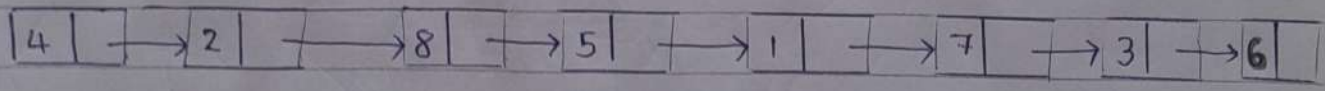
Ananya Ganapati Hegde
1RV24MC017
MCA 'I' semester 'A' section

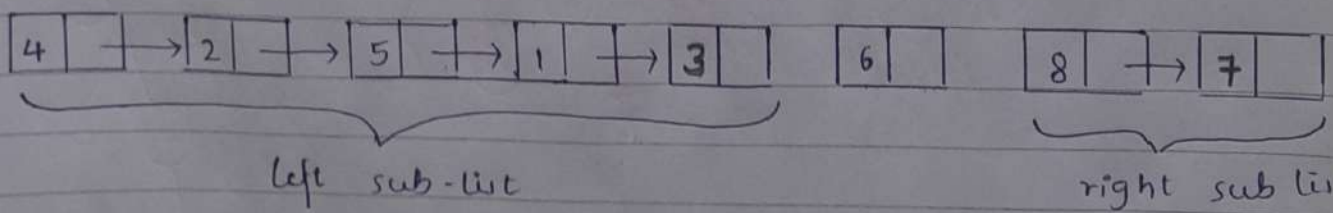
Sort a Linked List using quick sort

Quick sort is a sorting algorithm that follows the divide and conquer approach to sort a given array or any contiguous data structure. It works by selecting a pivot element from the list and compares it with each element in the list. If an element is less than pivot element it will go to the left sub-list and if its greater it will be put into right sub-list. Quick sort repeatidly partitions the current list until it can be no longer divided. At this point all the values will be in a sorted manner and we put this into new linked list which will be the sorted linked list.

~~Expte~~ Example: Sorting below linked list using quick sort

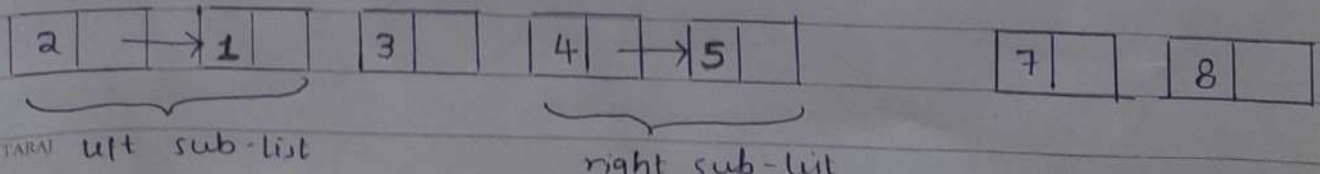


Iteration 1: choose 6 (last element as pivot)



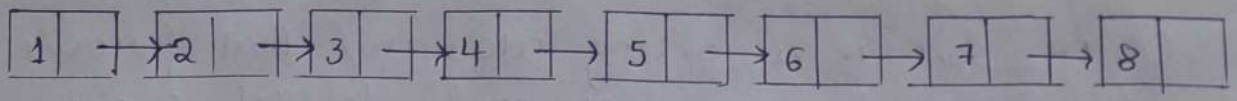
Iteration 2: pivot = 3

pivot = 7



Iteration 3: pivot = 1 , pivot = 5

Now we can no longer devide the sub partitions any furth
Combining these sublist into one linked list -



Which is a sorted list.

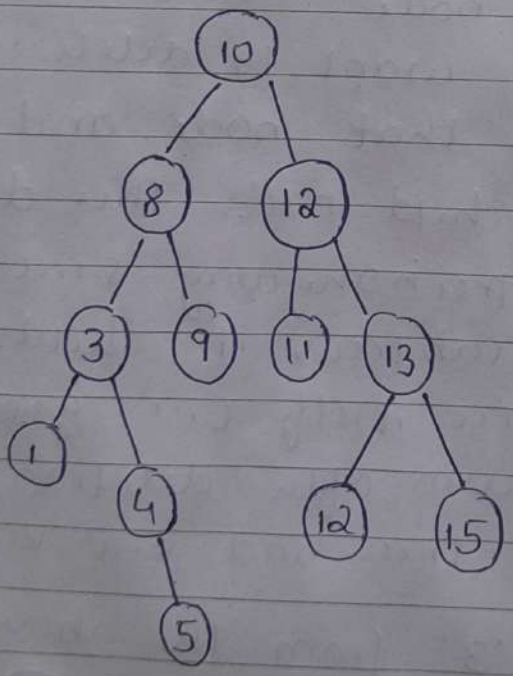
Time complexity : O(n log n)

Anamya Ganapati Hegde
1RV24MC017
MCA 'I' semester 'A' section

Deletion of a node in Binary Search Tree

'Binary Search Tree' is a tree data structure (i.e., consists of root element / node and descendents of root node (child nodes) and leaf nodes) where each node has either 0, 1 or 2 children such that all the nodes on the left of that node (left sub-tree) are lesser than the value of that node, and all the nodes on the right (right sub-tree) are greater than that node.

example:



Binary Search Tree

* All the nodes on the right sub-tree are greater and left sub tree are lesser at every level.

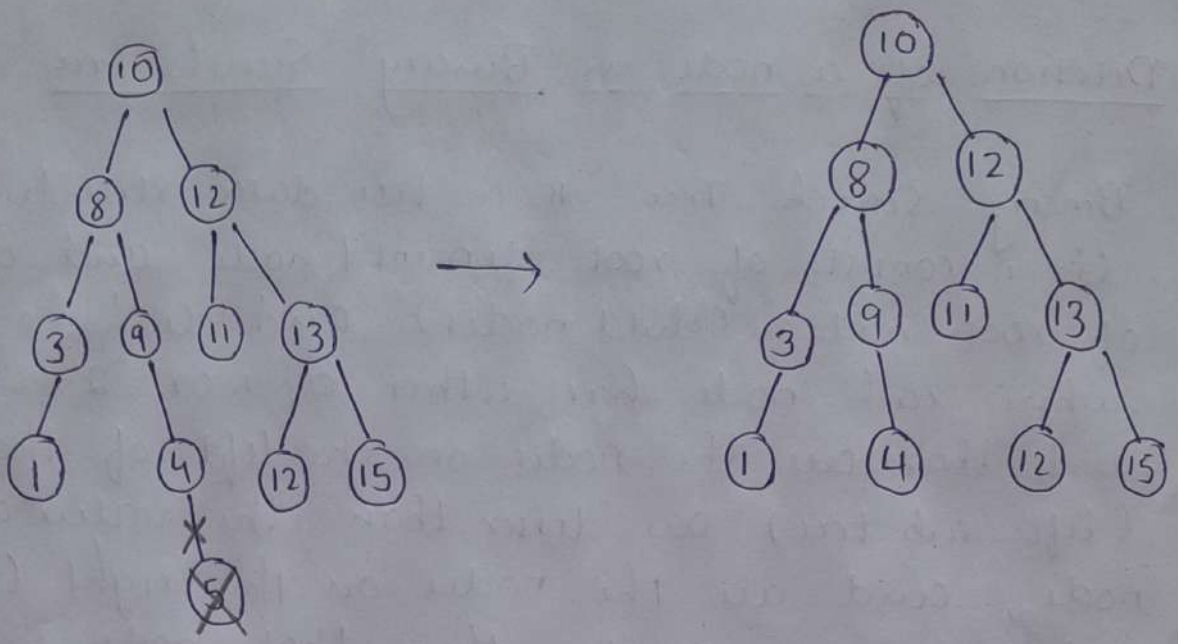
Deleting a node from Binary Search Tree

→ Case 1: Deleting a node that has no child.

This is the simplest case of deletion in BST. If a node is a leaf node (has no child)

we can simply remove the node by deleting the reference of that node and reclaiming the memory allocated.

example: Deleting '5' from the above BST.

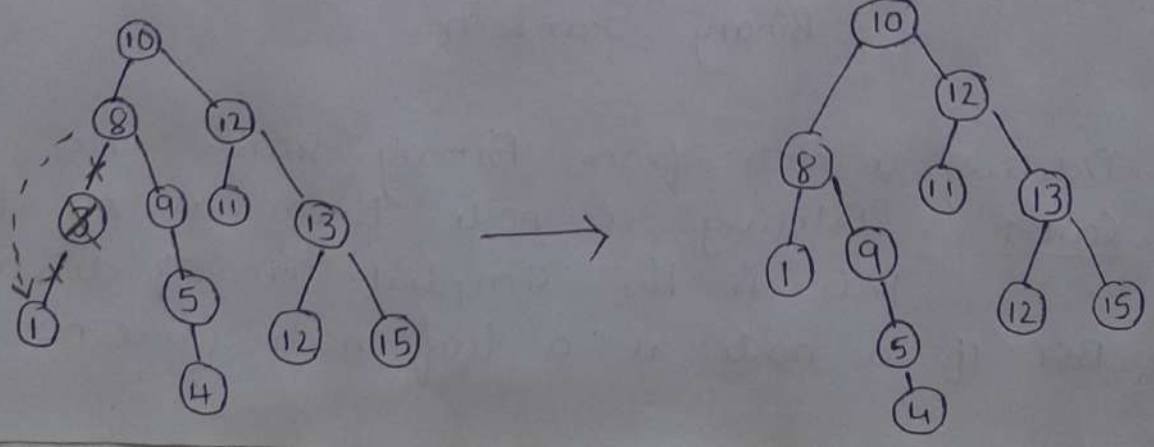


1, 11, 12, 15 are all leaf nodes and can be deleted this way.

⇒ Case 2: Single child node

When the node we want to delete has a one child we can delete that node, and link it's parent to its only child node. This does not exploit the binary search tree in anyway since - if the node is on the left subtree, its children will also be on the left sub-tree itself (i.e., lesser than the parent node). Hence even after deletion, it would still be a binary search tree and vice versa

example: Deleting '3' from the above BST



9 and 5 are also single child nodes and can be deleted this way.

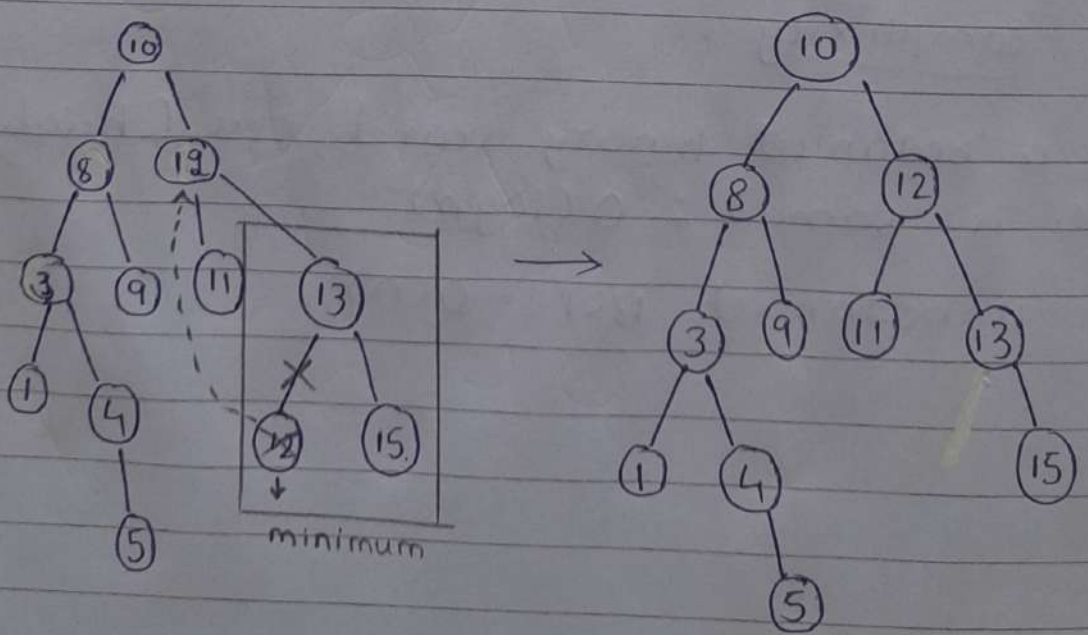
⇒ Case 3: 2 children node

Solution 1: Find the minimum node in the right subtree, copy the value in the node you want to delete. Delete the duplicate value in the right sub tree.

Why it works?

Since we are finding minimum from the right sub tree (all the nodes are greater than the node) it would still be lesser than or equal to nodes in the right subtree. Since it is from right subtree it is still greater than left sub tree. Hence even after deletion it would still be a binary search tree.

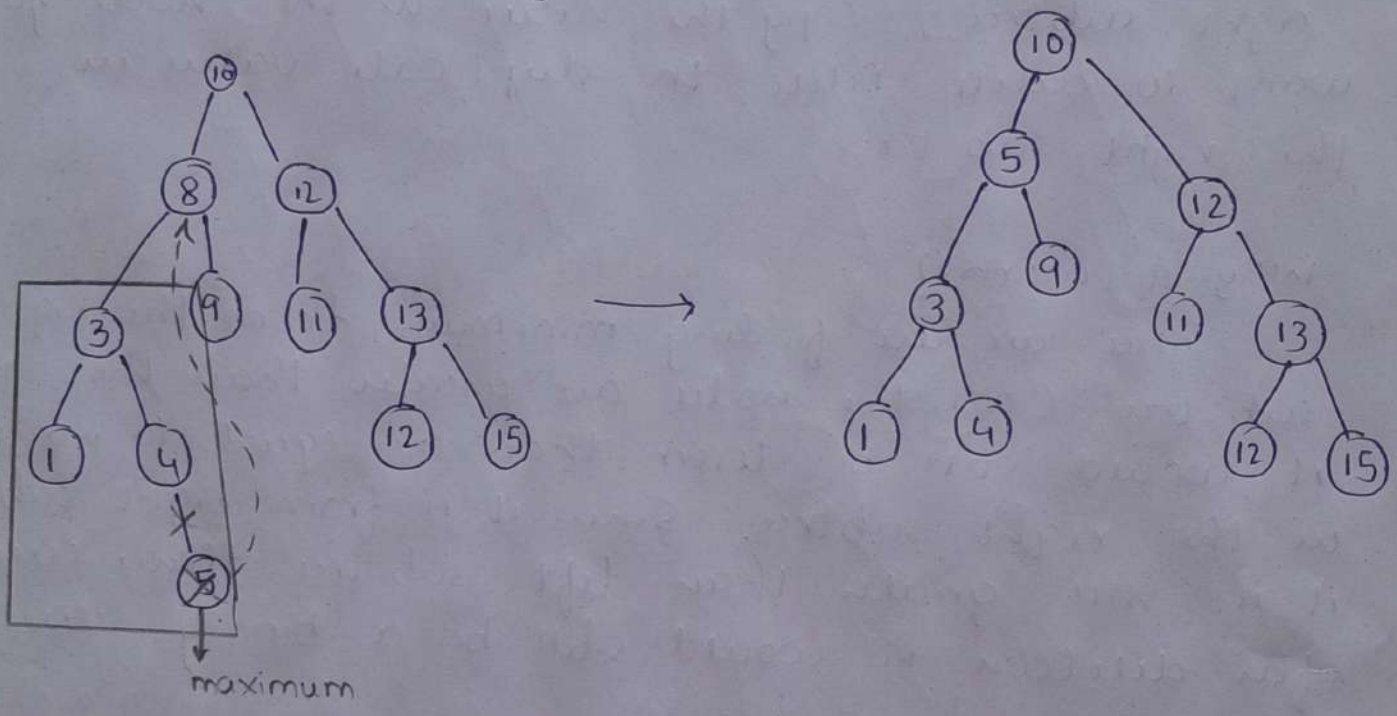
Example: Deleting '12' from above BST



Solution 2: Find the maximum in the left sub-tree copy the value to the node you want to delete remove the duplicate node in the left sub-tree.

This is another possible way to delete a node with 2 children, works the way similar to the solution proposed in solution 1.

example: deleting '8' in the above BST



* Time complexity

For a balanced binary search tree (height of each node is same): O(logn)

For unbalanced BST: O(n)

Name: LUBNA TABASSUM
USN: 1RV24MC062
SAP ID: RVCE24MCA039
Class: MCA I sem
Section: B

Skill Lab

Group Activity

Topic: 01

Array

"FIND LARGEST SUM CONTIGUOUS SUBARRAY"

Here, the problem statement "Find largest sum contigous subarray" says that - Among all the subarrays of a array find the subarray that gives maximum sum.

Example:
Let us consider an array:
arr = [-2, -3, 4, -1, -2, 1, 5, -3]

Possible subarrays are:-
[-2], [-3], [4], [-1], [-2], [1], [5], [-3]
[-2, -3], [4, -1], [-2, 1] [5, -3]
[-2, -3, 4], [4, -1, -2], [-2, -3, 4, -1],
[4, -1, -2, 1, 5]

Sum of elements in subarray:-
-2 + (-3) = -5   4 + (-1) = 3 . . . .
. . . .
-2 + (-3) + 4 + (-1) = -2   4 + (-1) + (-2) + 1 + 5 = 7

Here the largest sum of a subarray is

$[4, -1, -2, 1, 5] = 7$

This is the solution.

→ In order to find the solution for the problem statement "Find the largest sum contigous subarray there are many ways.
One of the best solution approach is by using "Kadane's algorithm".

KADANE'S ALGORITHM

Kadane's Algorithm is a dynamic programming technique, to find the largest sum of a continous subarray within 1-Dimensional array of elements.

* Kadane's Algorithm can also be used to find largest sum of submatrix within 2-Dimentional or multi-dimensional array.

Steps in Kadane's Algorithm:

Step 1: Initialize variables.

- max_so_far = $-\infty$ → To keep track of the maximum sum found so
- max_ending_far_here = 0 → To keep track of the sum of the

current subarray.

Step 2: Iterate through the array.

- For each element num in the array:
  1. Add num to max_ending_here
  2. If max_ending_here is greater than max_so_far, update max_so_far
  3. If max_so_ending_here becomes negative, reset it to 0.

Step 3:
Return max_so_far as the maximum sum of the contiguous subarray.

Algorithm (Pseudocode):

```
Initilize max_so_far = -∞
Initilize max_ending_here = 0
For each element num in the array:
    add num to max_ending_here
    If max_ending_here > max_so_far:
        update max_so_far = max_ending_here
    If max_ending_here < 0:
        Reset max_ending_here = 0
Return max_so_far
```

Let us take program example and understand kadane's algorithm:

The following code is written in python programming language.

Code:

```
def max_subarray_sum_kadane
                    (arr):
    max_so_far = float('-inf')
    max_ending_here = 0

    for num in arr:
        max_ending_here += num
        max_so_far = max(max_
            so_far, max_ending
            here)
        if max_ending_here < 0:
            max_ending_here = 0

    return max_so_far
# Example usage
arr = [-2, -3, 4, -1, -2, +1, 5, -3]
print("Largest sum:", max_sub-
-array_sum_kadane(arr))
```

output:-

Largest sum (kadane's Algorithm): 7

Step by Step execution of program:

The above code of Kadane's algorithm to find largest sum subarray works in the following ways:-

The input given is
arr = [-2, -3, 4, -1, -2, 1, 5, -3]
Execution is as follows:-

Index	Element	max_ending_here (current sum)	max_so_far (Max sum found)
0	-2	-2	-2
1	-3	-3	-2
Since sum can be -ve, the above will be reset to 0			
2	4	4	4
3	-1	4+(-1)=3	4
4	-2	4+(-1)+(-2)=1	4
5	1	4+(-1)+(-2)+1=2	4
6	5	4+(-1)+(-2)+1+5=7	7 (updated)
7	-3	4+(-1)+(-2)+1+5-3=4	7

Here, the largest sum is 7.
Therefore the variable max_so_far will return 7.
The subarray that gives largest sum is
[+4, -1, -2, 1, 5] = 7

Stack & Queue

"IMPLEMENT STACK USING DEQUE"

Before learning how to implement stack using deque, let us understand what is stack & deque.

Stack:

A stack is a linear data structure that follows: LIFO rules (Last In First Out). This means the element inserted last will be removed first.

Deque (double-ended queue):

A deque (pronounced as "deck" is a double-ended queue where elements can be inserted or deleted from both the ends. (front & rear end).

Implementation of stack using deque:

We can implement a stack using deque (double-ended queue) from Python's collections module, as it provides efficient O(1) time complexity for both insertion & deletion at the end.

CLASS	SUBJECT	
ROLL NO.	DATE	

Steps to implement a stack using deque.

Step 1: Initialize the stack
- create a class Stack
- inside the constructor (__init__) define empty deque to store stack elements.

Step 2: Push operation
- use deque.append(x) to insert an element at the right end.

Step 3: pop operation
- use deque.pop() to remove the last inserted element

Step 4: peep operation
- use deque[-1] to retrieve the top element without removing it.

Step 5: check if stack is empty
- check if the deque has zero elements (len(self.stack) == 0)

Step 6: Get stack size
- return the no of elements using len(self.stack)

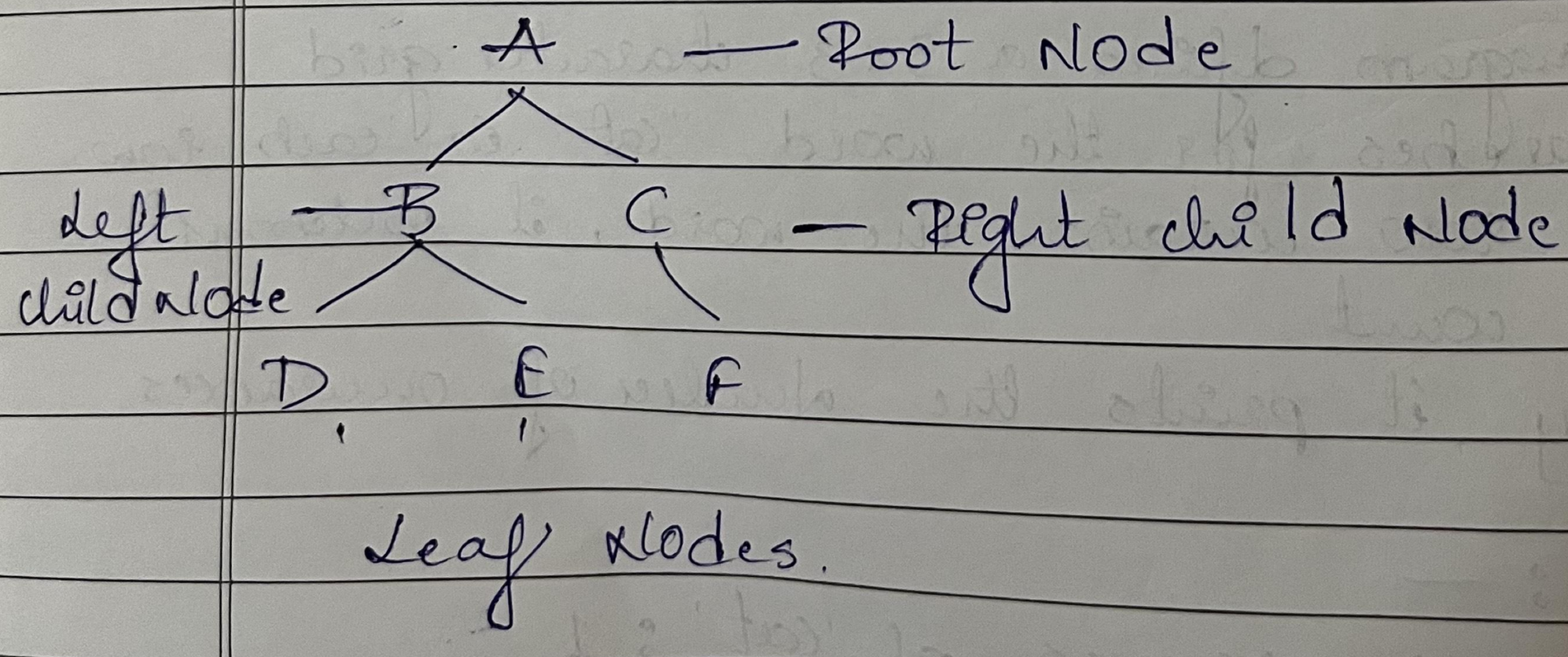
Date: / /

* Tree :— A tree is a hierarchical data structure that consists of nodes where each node has a value and may have child nodes. It is widely used in computer science for organizing data efficiently.

* Characteristics of a Tree :

1. Root Node : The topmost node of the tree
2. Parent child Relationship : Each node has a Parent and may have multiple children
3. Leaf Nodes : Nodes without children
4. Height of Tree : The longest path from the root of a leaf node.
5. Subtree : A small tree within a large tree.

Example of a tree :



Date: / /

* Mirror of a Tree: A Mirror is a tree where the left and right subtrees of every node are swapped the Mirror image of a tree is obtained by recursively swapping the left and right children of all nodes.

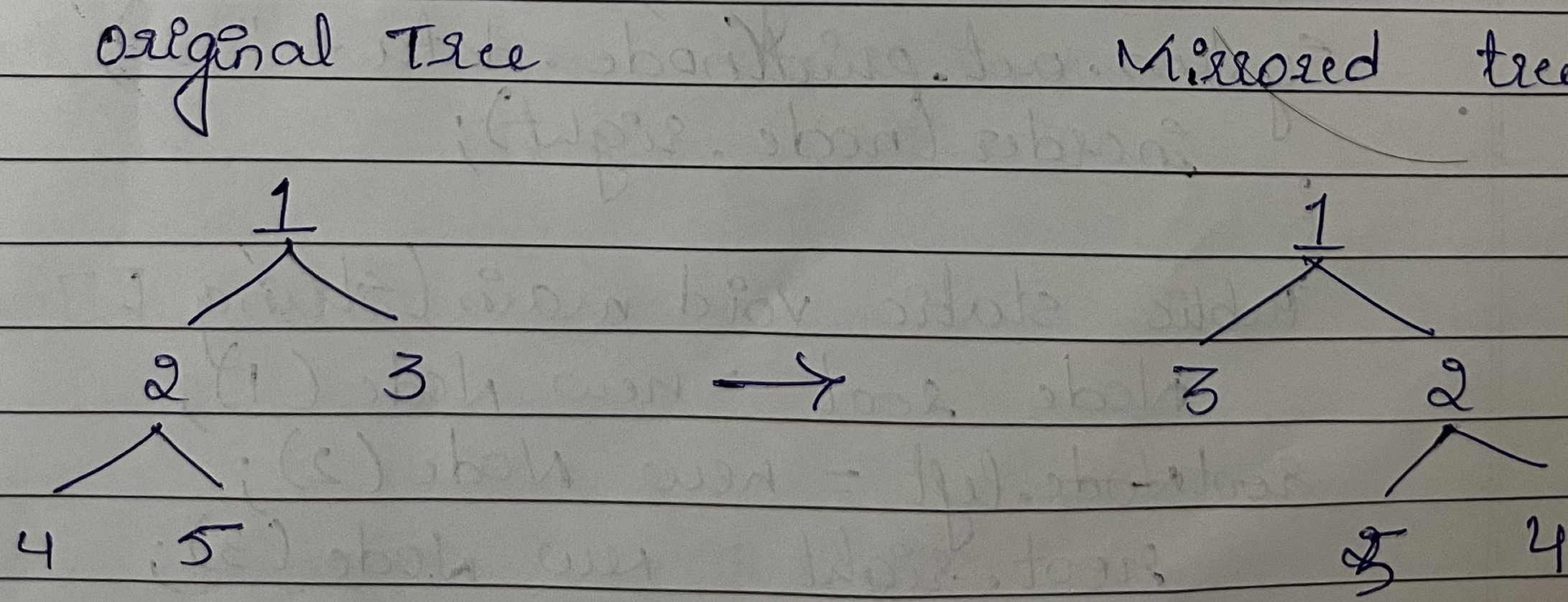
* characteristics of a Mirror Tree:

1. Left and Right subtrees are swapped
- Every node's left child becomes the right child and vice versa.

2. Recursive structure
- the Mirroring process is applied recursively to all subtrees.

3. Inorder Traversal of original and Mirrored Trees are Opposite.
- if we Perform an inorder traversal on the original and mirrored tree, the results will be reversed

* Example of a Mirror Tree:



Date: / /

- Here the original tree is converted to Mirror tree
- The left and right children of every node are swapped.

* Program.

```
class Node {  - represents a tree node
    int data;  the value of a node
Node left, right;  - Points to child nodes
Node (int data) { this.data = data; }
}

class MirrorTree {
Static void Main Mirror(Node node) {
    if (node == null) return;
    Node temp = node.left;
    node.left = node.right;
    node.right = temp;
    mirror (node.left);
    mirror (node.right);
}

static void inOrder (Node node) {
 if (node == null) return;  Inorder traversal
  inOrder (node.left);  left → Root → right Print tree
  System.out.print(node.data + " ");  elements in sorted order
    inOrder (node.right);
}

Public static void main (String [] args)  Main Method
    Node root = new Node (1);
  root.left = new Node (2);
    root.right = new Node (3);
```

{ swap operations

Date: / /

```
inOrder(root);      Print original tree
mirror(root);       convert to mirror image
System.out.println();
inOrder(root);      Print mirrored tree
}
}
```

Output :- * Given the Binary Tree values.
take 1, 2, 3.

```
    1
   / \
  2   3.
```

- inorder Traversal (Left → Root → right)
- Traverse left subtree → 2
  visit root → 1
  Traverse right subtree → 3

- output Before Mirroring :
  2, 1, 3.

* Mirrored Binary Tree after Mirroring

```
    1
   / \
  3   2
```

After swapping the left and right subtrees at each node.

- Inorder Traversal (Left → root → right
- Traverse left subtree → 3
- Visit root → 1
- Traverse right subtree → 2

- output after mirroring
  3, 1, 2

Final Program output :-
2 1 3
3 1 2

Date: / /

* Count of number of given string in 2D character Array. ?

→ Counting the number of times a given string appears in a 2D-character array by checking in all 8 Possible directions (horizontally, vertically, and diagonally).

① Horizontally - Left to right, right to left
② Vertically - top to bottom, bottom to top
③ Diagonally - four possible diagonal directions

For Example,
consider this 3×3 grid.

c a t
a t c
t c a.

- If we search for "cat", we can find it in different locations and directions.

like;

① (0,0) → Right (c → a → t).
② (0,0) → Diagonal Down (c → t → a).
③ (0,2) → Left (t → a → c).
④ (2,0) → Diagonal up (a → t → c)

Date: / /

* Program

```
Public class Simple WordSearch{
Public static void main (string[] args) —main method
    char [] [] grid = {
        {'c', 'a', 't' },     this 3×3 character grid represents
        {'a', 'c', 't'}      letters arranged in rows
        {'t', 'a', 't'},     the goal is to search for word 'cat' in these rows
    };
  string word = "cat"; it is looking for word cat
    int count = 0;
  for (int i=0; i<grid.length; i++){
 string rowstring = new string (grid [i]);   } it converts the row from a character array to string (c
    if (rowstring.contains(word)){
        count++;
    }
  }
system.out.println ("occurences of '"+ word+"': "+ count);
}                 it will finally display the word cat.
}
```

Here :

- this Program defines a 3×3 character grid
- It searches for the word "cat" in each row.
- if the row contains the word, it increments the count
- Finally, it prints the number of occurances.

output :—

Occurrences of 'cat' : 1

Girish Yandigeri USN: 1RV24MC037

Sorting on Array based on set bits

1) What is a set bit?

→ It is a bit that has a value of 1 in the binary representation of a number.

ex:→ Binary of 5 = 101 (2 set bits)

" " 7 = 111 (3 set bits)

2) Problem statement

"Given an array of integers, we need to sort them in descending order based on the number of set bits in their binary representation.

3) Example to illustrate

arr = [5, 3, 7]

number	binary representation	setbit count
5	101	2
3	11	2
7	111	3

sorting based on set bits

1. The number with most bits come first.
2. Same no. of set bits numbers retain original order

- 7 has the hgest no. of set bits, so 7 comes first

3) How to count set bits

1. basic method - normal count
2. Mathematical function - Brian kernighans Algo
3. Python provides built-in function = bin(n).count('1')

4) Real life applications

- Cryptography
- Hardware optimization
- Data compression
- Computer networks

5) Time complexity and space complexity

Time complexity

- counting set bits take O(logN) (because a number N has $\log_2 N$ bits)
- sorting taks O(N logN)
- Total: O(N logN)

Space complexity

- If sorting is done in place, requires O(1) space extra
- If creates a new list, takes O(N) extra space