# R-8.1

## **Which node is root node?**

/user/rt/courses/ is a root node as it does not have any parent and it is top node of a tree. It has childs which we say leaved of a node. Cs016/ and Cs252/ are leaves of root node.

## What are internal nodes?

All nodes are internal nodes except leaf nodes. So, /user/rt/courses/, cs016/, cs252/, grades, homeworks/, programs/, projects/, grades, papers/, demos/ are internal nodes as they have child nodes. Leaf nodes don’t have children, so they are not internal nodes.

## How many descendants does node cs016/ have?

Cs016/ have 9 descendants as it has three leaf nodes grades, homeworks/ and programs/. homeworks/ has further three nodes hw1, hw2, and hw3/. Programs/ has also pr1, pr2 and pr3 as its leaf nodes. So, 3+3+3 will be 9 descendants for cs016/.

## How many ancestors does node cs016/ have?

It has only one ancestor that is its parent named as /user/rt/courses. Upper-level nodes with related to a specific node are termed as ancestors of that specific node.

## What are siblings of node homeworks/?

Nodes that are leaf nodes for same parent are siblings. So, homeworks/ siblings are programs and grades. Hence, count of siblings is 2 for homeworks/.

## Which nodes are in subtree rooted at node projects/ ?

The lower-level nodes are termed to be included in subtree for specific node like projects/ including itself. The subtree rooted at projects/ consists of the nodes projects/, papers/, demos/, buylow, sellhigh, and market. Hence, total nodes in that subtree are 6.

## What is depth of node papers/?

Depth of node papers/ is 3 as depth is number of edges from a root node to a specific node. So, in this case 3 edges connected root to papers/ making depth of a node papers/.

## **What is Height of a tree?**

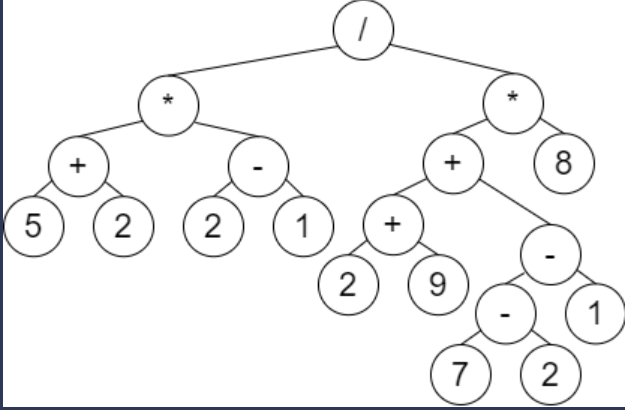
Longest path having edges that is used to connect root node to most far leaf node having maximum of edges is height of a tree. So according to figure, height of a tree is 4 as root is going maximum at leaf nodes of papers/ and demos/.

# R-8.13

(((5+2)\*(2-1))/((2+9)+((7-2)-1))\*8)

Binary Expression Tree Below:

/ is root node. In this tree, operators are root nodes and operands are leaf nodes of them.



# R-8.29

## PseudoCode

function euler\_tour(node):

if node is null:

return 0

// Start the tour at this node

tour\_start(node)

// Store the current index

index = current\_index

// Visit the left subtree

left\_descendants = euler\_tour(node.left)

// Visit this node

process\_node(node, index)

// Visit the right subtree

right\_descendants = euler\_tour(node.right)

// Calculate the total descendants

descendants = left\_descendants + right\_descendants + 1

// Store the descendants count for this node

store\_descendants\_count(node, descendants)

// Return the total descendants count for this subtree

return descendants

function tour\_start(node):

// Increment the current index

current\_index++

function process\_node(node, index):

// Store the index of this node

node.index = index

function store\_descendants\_count(node, descendants):

// Store the descendants count for this node

node.descendants\_count = descendants

// Initialize index counter

current\_index = 0

// Perform Euler tour traversal starting from the root node

euler\_tour(root\_node)

## Euler Tour Traversal:

This traversal method is a process of visiting each node in the tree three times: once before traversing its subtrees, then after traversing its left subtree, and at last after traversing its right subtree.

## Function Definitions:

euler\_tour(node): This function recursively traverses the binary tree and calculates the number of descendants for each node. It returns the total number of descendants for the subtree rooted at the current node.

tour\_start(node): This function marks the start of the tour at the current node and increments the current index counter.

process\_node(node, index): This function is called when visiting a node during the traversal. It stores the index of the node.

store\_descendants\_count(node, descendants): This function stores the computed number of descendants for the current node.

## Traversal Process:

At each node, the traversal starts (tour\_start) to mark the beginning of the tour and record the current index.

Then, it recursively visits the left subtree, computing the number of descendants (left\_descendants).

After processing the left subtree, it visits the current node (process\_node) to record its index and performs any necessary calculations.

Next, it recursively visits the right subtree, computing the number of descendants (right\_descendants).

Finally, it calculates the total number of descendants for the current node by summing the counts of its left and right subtrees plus one (the current node itself).

### Initialization:

Before starting the traversal, the index counter (current\_index) is initialized to 0.

### Execution:

The Euler tour traversal is initiated from the root node of the binary tree.

# R-9.4

An airport is developing a computer simulation of air-traffic control that handles events such as landings and takeoffs. Each event has a time stamp that denotes the time when the event will occur. The simulation program needs to efficiently perform the following two fundamental operations:

• Insert an event with a given time stamp (that is, add a future event). • Extract the event with smallest time stamp (that is, determine the next event to process).

Which data structure should be used for the above operations? Why?

Priority que will be best data structure here as insertion of data in that data structure has time complexity of O (log n ) and priority factor will let military, emergency, flights with more products to that has earlier departure time to be given priority rather than others. Moreover, when we extract data from it, it will be fast as according to Time complexity of O(1) that is constant time.

# R-9.7

Illustrate the execution of the selection-sort algorithm on the following

input sequence: (22, 15, 36, 44, 10, 3,9, 13, 29, 25).

## Initial Sequence:

(22, 15, 36, 44, 10, 3, 9, 13, 29, 25)

## Step 1:

Find the smallest element in the entire array and swap it with the first element.

Smallest element: 3 (at index 5)

Swap 3 with 22.

Sequence after Step 1: (3, 15, 36, 44, 10, 22, 9, 13, 29, 25)

## Step 2:

Consider the subarray starting from index 1 (excluding the first element) and repeat the process of finding the smallest element and swapping it with the first unsorted element.

Smallest element in subarray: 9 (at index 6)

Swap 9 with 15.

Sequence after Step 2: (3, 9, 36, 44, 10, 22, 15, 13, 29, 25)

## Step 3:

Same process will continue for other steps  
After Step 3: (3, 9, 10, 44, 36, 22, 15, 13, 29, 25)

After Step 4: (3, 9, 10, 13, 36, 22, 15, 44, 29, 25)

After Step 5: (3, 9, 10, 13, 15, 22, 36, 44, 29, 25)

After Step 6: (3, 9, 10, 13, 15, 22, 25, 44, 29, 36)

After Step 7: (3, 9, 10, 13, 15, 22, 25, 29, 44, 36)

After Step 8: (3, 9, 10, 13, 15, 22, 25, 29, 36, 44)

Final Sorted Sequence: (3, 9, 10, 13, 15, 22, 25, 29, 36, 44)

# R-9.8

Illustrate the execution of the insertion-sort algorithm on the following

input sequence: (22, 15, 36, 44, 10, 3,9, 13, 29, 25).

Initial Sequence: (22, 15, 36, 44, 10, 3, 9, 13, 29, 25)

## Step 1:

Starting with the second element (15), we compare it with the elements to its left and insert it into the correct position.

15 is smaller than 22, so we swap them.

Sequence after Step 1: (15, 22, 36, 44, 10, 3, 9, 13, 29, 25)

## Step 2:

For the third element (36), we compare it with the elements to its left and insert it into the correct position.

36 is greater than 22, so no swap is needed.

Sequence after Step 2: (15, 22, 36, 44, 10, 3, 9, 13, 29, 25)

## Step 3:

For the fourth element (44), we compare it with the elements to its left and insert it into the correct position.

44 is greater than 36, so no swap is needed.

Sequence after Step 3: (15, 22, 36, 44, 10, 3, 9, 13, 29, 25)

## Step 4:

For the fifth element (10), we compare it with the elements to its left and insert it into the correct position.

10 is smaller than 44, so we swap them.

10 is smaller than 36, so we swap them.

10 is smaller than 22, so we swap them.

10 is smaller than 15, so we swap them.

Sequence after Step 4: (10, 15, 22, 36, 44, 3, 9, 13, 29, 25)

## Step 5:

For the sixth element (3), we compare it with the elements to its left and insert it into the correct position.

3 is smaller than 44, so we swap them.

3 is smaller than 36, so we swap them.

3 is smaller than 22, so we swap them.

3 is smaller than 15, so we swap them.

3 is smaller than 10, so we swap them.

Sequence after Step 5: (3, 10, 15, 22, 36, 44, 9, 13, 29, 25)

## Step 6:

Repeat the process for the remaining elements.

After Step 6: (3, 9, 10, 15, 22, 36, 44, 13, 29, 25)

After Step 7: (3, 9, 10, 13, 15, 22, 36, 44, 29, 25)

After Step 8: (3, 9, 10, 13, 15, 22, 25, 36, 44, 29)

After Step 9: (3, 9, 10, 13, 15, 22, 25, 29, 36, 44)

Final Sorted Sequence: (3, 9, 10, 13, 15, 22, 25, 29, 36, 44)

# R-9.13

Illustrate the execution of the in-place heap-sort algorithm on the follow-

ing input sequence: (2,5, 16,4, 10, 23, 39, 18, 26, 15).

## Initial Heap Construction:

We start by constructing a max-heap from the given input sequence.

## The initial heap looks like this:

39

/ \

26 23

/ \ / \

18 16 10 15

/ \ \

5 4 2

Now, we perform heapify operations to ensure that the heap property is maintained. This involves swapping elements to move the largest element to the root of the heap.

## After heapifying, the heap looks like this:

26

/ \

18 23

/ \ / \

15 16 10 2

/ \ \

5 4 39

## After again operation:

23

/ \

18 16

/ \ / \

15 4 10 2

/ \ \

5 26 39

We repeatedly extract the maximum element from the root of the heap and place it at the end of the array. Then, we adjust the heap to maintain the heap property.

After the first extraction: (39, 18, 16, 15, 5, 4, 10, 2, 26, 23)

After the second extraction: (26, 18, 16, 15, 5, 4, 10, 2, 23)

After the third extraction: (23, 18, 16, 15, 5, 4, 10, 2)

After the fourth extraction: (18, 15, 16, 2, 5, 4, 10)

After the fifth extraction: (16, 15, 10, 2, 5, 4)

After the sixth extraction: (15, 5, 10, 2, 4)

After the seventh extraction: (10, 5, 4, 2)

After the eighth extraction: (5, 2, 4)

After the ninth extraction: (4, 2)

After the tenth extraction: (2)

Final Sorted Sequence: (2, 4, 5, 10, 15, 16, 18, 23, 26, 39)