Heap / Binary Search Tree

Hsuan-Tien Lin

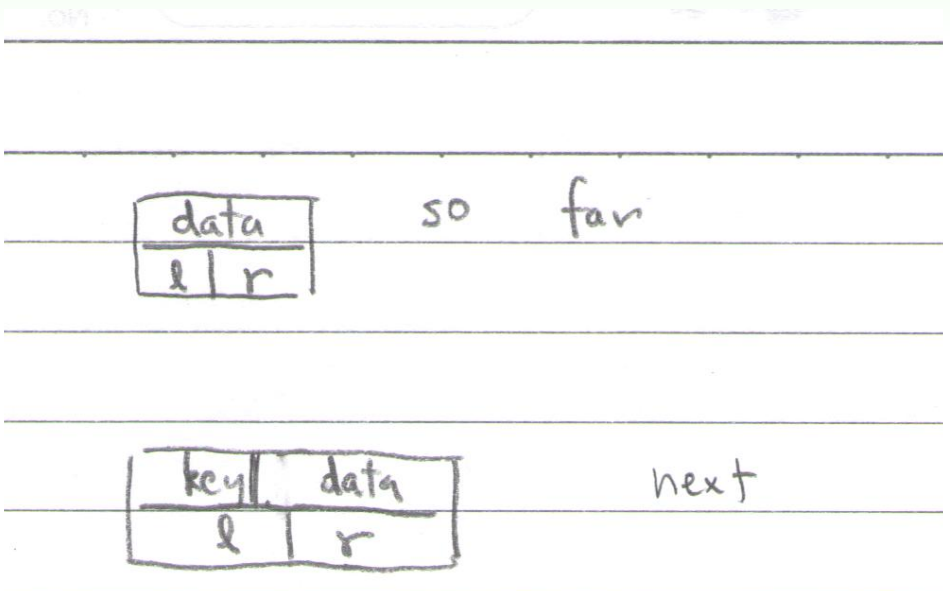
Dept. of CSIE, NTU

April 13, 2021

What We Have Done

- tree: hierarchical access
- binary tree: "simplest" tree, where (complete binary tree $\sim$ array)
- binary max-tree: binary tree + max-at-root, for priority queue

Need: Priority Queue (with Binary Tree)



- key: priority
- data: item in todo list

goal: get the node with largest key

Design Thoughts of Priority Queue with Tree (2/4)

`removeLargest` needs 2nd largest to replace

- to allow fastest removal, put 2nd largest close to next entry points
- next entry points of tree?

max tree:

- root key $\geq$ other node's key
- every sub-tree also max tree

`removeLargest` (version 0): recursive duel of sub-tree roots

- O(高)↑ O(h) O(n)↑

↑O(log n) for n elements

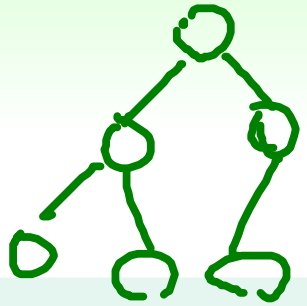
Design Thoughts of Priority Queue with Tree (3/4)

- time complexity of `removeLargest` w.r.t. h? $O(h)$

Design Thoughts of Priority Queue with Tree (3/4)

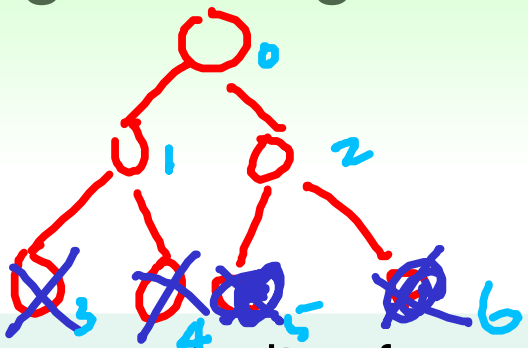
- time complexity of `removeLargest` w.r.t. h?
- to make h small w.r.t. n, need short trees
  —what is shortest binary tree with n nodes?

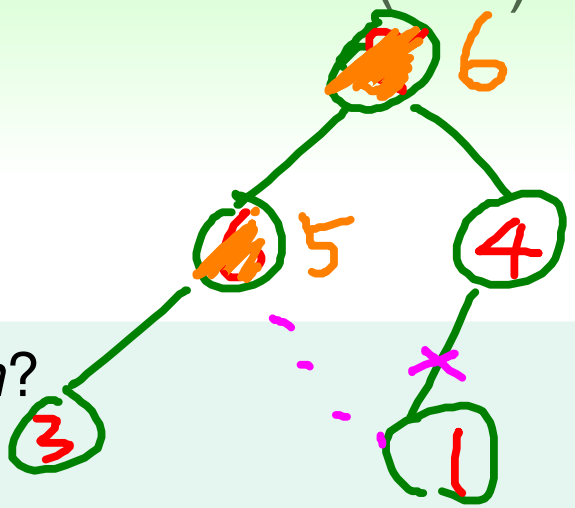
Design Thoughts of Priority Queue with Tree (3/4)



- time complexity of `removeLargest` w.r.t. h?
- to make h small w.r.t. n, need short trees
  —what is shortest binary tree with n nodes?
- (binary) max-heap: max-tree + complete binary tree

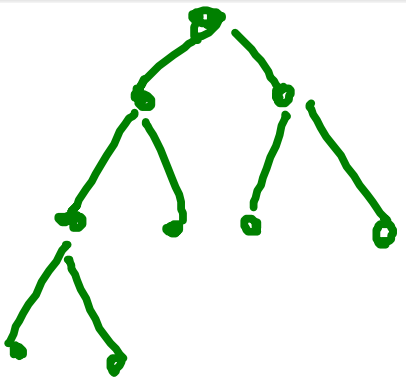
Design Thoughts of Priority Queue with Tree (3/4)

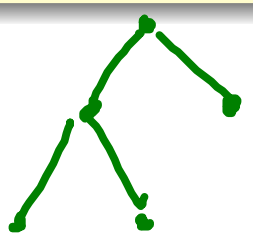




- time complexity of `removeLargest` w.r.t. h?
- to make h small w.r.t. n, need short trees
  —what is shortest binary tree with n nodes?
- (binary) max-heap: max-tree + complete binary tree

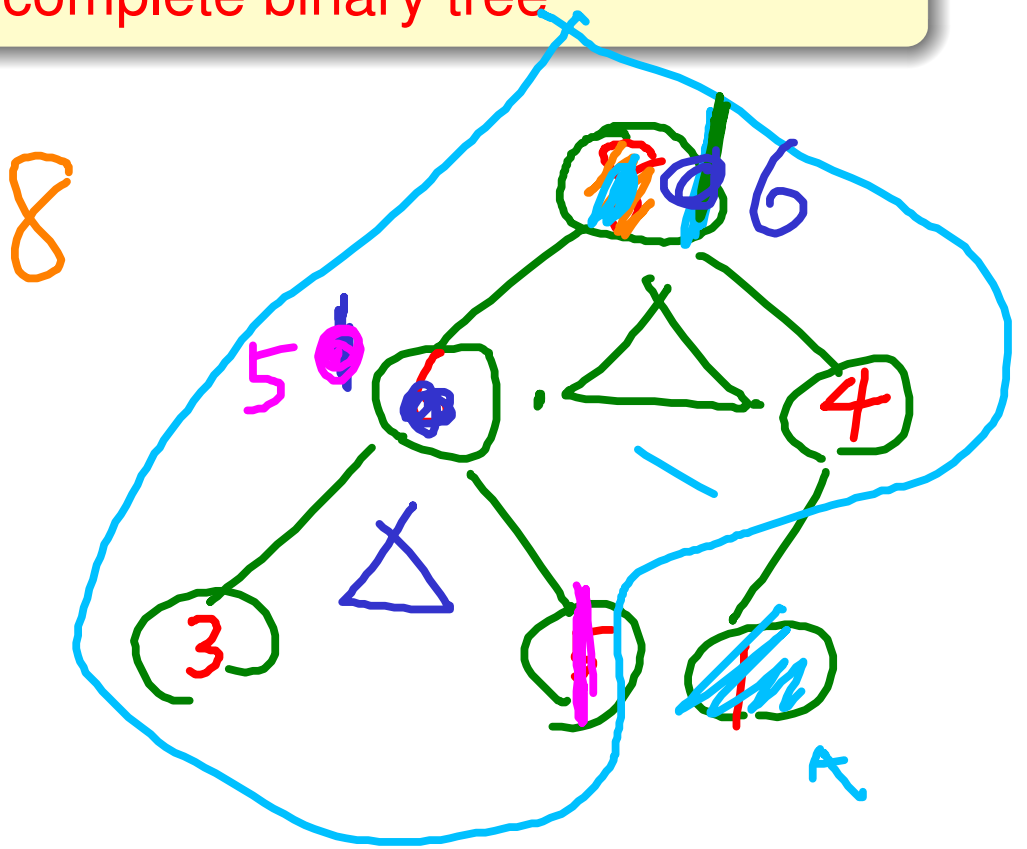
does `removeLargest` (version 0) work with binary heap?





Design Thoughts of Priority Queue with Tree (4/4)

`removeLargest` (version 0) NOT work with binary heap
—hard to preseve complete binary tree



Design Thoughts of Priority Queue with Tree (4/4)

`removeLargest` (version 0) NOT work with binary heap
—hard to preseve complete binary tree

- preserve complete binary tree first? how?

Design Thoughts of Priority Queue with Tree (4/4)

`removeLargest` (version 0) NOT work with binary heap
—hard to preseve complete binary tree

- preserve complete binary tree first? how?
- `removeLargest` (version 1)
  - move last node to root
  - three-way-duel (root & two sub-roots) in each sub-tree recursively: $O(h)$

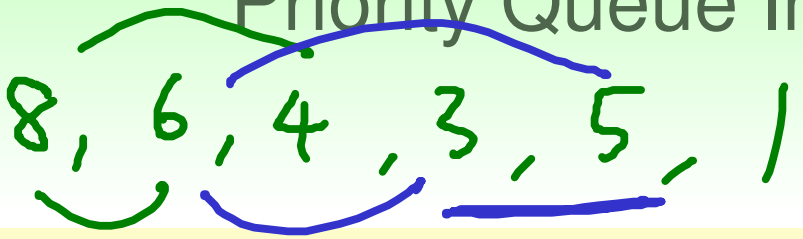
Design Thoughts of Priority Queue with Tree (4/4)

`removeLargest` (version 0) NOT work with binary heap
—hard to preseve complete binary tree

- preserve complete binary tree first? how?
- `removeLargest` (version 1)
  - move last node to root
  - three-way-duel (root & two sub-roots) in each sub-tree recursively: $O(h)$

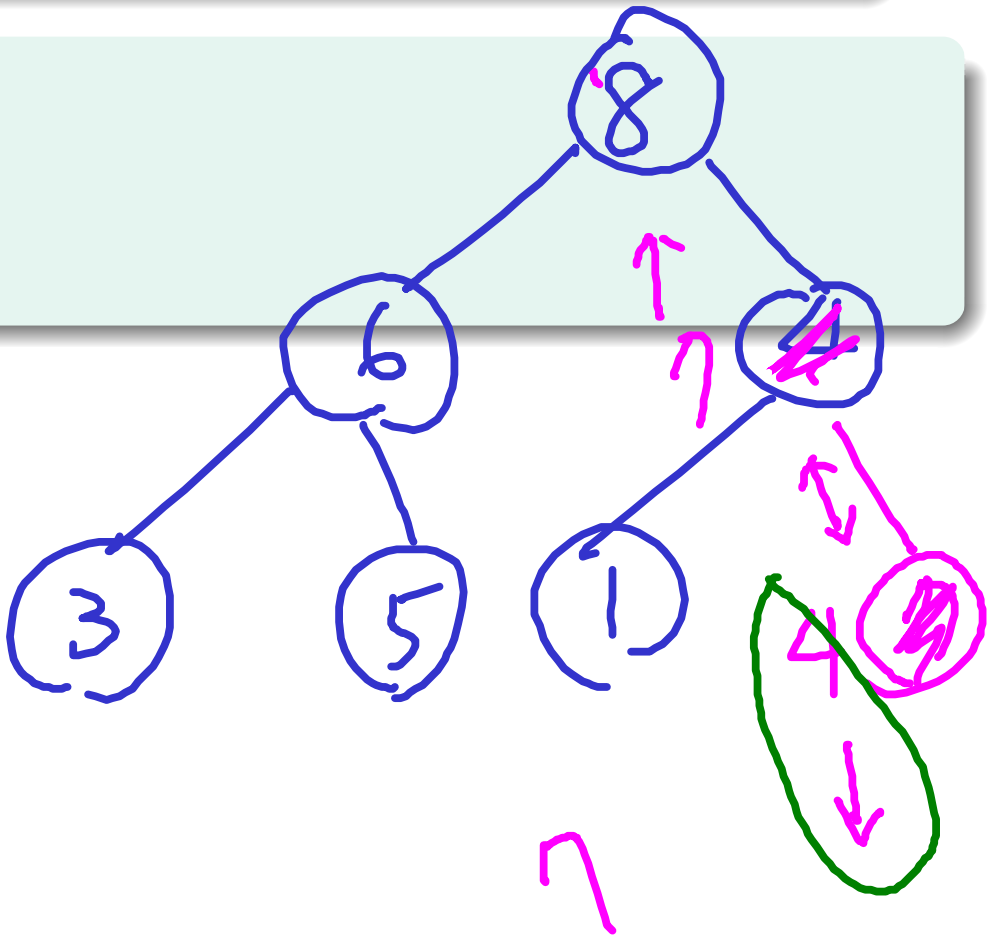
lesson learned: preserve harder constraints first

Priority Queue Insertion



binary max-heap: max-tree + complete binary tree

- insertion: preserve which first?



Priority Queue Insertion

binary max-heap: max-tree + complete binary tree

- insertion: preserve which first?
- how? insert where first?

Priority Queue Insertion

binary max-heap: max-tree + complete binary tree

- insertion: preserve which first?
- how? insert where first?
- how to preserve other property?

Priority Queue Insertion

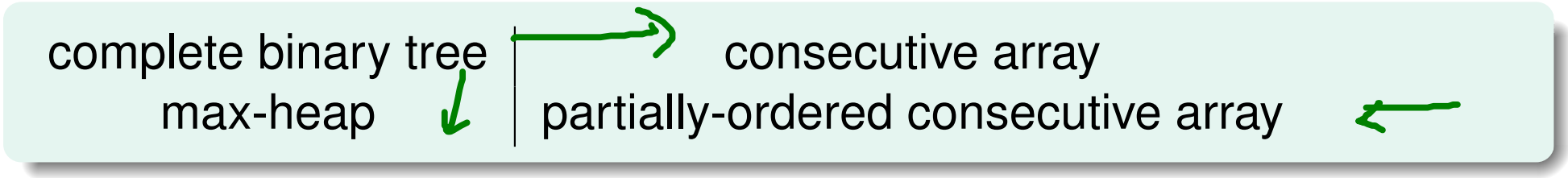
binary max-heap: max-tree + complete binary tree

- insertion: preserve which first?
- how? insert where first?
- how to preserve other property?

`insert`

- insert at last position 
- duel with parents until satisfied: $O(h)$

Binary Max-Heap in Array



Binary Max-Heap in Array

complete binary tree max-heap	consecutive array partially-ordered consecutive array

unordered array	partially-ordered array (max-heap)
selection sort $O(n) \cdot O(n)$	heap sort $O(n) \cdot O(\log n)$

先有 heap

run sel sort

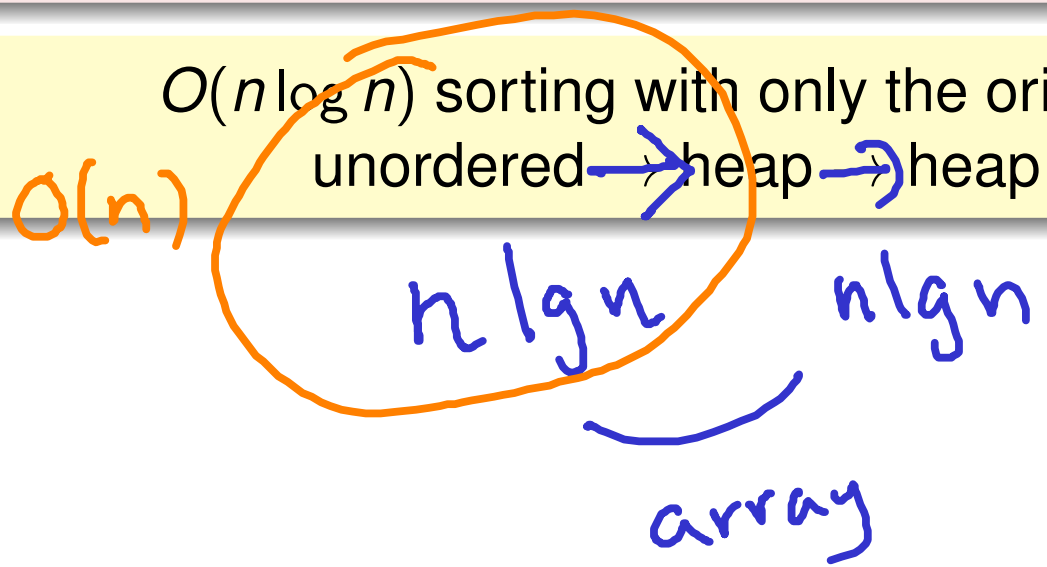
w/ heap

Binary Max-Heap in Array

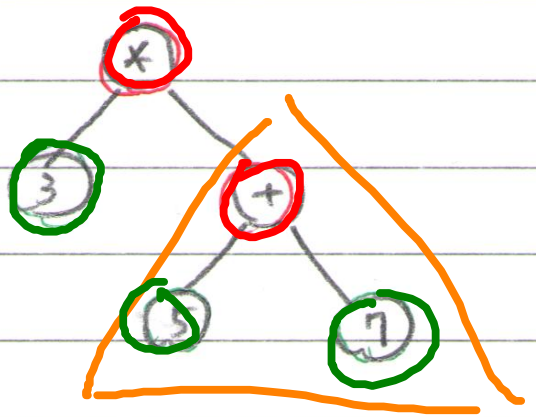
complete binary tree	consecutive array
max-heap	partially-ordered consecutive array

unordered array	partially-ordered array (max-heap)
selection sort $O(n) \cdot O(n)$	heap sort $O(n) \cdot O(\log n)$

$O(n \log n)$ sorting with only the original array:
unordered → heap → heap sort



More on Binary Trees: (Binary) Expression Tree



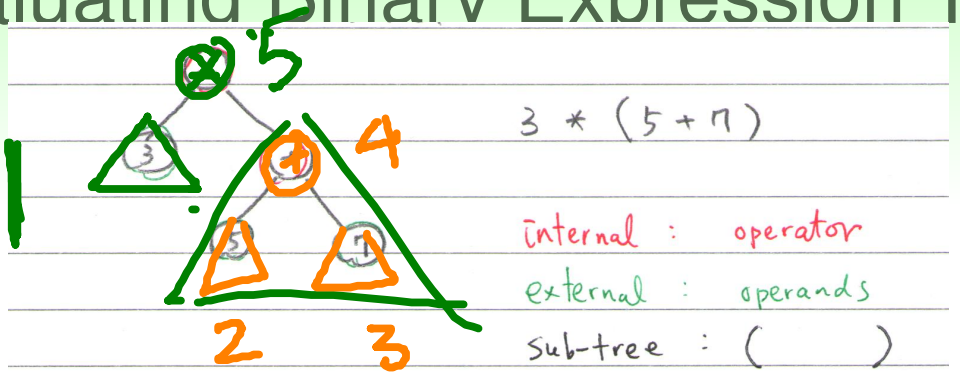
3 * (5 + 7)

internal : operator

external : operands

sub-tree : ( )

Evaluating Binary Expression Tree

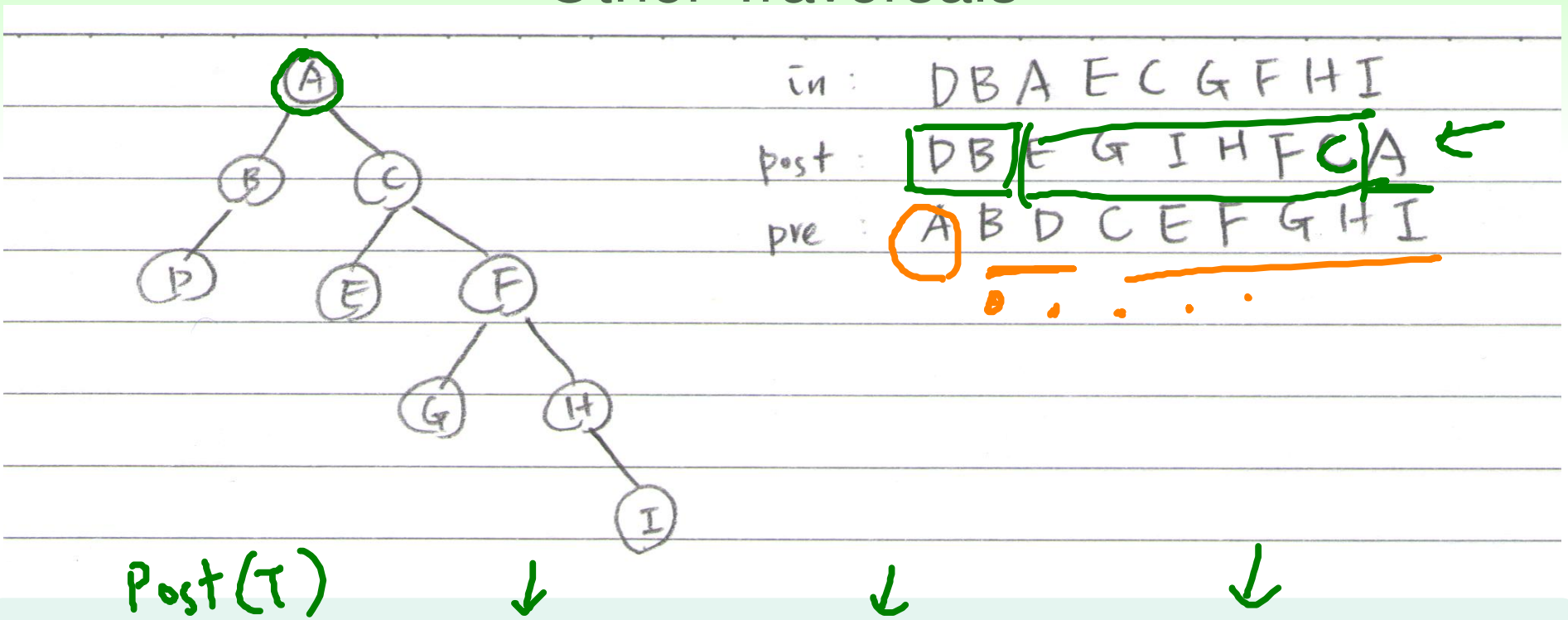



```
EVALUATE(T)
1  if T.data is a number
2        return T.data
3  else
4        a = EVALUATE(T.left) // recursion on left
5        b = EVALUATE(T.right) // recursion on right
6        return compute(a, T.data, b) // action on data
```

traversal: how $T.data$ is visited

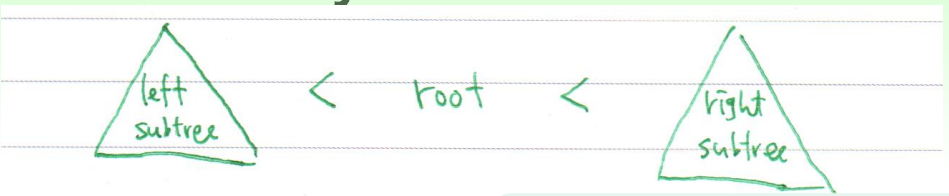
post-order traversal: $T.data$ visited after sub-tree recursions

Other Traversals



- post-order: `Post(T.left); Post(T.right); Visit(T.data);`
  —expression tree evaluation: Visit by result computation
- pre-order: `Visit(T.data); Pre(T.left); Pre(T.right);`
  e.g. comparing if two trees are equal using equality as Visit
- in-order: `In(T.left); Visit(T.data); In(T.right);`
  e.g. output ordered data from tree if $T_L < root < T_R$ (see next)

Binary Search Tree



any subtree

BST-SEARCH(T, key)

```
1
2   while T ≠ NIL
3       mid = T // root
4       if mid.key = key
5           return mid
6       elseif mid.key < key
7           return ??
8       elseif mid.key > key
9           return ??
10  return FAIL
```

complexity??

$O(h)$

$O(\log n)$

BINARY-SEARCH(A, key)

```
1   left = 1, right = A.length
2   while left ≤ right
3       mid = floor((left+right)/2)
4       if A[mid] = key
5           return mid
6       elseif A[mid] < key
7           left = mid + 1
8       elseif A[mid] > key
9           right = mid − 1
10  return FAIL
```

complexity: $O(\log n)$