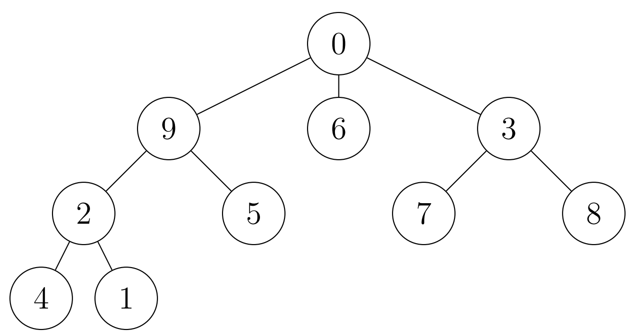
Tutorial 02 - Recurrences, trees, amortized analysis
CS 240E Winter 2023
University of Waterloo
Monday, January 23, 2023

1. **Recurrence relation.**
   Consider the following recursion: $T(0) = 0$,
   $$T(n) = n + 1 + \min_{0 \le i \le n-1} \{T(i) + T(n-i-1)\} \qquad \text{for } n \ge 1.$$
   Argue $T(n) \in \Omega(\log n)$ by showing $T(n) \ge (n+1)\log(n+1)$.

   Hint: show that $f(x) = x \log x$ is convex.

2. **Binary lifting.**
   We are given a tree on n nodes (labelled $0, \ldots, n-1$) where node 0 is the root. The tree is represented with the array *parent*, where *parent*$[i]$ denotes the parent of i (and *parent*$[0] = -1$).



	0	1	2	3	4	5	6	7	8	9
parent:	-1	2	9	0	2	9	0	3	3	0

Figure 1: Example tree and corresponding array *parent*.

   The *k-th ancestor* of node x in a rooted tree is the node we reach by moving k steps from x towards the root. In Figure 1, the 2-nd ancestor of 1 is 9.

   Our goal is to answer *many queries* of the form: given x and k, find the k-th ancestor of x.

(a) Suppose we have a black-box $anc(x, i)$ that returns 2^i-th ancestor of x in constant time.

Give an algorithm to find the k-th ancestor of x in $O(\log k)$ time.

(b) Define the two-dimensional array of integers

$$anc[0..n-1][0..\lceil \lg n \rceil - 1],$$

with the intended meaning:

$$anc[x][i] = 2^i\text{-th ancestor of } x.$$

Suppose that the tree satisfies the min-heap order property

Explain how to compute this array in $O(n \log n)$ time.

(c) In fact, we sometimes need less than $\log n$ entries in the second dimension of the array *anc*. For a fixed tree T, give an exact tight bound on the size of the second dimension of *anc*.

3. **Amortized analysis.**

We are given a binary search tree on n nodes, storing n distinct keys. We can list all keys in increasing order using in-order traversal in time linear in n.

The operation $successor(x)$ returns the in-order *successor* of x in the tree, which is the node z with $x.key < z.key$ and no other keys are stored in between (or `null` if such z does not exist), in $\Theta(\text{height of the tree})$ time.

Consider the algorithm to print all keys in the tree T in increasing order:

```
x = T.get_min()
print(x.key)
while(T.successor(x) is not null):
    x = T.successor(x)
    print(x.key)
```

(a) Give an asymptotic bound on the worst-case runtime of this algorithm, if the height of T is in $\Theta(\log n)$.

(b) Show that the amortized runtime of *successor* is $O(1)$ (and therefore the runtime of the algorithm is $\Theta(n)$).