Postorder Traversal

→ recursively traverses the subtrees rooted at the children of the root first

```
Algorithm postorder(T,p):
  for each child c in T.children(p)
    postorder(T, c)
  perform visit action for position p
```

Running Time of Preorder/Postorder

→ both pre and postorder are efficient ways to traverse

→ at each position p, the nonrecursive part of traversal requires $O(c_p+1)$ time where c_p is the # of children of p

→ the overall running time of either traversal is $O(n)$

Breadth-First Traversal (BFS)

→ a way of traversing a tree so that we visit all positions at depth d before we visit the positions at d+1

```
Algorithm breadth_first(T):
  initialize queue to contain T.root()
  while Q not empty
    p= Q.dequeue()
    perform visit for p
      Q.enqueue(c)
```

Inorder Traversal (DFS variant)

→ visit a position between recursive traversals of left & right subtrees

→ informally can be seen as visiting nodes of T from left → right

```
Algorithm inorder(p):
  if p has a left child then
     inorder(left child)
  perform visit action for p
  if p has a right child then
     inorder(right child)
```