1,\ldots,7]$	[2 9 7 6 5 8 3]
n	7
i	3
k	3
v	7
j	
$H[k]$	7
$H[j]$	
$H[j+1]$	
not HEAP	TRUE
$2 \times k \leq n$	TRUE
$j < n$	
$H[j] < H[j+1]$	
$v \geq H[j]$	

How to build a heap?

for $i \leftarrow \lfloor n/2 \rfloor$ downto 1 **do**
    $k \leftarrow i$
    $v \leftarrow H[k]$
    HEAP $\leftarrow$ FALSE
    while not HEAP **and** $2 \times k \leq n$ **do**
        $j \leftarrow 2 \times k$
2 → **if** $j < n$ **then**
            if $H[j] < H[j+1]$ **then**
                $j \leftarrow j + 1$
        if $v \geq H[j]$ **then**
            HEAP $\leftarrow$ TRUE
        else
            $H[k] \leftarrow H[j]$
            $k \leftarrow j$
    $H[k] \leftarrow v$

Memory state (2)	
$H[1, \ldots, 7]$	[2 9 7 6 5 8 3]
n	7
i	3
k	3
v	7
j	6
$H[k]$	7
$H[j]$	8
$H[j+1]$	3
not HEAP	TRUE
$2 \times k \leq n$	TRUE
$j < n$	TRUE
$H[j] < H[j+1]$	
$v \geq H[j]$	

How to build a heap?

for $i \leftarrow \lfloor n/2 \rfloor$ downto 1 **do**
    $k \leftarrow i$
    $v \leftarrow H[k]$
    HEAP $\leftarrow$ FALSE
    while not HEAP **and** $2 \times k \leq n$ **do**
        $j \leftarrow 2 \times k$
        if $j < n$ **then**
3 →   **if** $H[j] < H[j+1]$ **then**
                $j \leftarrow j + 1$
        if $v \geq H[j]$ **then**
            HEAP $\leftarrow$ TRUE
        else
            $H[k] \leftarrow H[j]$
            $k \leftarrow j$
    $H[k] \leftarrow v$

Memory state (3)	
$H[1, \ldots, 7]$	[2 9 7 6 5 8 3]
n	7
i	3
k	3
v	7
j	6
$H[k]$	7
$H[j]$	8
$H[j+1]$	3
not HEAP	TRUE
$2 \times k \leq n$	TRUE
$j < n$	TRUE
$H[j] < H[j+1]$	FALSE
$v \geq H[j]$	

How to build a heap?

for $i \leftarrow \lfloor n/2 \rfloor$ downto 1 **do**
  $k \leftarrow i$
  $v \leftarrow H[k]$
  HEAP $\leftarrow$ FALSE
  while not HEAP **and** $2 \times k \leq n$ **do**
    $j \leftarrow 2 \times k$
    if $j < n$ **then**
      if $H[j] < H[j+1]$ **then**
        $j \leftarrow j + 1$
4   **if** $v \geq H[j]$ **then**
      HEAP $\leftarrow$ TRUE
    else
      $H[k] \leftarrow H[j]$
      $k \leftarrow j$
  $H[k] \leftarrow v$

Memory state (4)	
$H[1, \ldots, 7]$	[2 9 7 6 5 8 3]
n	7
i	3
k	3
v	7
j	6
$H[k]$	7
$H[j]$	8
$H[j+1]$	3
not HEAP	TRUE
$2 \times k \leq n$	TRUE
$j < n$	TRUE
$H[j] < H[j+1]$	FALSE
$v \geq H[j]$	FALSE

How to build a heap?

for $i \leftarrow \lfloor n/2 \rfloor$ downto 1 **do**
    $k \leftarrow i$
    $v \leftarrow H[k]$
    HEAP $\leftarrow$ FALSE
    while not HEAP **and** $2 \times k \leq n$ **do**
        $j \leftarrow 2 \times k$
        if $j < n$ **then**
            if $H[j] < H[j+1]$ **then**
                $j \leftarrow j + 1$
        if $v \geq H[j]$ **then**
            HEAP $\leftarrow$ TRUE
        else
            $H[k] \leftarrow H[j]$
5 → $k \leftarrow j$
    $H[k] \leftarrow v$

Memory state (5)	
$H[1,\ldots,7]$	[2 9 8 6 5 8 3]
n	7
i	3
k	6
v	7
j	6
$H[k]$	8
$H[j]$	8
$H[j+1]$	3
not HEAP	TRUE
$2 \times k \leq n$	TRUE
$j < n$	TRUE
$H[j] < H[j+1]$	FALSE
$v \geq H[j]$	FALSE

How to build a heap?

```
for i ← ⌊n/2⌋ downto 1 do
    k ← i
    v ← H[k]
    HEAP ← FALSE
6 → while not HEAP and 2 × k ≤ n do
        j ← 2 × k
        if j < n then
            if H[j] < H[j + 1] then
                j ← j + 1
        if v ≥ H[j] then
            HEAP ← TRUE
        else
            H[k] ← H[j]
            k ← j
    H[k] ← v
```

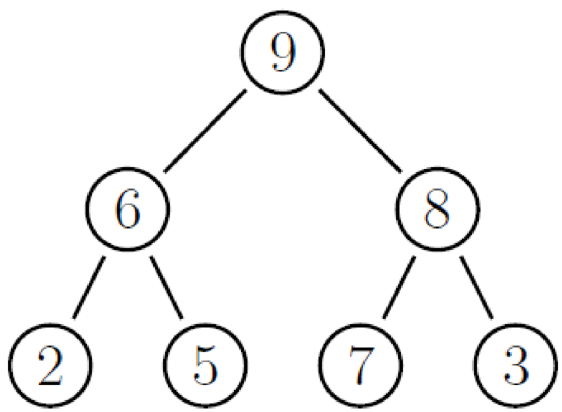
Memory state (6)	
$H[1,\dots,7]$	[2 9 8 6 5 8 3]
n	7
i	3
k	6
v	7
j	6
$H[k]$	8
$H[j]$	8
$H[j+1]$	3
not HEAP	TRUE
$2 \times k \leq n$	FALSE
$j < n$	TRUE
$H[j] < H[j+1]$	FALSE
$v \geq H[j]$	FALSE

How to build a heap?

for $i \leftarrow \lfloor n/2 \rfloor$ downto 1 **do**
  $k \leftarrow i$
  $v \leftarrow H[k]$
  HEAP $\leftarrow$ FALSE
  while not HEAP **and** $2 \times k \leq n$ **do**
    $j \leftarrow 2 \times k$
    if $j < n$ **then**
      if $H[j] < H[j+1]$ **then**
        $j \leftarrow j + 1$
    if $v \geq H[j]$ **then**
      HEAP $\leftarrow$ TRUE
    else
      $H[k] \leftarrow H[j]$
      $k \leftarrow j$
7 → $H[k] \leftarrow v$

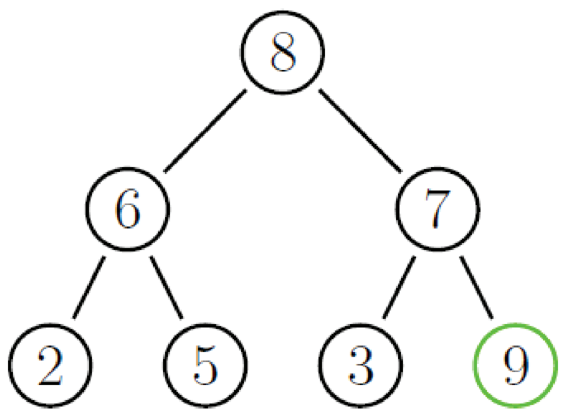
Memory state (7)	
$H[1, \ldots, 7]$	[2 9 8 6 5 7 3]
n	7
i	3
k	6
v	7
j	6
$H[k]$	7
$H[j]$	8
$H[j+1]$	3
not HEAP	TRUE
$2 \times k \leq n$	FALSE
$j < n$	TRUE
$H[j] < H[j+1]$	FALSE
$v \geq H[j]$	FALSE

How to eject an item?



- We **swap** the root with the last item z in the heap, and then let z **sift down** to its proper place.

How to eject an item?



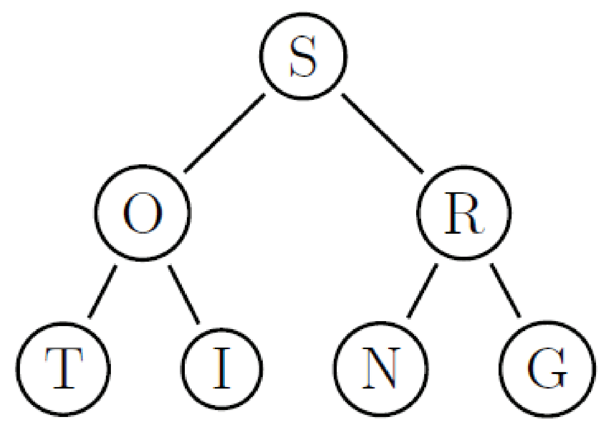
- We **swap** the root with the last item z in the heap, and then let z **sift down** to its proper place.
- The last element (in green) is **no longer** part of the heap (n is decremented).
- Ejection is $O(\log n)$

Building and then depleting a heap

- **Build** a heap from the items:

  [S, O, R, T, I, N, G]
- Repeatedly **eject** the largest

- Did you noticed a pattern?

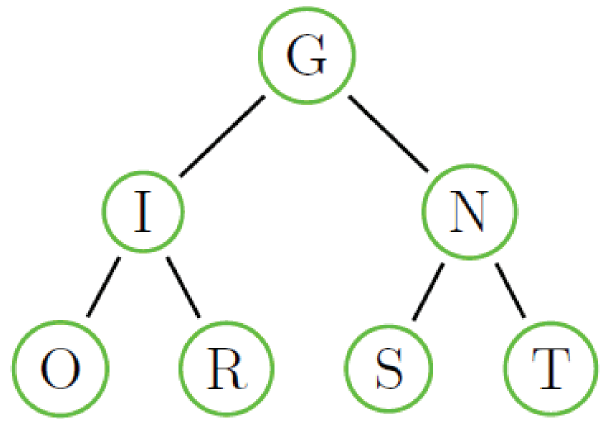


Building and then depleting a heap

- **Build** a heap from the items:

  [S, O, R, T, I, N, G]
- Repeatedly **eject** the largest

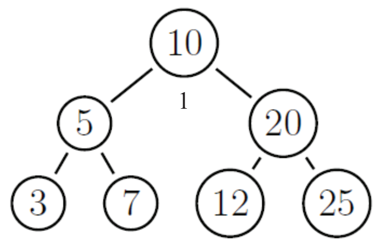
- Did you noticed a pattern?
  - This is **heapsort**, a $\Theta(n \log n)$ sorting algorithm.
  - Given an unsorted array $H[1] \ldots H[n]$:
    1. Turn H into a heap
    2. Apply the eject operation n-1 times
  - In place, not stable (operations on the heap change the relative order of equal items).



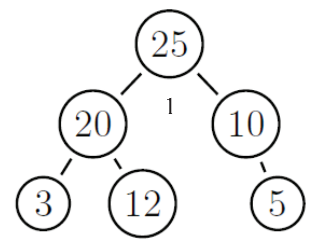
Which one is a heap?

$$\left[\ \underset{1}{25}\quad 20\quad 10\quad 3\quad \emptyset\quad 7\quad 5\ \right]$$

A: Array 1



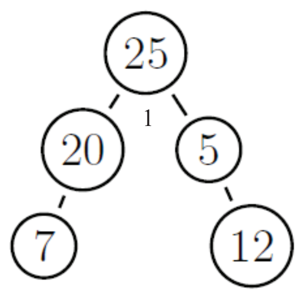
B: Tree 1



C: Tree 2

$$\left[\ \emptyset\quad \underset{1}{25}\quad 7\quad 20\quad 3\quad 5\quad \emptyset\quad 10\ \right]$$

D: Array 2



E: Tree 3

$$\left[\ \emptyset\quad \underset{1}{25}\quad 25\quad 25\quad 25\quad 7\quad 25\quad 7\ \right]$$

F: Array 3

Next lecture

- Transform-and-Conquer
    - Pre-sorting (Levitin Section 6.1)