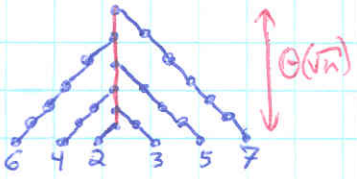
③ Long-path decomposition:

- find maximum-length root-to-leaf path
  ⇒ length = height of tree
- recurse on subtrees resulting from removal
- store each path as an array ordered by depth
  + pointer to parent path (in recursion)
- query: node v stores (its) index i in path's array

```
if ℓ ≤ i
then return array[i-ℓ]
else v ← parent(array[0])
     ℓ ← ℓ - i - 1
     repeat
```

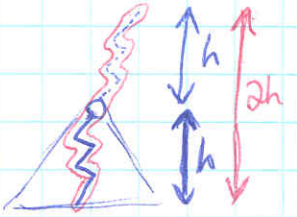
- node of height h is on path of length $\geq h$
- still might visit $\Theta(\sqrt{h})$ paths

⇒ $O(\sqrt{n})$ query



④ Ladder decomposition:

- ladder = extension of path of length ℓ up by ℓ levels
- ladders not disjoint (but still paths)
- still $O(n)$ space:
- store ladder as array ordered by depth



- v's ladder = extension of unique path containing v
- height(v) = h ⇒ height(top node in v's ladder) $\geq 2h$ or height(tree) (root)
- query: as above, but using ladders instead of paths
- now each step at least doubles height (or finishes)

⇒ $O(\lg n)$ query