

Recursion & Backtracking

- Karun Karthik

Contents →

- | | |
|---|----------------------|
| ① Power of two | ⑪ Subsets II |
| ② Power of three | ⑫ Combination sum II |
| ③ Power of four | ⑬ N-Queens II |
| ④ Subsets | |
| ⑤ Combination sum | |
| ⑥ Rat in a maze | |
| ⑦ N-Queens | |
| ⑧ Sudoku solver | |
| ⑨ Knight's tour problem | |
| ⑩ Letter combination of a phone number. | |

DI

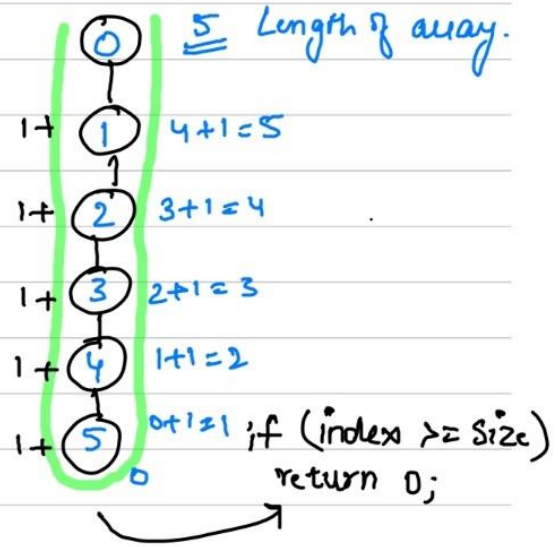
Recursion

⑩ Length of an array

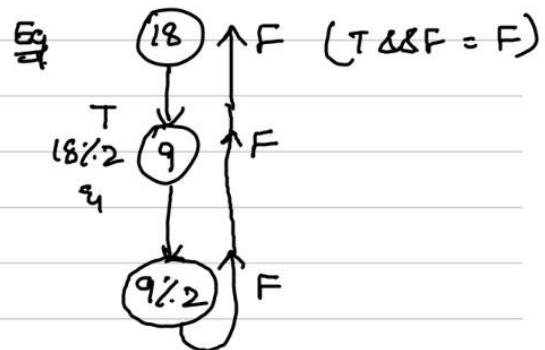
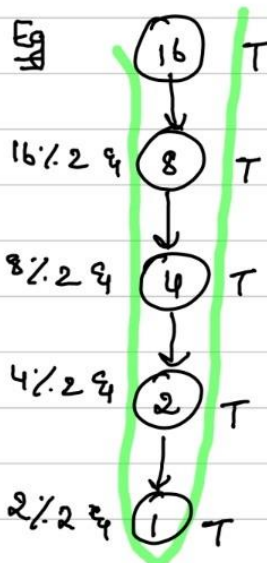
[20, 10, 40, 50, 30]
0 1 2 3 4

TC = $O(n)$

SC = $O(n)$.



⑪ Power of 2 $\rightarrow 2^x = 2^0 \cdot 2^1 \cdot 2^2 \dots 2^n$.



TC = $O(\log_2 n)$

Linked List

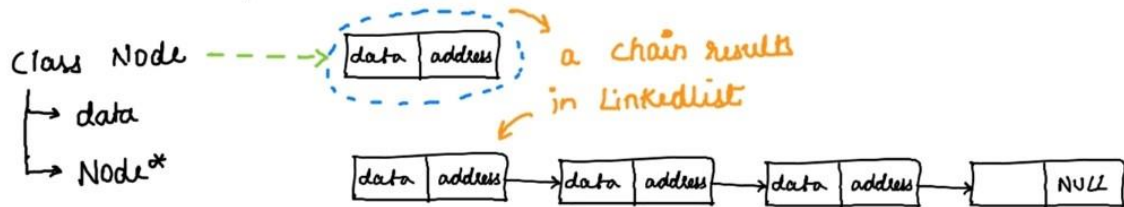
- Karun Karthik

Contents →

0. Introduction
1. Reverse a Linked List
2. Middle of Linked List
3. Delete node in a Linked List
4. Merge two sorted Lists
5. Add two numbers
6. Add two numbers II
7. Linked List Cycle
8. Linked List Cycle II
9. Remove Nth node from End of List
10. Palindrome Linked List
11. Remove duplicates from sorted List
12. Swapping nodes in Linked List
13. Odd Even Linked List
14. Swap Nodes in Pairs
15. Copy list with Random Pointer
16. Reverse Nodes in K-group
17. Design Linked List
18. Sort List

Linked List

linkedlist is linear data structure, which consists of a group of nodes in a sequence.



Advantages

1. Dynamic nature
2. Optimal insertion & deletion
3. Stacks and queues can be easily implemented
4. No memory wastage

Real life Applications

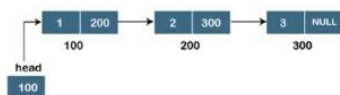
1. Previous & next page in browser
2. Image Viewer

Disadvantages

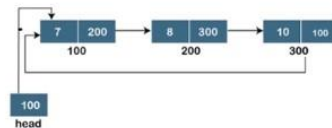
1. More memory usage due to address pointer.
2. Slow traversal compared to arrays.
3. No reverse traversal in singly linked list
4. No random access.

Typu

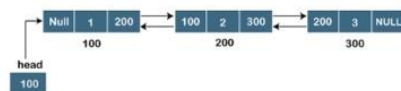
1. Singly linkedlist



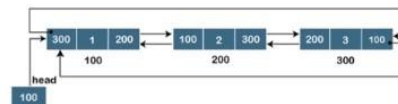
- ### 3. Circular Linked list



2. Doubly linkedlist



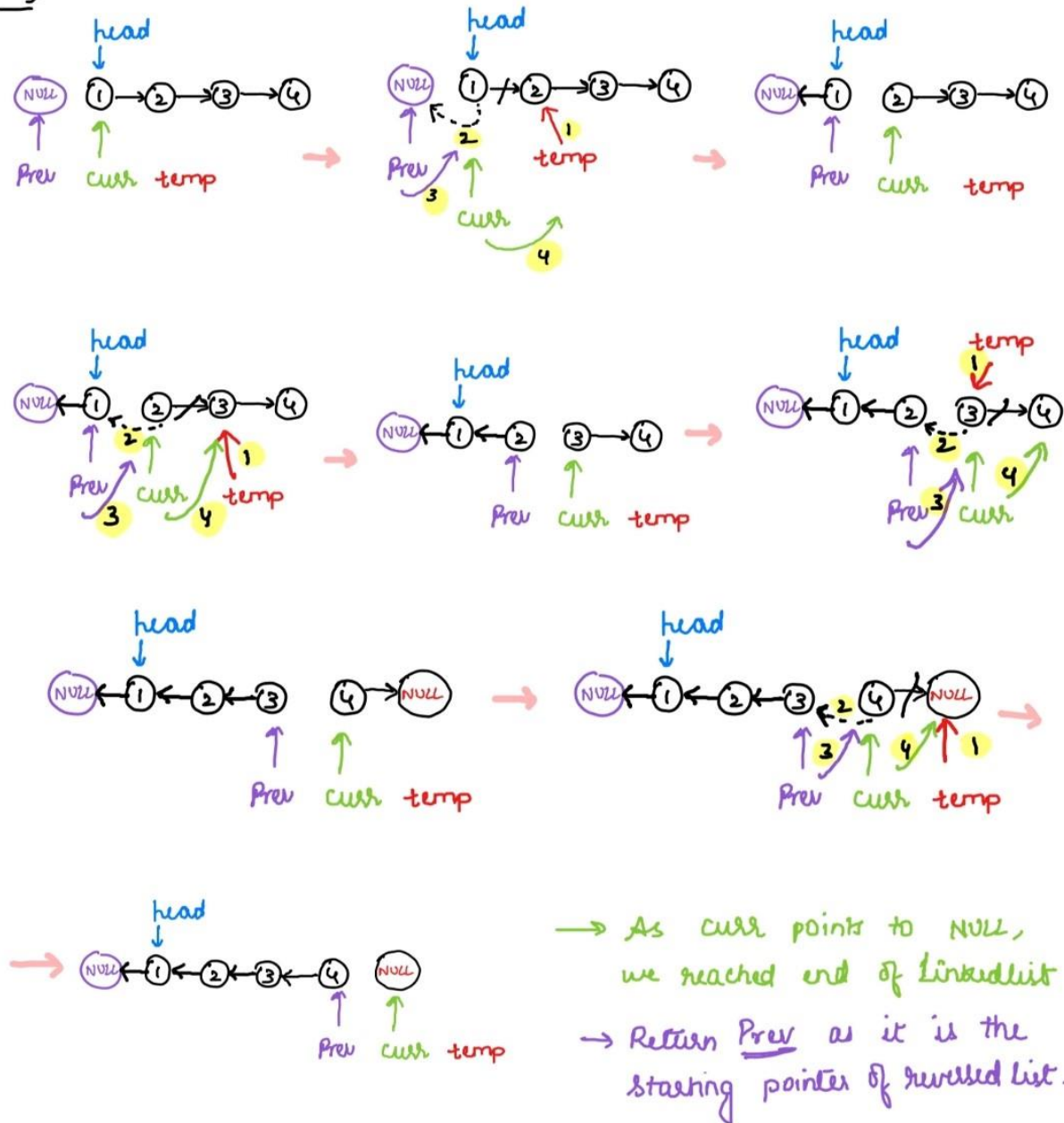
4. Doubly circular linkedlist



① Reverse a linkedlist → Given a linkedlist, return reversed list.

Eg $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \Rightarrow 4 \rightarrow 3 \rightarrow 2 \rightarrow 1$

Sol)



Trees - Part 1

- Karun Karthik

Contents

0. Introduction
1. Max depth of Binary tree
2. Max depth of N-ary tree
3. Preorder of binary tree
4. Preorder of N-ary tree
5. Postorder of binary tree
6. Postorder of N-ary tree
7. Inorder of Binary tree
8. Merge two binary trees
9. Sum of root to leaf paths
10. Uni-valued Binary tree
11. Leaf similar trees
12. Binary tree paths
13. Sum of Left leaves
14. Path sum
15. Left view of Binary tree
16. Right view of Binary tree
17. Same tree
18. Invert Binary tree
19. Symmetric tree
20. Cousins of Binary tree

Trees

Why trees?

Tree - collection of tree-nodes

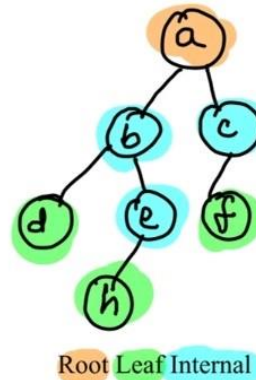
1. Hierarchy
2. Computer system.
(UNIX)

① Class Treenode

↳ data
↳ list <Treenode> children

② Binary Tree → atmost 2 children (0,1,2)

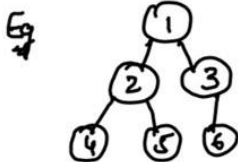
↳ data
↳ leftchild
↳ rightchild



③ Types →

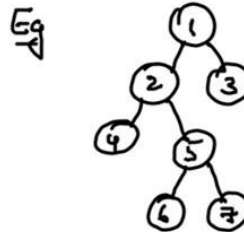
Ⓐ Complete Binary Tree

↳ all levels are completely filled except last one



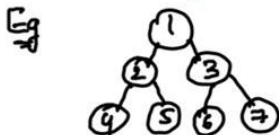
Ⓒ Full Binary tree

↳ if every node has 0 or 2 children



Ⓑ Perfect Binary Tree

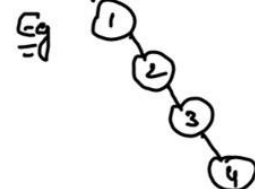
↳ every internal node has exactly 2 children



Ⓓ Skewed Binary Tree

(* used for finding complexity)

↳ all nodes have either one or no child.



Trees - Part 2

- Karun Karthik

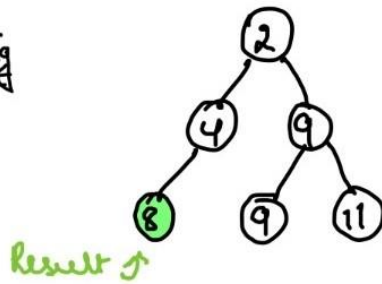
Contents

21. Print all nodes that do not have any siblings
22. All nodes at distance K in a Binary Tree
23. Lowest Common Ancestor
24. Level order traversal in Binary Tree
25. Level order traversal in N-ary Tree
26. Top view of Binary Tree
27. Bottom view of Binary Tree
28. Introduction to Binary Search Tree & Search in a BST
29. Insert into a BST
30. Range Sum of BST
31. Increasing order search tree
32. Two Sum IV
33. Delete Node in a BST
34. Inorder successor in BST
35. Validate BST
36. Lowest Common Ancestor of BST
37. Convert Sorted Array to BST
38. Construct BT from Preorder and Inorder traversal
39. Construct BT from Inorder and Postorder traversal
40. Construct BST from Preorder traversal

<https://www.linkedin.com/in/karun-karthik-5a0794187/>

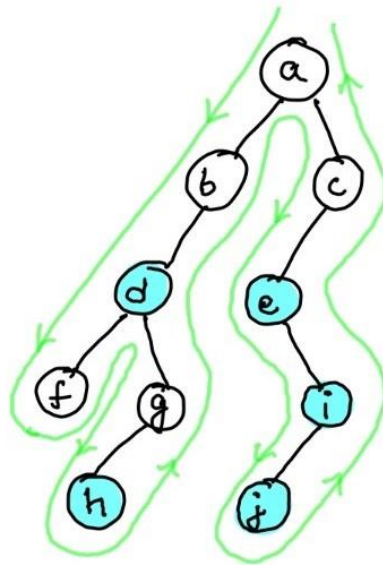
D7 (21) Print all nodes that don't have any siblings

Ans



Sibling \rightarrow same level, same parent

✓



Tc $\rightarrow O(n)$

Sc $\rightarrow O(n)$

At every node, check if

both branches exist \rightarrow then call both of them recursively

only left branch exist \rightarrow then call left branch recursively

only right branch exist \rightarrow then call right branch recursively



Recursion & Backtracking

- Karun Karthik

Contents →

- ① Power of two
- ② Power of three
- ③ Power of four
- ④ Subsets
- ⑤ Combination sum
- ⑥ Rat in a maze
- ⑦ N-Queens
- ⑧ Sudoku solver
- ⑨ Knight's tour problem
- ⑩ Letter combination of a phone number.
- ⑪ Subsets II
- ⑫ Combination sum II
- ⑬ N-Queens II

DI

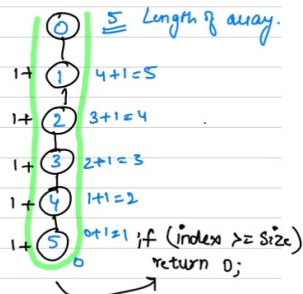
Recursion

⑩ Length of an array

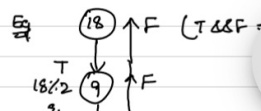
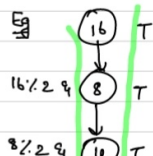
[20, 10, 40, 50, 30]
0 1 2 3 4

T.C = O(n)

S.C = O(n)



① Power of 2 → $2^k = 2^0 \cdot 2^1 \cdot 2^2 \dots 2^n$





Contents



Recursion & Back-tracking



Trees - 1



Trees - 2



Linked List



**Application link in the
comments section**