Recursive function

**BuildHeap (e) // e is initially root**

- If e not leaf
  - BuildHeap (left child of e)
  - BuildHeap (right "")
  - Percolate-Down(e)

**RunTime:**

Seems like \( O(n \log n) \)

Percolate \( n \) nodes, each percolation takes \( \log n \)

But actually time is \( O(n) \)!

**Beautiful proof:**

1. Last here \( \rightarrow 0 \)
2. Next here \( \rightarrow 9 \)
3. Star here \( \rightarrow \) Run time \( \sim \) # swaps

Suppose always swap with left-most un-swapped descendant

Cross out node when we swap with it

# X's = # swaps \( \sim \) runtime.

How many X's? \( n = \log(n) = O(n) \).
Notice: can't start at top and swap w/ smaller

10
/  \
12   9
/ \
14 6 5 8

⇒ 4
  / \
12 10
  /  \
14 6 5 8

⇒ 4
  / \
12   5
  / \
14 6 10 8

Can already see 9 + 6 violate heap order property (9 > 6)

Another operation

Decrease Key (x, value) - decrease x's priority to value

ex: Decrease Key (25, 18) (from BuildHeap() example)

How to implement in O(log n)? Need to find x.

⇒ Keep look-up (hash) table of elements + their indices (key: element index)

(key) (index)

x | i

find index i of x in table
set array[i] = value

8
percolate array[i] up until heap order satisfied
update table (at most O(log n) updates)

25
6

5 7 10 20 30 25 35 40 15 60 50
0 1 2 3 4 5 6 7 9 10 11
Ex. \( c = 6 \)

\[
\begin{array}{ccccccc}
5 & \quad & \quad & \quad & \quad & \quad & \quad \\
7 & \quad & 20 & \quad & \quad & \quad & \quad \\
10 & 30 & 18 & 35 & \quad & \quad & \quad \\
\end{array}
\]

\[
\begin{array}{ccccccc}
5 & \quad & \quad & \quad & \quad & \quad & \quad \\
7 & \quad & 18 & \quad & \quad & \quad & \quad \\
20 & 35 & \quad & \quad & \quad & \quad & \quad \\
\end{array}
\]
Graphs - used to model pairwise relations between entities.

Graph \( G = (V, E) \) where 
- \( V \): vertices/nodes
- \( E \): edges/arcs/links \( (u, v) \) when \( u, v \in V \)

"Nodes ~ cities"
"Edges ~ roads"

Undirected graph - no particular ordering of vertices of an edge:

Vertices: \( V = \{ a, b, c, d, e \} \)
Edges: \( E = \{ (a, b), (a, c), (b, c), (b, d), (b, e), (c, d), (d, e) \} \)

- Or -
\( E = \{ (b, a), (c, a), (c, b), (d, b), (e, b), (d, c), (e, d) \} \)

Directed graphs - pair of vertices of an edge are ordered

Edges: \( E = \{ (a, b), (b, c), (b, d), (c, a), (c, d), (d, e), (e, b) \} \)

"certain roads blocked off bc of snow"
adjacency (undirected): \( u \) adjacent to \( v \) if \( (u,v) \) or \( (v,u) \) \( \in \) \( E \)

adjacency (directed): 

\[
\text{(path - sequence of vertices } v_1, v_2, \ldots, v_n \text{ such that an edge exists for every adjacent pair in the trees)}
\]

\[
\text{example: } P: c - a - b - c - d - e = (c, a), (a, b), (b, c), (c, d), (d, e)
\]

path length: number of edges in path

\[
\text{example: } |P| = 5
\]

distance of vertices \( u, v \) = length of shortest path from \( u \) to \( v \)

\[
\text{example: } \text{dist of } c, b = 2 \quad (\text{path: } c - a - b)
\]

(cycle - path of \( n \) vertices \( v_1, \ldots, v_n \) where \( v_1 = v_n \))

weighted graph: edges have weights/cost.

\( w_{u,v} \) = weight of edge \((u,v)\)

\[
\begin{align*}
\text{amount of time it takes} & \quad \text{to traverse the edge} \\
\end{align*}
\]

Can also have weighted undirected graphs.