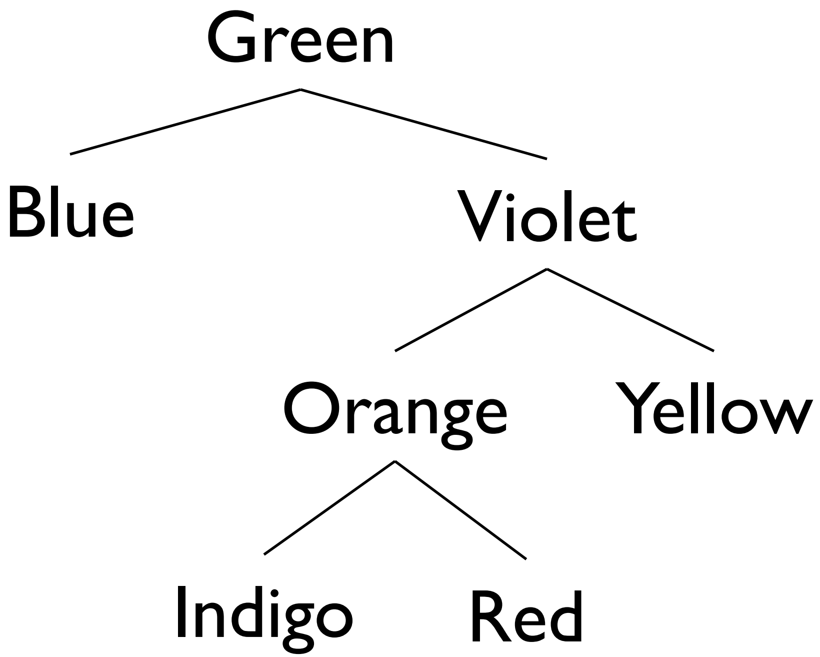
CSCI 136
Data Structures & Advanced Programming

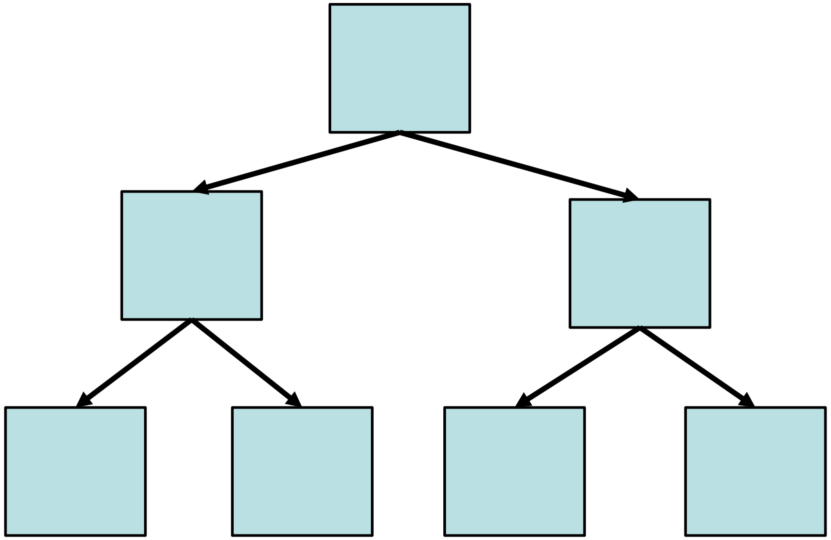
Alternative Tree Representations

BinaryTree Overheads?

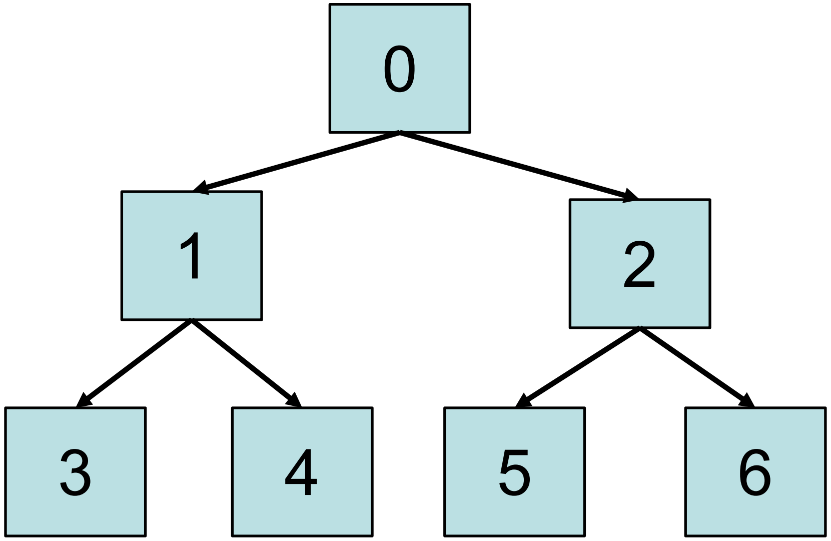


- Total # “references” = 4n
    - Since each BinaryTree maintains a reference to left, right, parent, value
- 2-4x more overhead than vector, SLL, array, …
- But trees capture successor and predecessor relationships that other data structures don’t… unless?

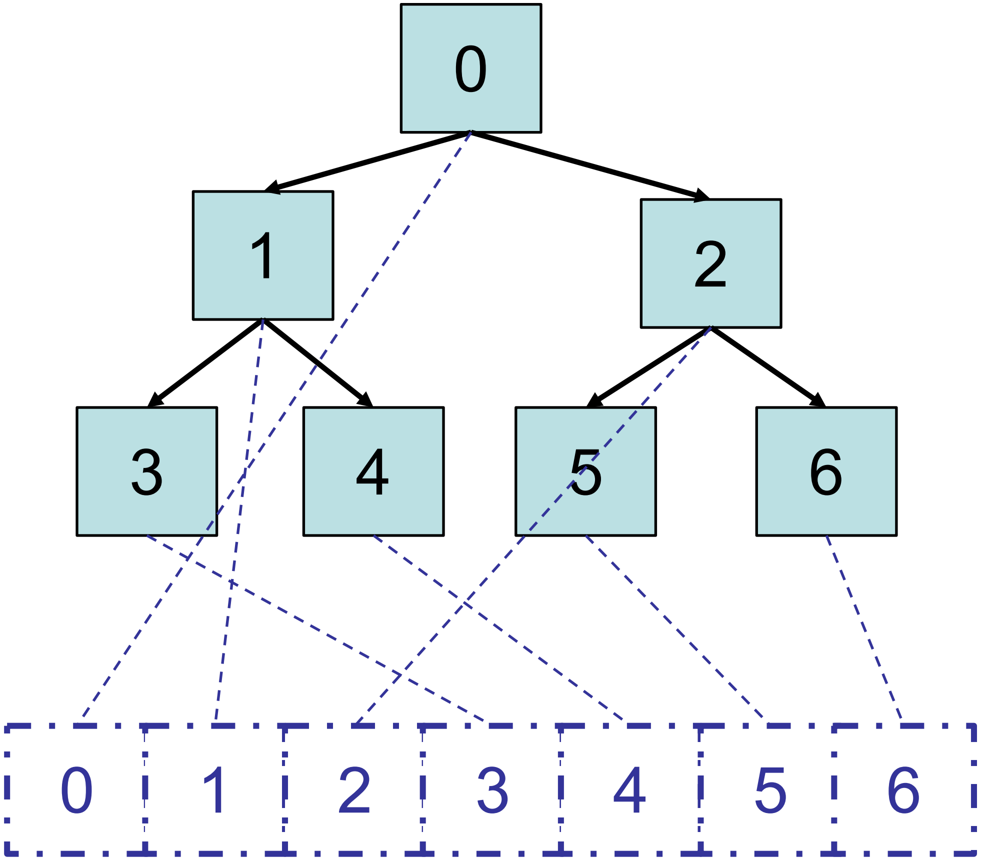
Consider the following (full) tree



Number the Nodes in BFS Order



Store them in An Array at that Index!



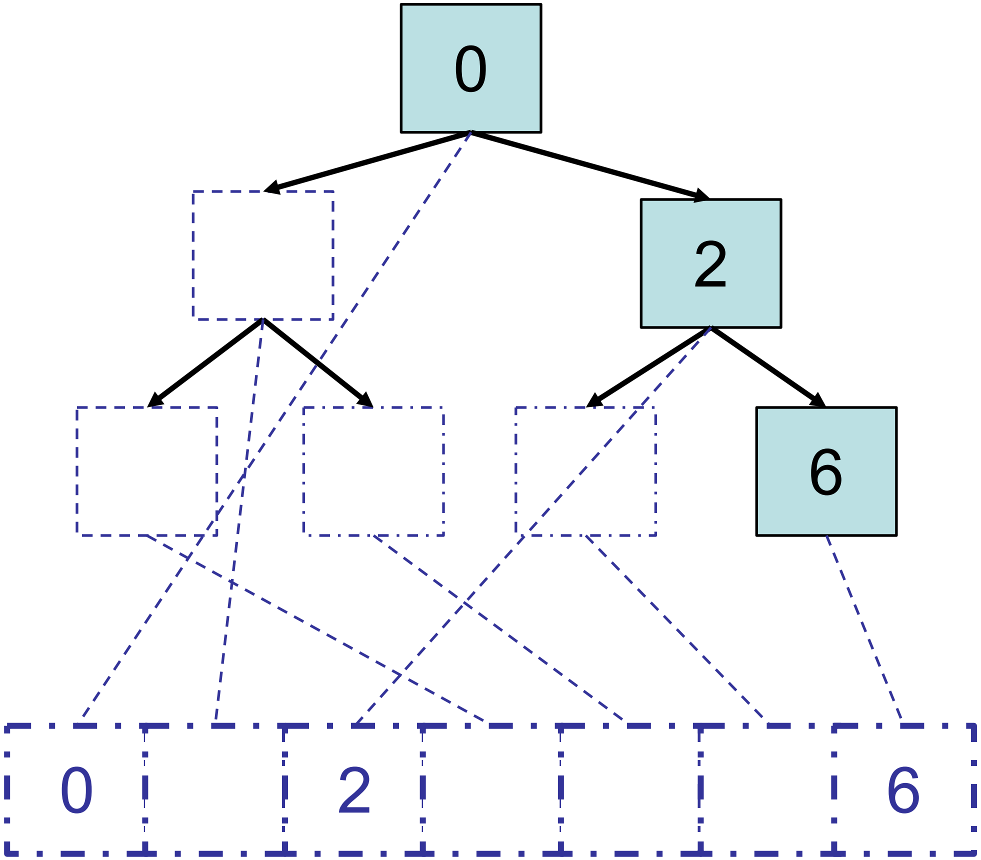
Array-Based Binary Trees

- How to encode structure of tree in an array:
- Put root at index 0
- Put the children of node at index *i* at:
  - left(i): 2i+1
  - right(i): 2i+2
- Put the parent of node j at:
  - parent(j): (j-1)/2
  - *Note*: integer truncation takes care of "rounding"

ArrayTree Tradeoffs

- Why are ArrayTrees good?
    - Save space for links
    - No need for additional memory to be allocated/garbage collected
    - Works well for full or complete trees
        - Complete: All levels except last are full and all gaps are at right
            - “A *complete* binary tree of height h is a full binary tree with 0 or more of the rightmost leaves of level h removed”
- Why bad?
    - Could waste a lot of space
    - Tree of height of n requires 2^{n+1}-1 array slots even if only O(n) elements

We Leave Gaps for Nodes That Could Exist



Final Thoughts

- For “dense” trees, an array representation is efficient
  - There are many contexts where a dense tree is a reasonable assumption
- If we can design a data structure that always preserves tree completeness, we should strongly consider an array representation
  - (Remember this when we get to heaps!)