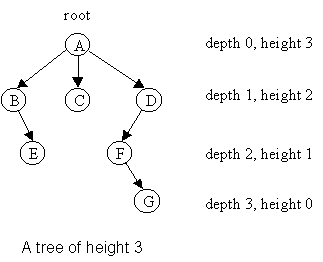


1. **Position of rightmost set bit:** log base 2 (AND operation of n and 2’s complement of n) +1 = (Log 2 (n & -n)) + 1
2. **Reverse a linked list**: Using 3 pointer
3. **Merged point of a linked list**: Iterate the longer list by the difference in lengths of LL and then iterate till the values become same. Better approach (avoids length): Traverse the LL simultaneously and if any of the two reaches the end then move it to another LL. If both become equal then that is the merged point. If both become null together then no intersection point.
4. **Remove nth position from the end**: Take a dummy pointer before head and initialize slow and fast as dummy. Iterate fast pointer n times then iterate fast and slow together. Delete slow next node. Initialize head as dummy next for corner case n=length of LL.
5. **Middle element**: Using slow and fast pointer
6. **Loop in a Linked list**: Using slow and fast pointer
7. **Add 2 numbers as linked list**: Add each node and store it in carry and sum. Store the sum in a new LL with head as 0 and traverse while both LL are not null and carry is 0. Delete the first node of new LL**.**
8. **Delete a given node in LL**: We have to copy the next node to this node and delete the next node.
9. **Palindrome or not**: Go to middle element (fast slow pointers). Reverse the next half of LL. Now move slow pointer (present at middle) and dummy starting from head simultaneously till slow is not null and compare if data is equal or not.
10. **Find the starting point of the loop:** Get the meeting point of fast and slow pointer. Traverse slow and a new pointer from start till they meet. That point is the starting point of loop. (See proof from copy)
11. **Flattening of a LL:** It is like merging of many LL. We can use recursion here as merging the last two LL till only one merged LL remains.
12. **Rotate a LL by k times:** Find the length and reduce k as k%l. Point the last node to head to make it a circular LL. Now traverse to n-kth node and point the next node as head and disconnect them.
13. **Reverse nodes in K groups:**
14. **Longest common Prefix:** Traverse the first index of all strings and make sure they are equal. Similarly, do it for other strings until a character is different or any of the string traversal is completed.
15. **Next greater element:** Take a stack and traverse from end of the list. If top element of stack is greater than the current element then adds that element in stack else remove the top element till element is greater. If length of stack is 0 return -1. Else if stack top element is greater than the current element than return that element.
16. **Next greater element (Circular):** For this we need to traverse the array twice and follow the procedure as mentioned above.
17. **Zigzag traversal of Binary Tree:** Take a queue and insert the root. Traverse the queue and add its children to the queue and its data to a list. Maintain a flag such that 0 means left to right and vice versa. Print the list element in forward or backward order as per the flag.
18. **Longest common Subsequence:** Use a 2d array of size length of string 1 X length of string 2. Now traverse the array and if the characters match, then take the value of that position as previous diagonal+1 else the maximum of the top and left values. The final answer is stored in the last position.
19. **1 missing no from AP series:** Find the difference between last and first element and divide by the size of array to get the common difference. Now use Binary search. Check left and right of middle element to get the missing element. If not found compare the value of middle element with the estimated value and modify low and high.
20. **Valid parenthesis:** Take a stack. Traverse the string and if it is a opening bracket then push it in stack else if it is a closing bracket check for the length of stack, if it is 0 return false. Now pop the top element from stack check if the closing bracket and opening brackets match or not, return the answer accordingly. At the end make sure that the length of stack is 0.
21. **Longest valid parenthesis:** Approach 1: Take a stack and insert -1. Now for open bracket push the index and for closed bracket pop the stack. If stack becomes empty push its index. Else the max length = maximum of current index – top of stack and max length so far. Approach 2**:** Count the number of open and closed brackets. If they are equal then, max length = maximum of 2\*open and max length so far. Else if closed is greater than open initialize both as 0. We again have to iterate the array from backwards in similar manner as the cases where open>closed is missed.
22. **Valid Parenthesis string:** Take 2 stacks. Add star indexes in one and open bracket indexes in another. If closed bracket is encountered pop from stack with open brackets indexes if it is empty pop from the other stack. If both are empty then is it is not a valid string. If bracket stack is not empty, pop it and star bracket together. Every index of star needs to be greater than that of open bracket as ‘\*(‘ is unacceptable. Then only its valid.
23. **Topological sort:** It is a sort such if there is a path from u to v then u appears before v. Perform a DFS traversal and insert the nodes whose execution is complete in a stack.
24. **The Celebrity Problem:** Only one or none celebrity is possible. Add every index in stack. Then pop any two elements until any one element remains. If A knows B, then A cannot be a celebrity else B cannot be a celebrity. Now traverse and check if the remaining element is celebrity or not.
25. **Sliding Window maximum:** Use a dequeue. Maintain a sequence of indexes in dequeue such that the elements are always decreasing. If an index is out of window remove it. If a current number is greater than last element in dequeue remove elements till it is inserted maintaining the descending order sequence. Always pop first element for answer.
26. **Minimum number of platforms required for a railway:** Sort the arrays of arrival and departure independently. Now traverse the array and take different pointer for both. If departure of j > arrival of i increase the platform. Else increase the pointer j until departure of j >= arrival of i.
27. **Boundary traversal:** First include the root, then the left part, bottom and right part. For left part, if left present then include it else if right present include it. Don’t add any leaf nodes here. For bottom part, in order traverse and include all the leaf nodes. For right part, if right present then include it else if left present include it in reverse order. Don’t include leaf nodes here.
28. **kth smallest element:** Approach 1: Sort and print kth position element. Approach 2: Bubble sort k times. Approach 3: Quick sort and if random pivot index at the end is k return that element. Approach 4: Use a min heap and perform delete min function k times.
29. **Kth smallest element in a BST:** Use in order traversal and return kth element.
30. **Maximum number of islands:**
31. **Set matrix zero:** Use the first row and column of matrix. If a row has 0 then initialize 0 to (i,0) and (0,j) position.
32. **Pascal Triangle:** (i,j) element is (i-1)C(j-1)
33. **Next permutation:** Traverse from end and find the find element that is not is increasing order from end (say j). Again, traverse from end and find a number just greater than that number and replace them. Now reverse the from j+1 to end.
34. **Kadane’s algorithm:** Findmaximum subarray. If sum so far + present number < the present number then, sum so far=present number. Check for maximum every time.
35. **0,1,2 sort:** Counting sort. Also using 3 pointers low, mid, high. Initially low, mid=0 and high=n-1. Traverse with mid such that low separates 0 and 1 and high separates 1 and 2 till mid<=high.
36. **Stock buy and sell:** Initially cp=max and profit=0. Now and profit = max (profit, cp-price)
37. **Rotate a matrix by 90 degrees:** Transpose and reverse every row.
38. **Merge overlapping intervals:** Sort them. Now, merge any two intervals if ending of one is >= starting of other.
39. **Duplicate in array of N+1 elements:** First approach: Using frequency array. Better approach- Find meeting point with fast and slow. Then find the starting point of loop by traversing fast from start and slow from meeting point in same pace.
40. **Find the Missing and Repeating Number:** First approach: Use frequency array. Second approach- Summation of numbers – Summation of (1, N) is repeating no – missing no and Summation of numbers square – Summation of (1, N) squares is repeating no square – missing no square. Now we can find repeating no + missing no and those numbers respectively. Third approach- Find xor of all numbers given and numbers from (1, N). It will give us repeating no xor missing no and we will find the rightmost set bit of that no. Now I will take two arrays and keep the numbers(given) containing the rightmost bit at same position in one array and rest in the other array. Same with the numbers (1,N). Now I will find xor the arrays. One containing the rightmost bit will given us the missing no and the other will give us the repeating no.
41. **Inversions of array**: While merging in merge sort if any number in right array is smaller than a number In left array, all numbers in the array right of that number will be counted as inversions.
42. **Search a 2D array**: Given rows and columns are sorted. Case 1- If the column last element is always less than the next column first element. Use Binary search and treat 2D array as 1D array. Case 2- The condition is not mentioned. Start traversal from top right corner. If required element is less then move left else move down.
43. **Power (x,n)**: If n is even x=x\*x else answer=answer\*x. If n is negative, convert it to positive, find x^n and return 1/answer.
44. **Majority element (>n/2):** Traverse the array and if count is 0 initialize answer as current element. Further if current element is equal to answer increase count else decrease it.
45. **Majority element (>n/3):** Almost same as above. Instead of one answer take two answers and do the same process. Then we have to again traverse the array and find the frequency of both the numbers. If they are greater than floor of n/3 then print any of them. Note maximum only two answer is possible.
46. **Grid Unique Path:** If matrix size is m X n then unique path is (m+n-2) C (n-1) or (m+n-2) C (m-1). Also, recursion and dynamic approaches possible.
47. **NCR:** Initialize answer as 1 and traverse from 1 to r, answer\*= (n-r+i) / i. e.g., 10C3 = (10\*9\*8) / (3\*2\*1)
48. **Reverse pairs:**
49. **2 sum problem:** Use dictionary to store numbers with index and if target-number in dictionary or not.
50. **4 sum problem:** Use 2 loops for first two numbers. Then use low and high pointers for the other two. If their sum is equal to target then add the quadruplet to answer, else increase low or decrease high accordingly. Remember to avoid duplicates for all pointers.
51. **Longest consecutive subsequence:** Store every element in a set. Now traverse the array and check if that number-1 in set or not. If not then from there check for every consecutive number and find the longest.
52. **Largest subarray with 0 sum:** Traverse the array and calculate the prefix sum. If prefix sum is not in the map add it with its position else the difference between positions is a length of subarray with sum 0. Check for max among them. Don’t forget to check for if prefix=0 condition.
53. **Count subarrays with a given xor:** Let prefix xor till current element is px. Then px = Y ^ target where Y is prefix xor of some elements. Also, px ^ target = Y (Taking target xor on both sides). So, we will save the frequency of prefix xor in dictionary and if px ^ target is present in dictionary we will add its frequency to answer. Don’t forget to add 0 initially to dictionary.
54. **Longest Substring without repeating characters:** Take a starting pointer as l. Now traverse the string and keep on storing the character along with its index in a dictionary until they are not there. If any character is already stored remove all the characters from starting pointer till that index. At each step check if max length is less than i-l+1 or not.
55. **DFS traversal of tree:** In-order, pre-order and post-order traversal using recursion.
56. **BFS traversal of tree:** Use Queue.Push the root to the queue. While length of queue is not 0 pop an element from queue, insert its children and print its data.
57. **Maximum depth/height:** The depth of a node is the number of edges present in path from the root node of a tree to that node. The height of a node is the number of edges present in the longest path connecting that node to a leaf node. 1 + max (maximum depth (left subtree), maximum depth (right subtree)). Height of a tree= maximum depth.



1. **Balanced Binary Tree:** Thisfunction will return -1 if it is not BT is not a balanced BT else height of tree.If check (left subtree) or check (right subtree) =-1 return -1 else if abs (check (left subtree) - check (right subtree)) >1 then return -1 else 1+ check (left subtree) + check (right subtree)
2. **Diameter of a Binary tree:** Diameter is longest path between any two nodes. We can use the height of tree function just add max diameter=max (max diameter, left subtree height + right subtree height)
3. **Check if two trees are identical:** If both are none return True. If one of them is none return false. If their data is same, call the function recursively for subtrees and if they are true return true else return false.
4. **LCA in a Binary tree:** Lowest common ancestor is the common ancestor that is closest. Traverse the array and if the data is equal to either of data required then return that node else search for subarrays. If one of the subarrays doesn’t return None then return that value. It both of the subarray doesn’t return None then return that node itself.
5. **Evaluate infix expression:** Step 1:Convert infix to postfix.Traverse the string and add operands in answer and operators in stack. If operator is of higher precedence than add it in stack else pop everything and add to answer till lower precedence operator or stack is empty and add that operator to stack. Step 2: Postfix evaluation. Add operands to stack. If an operator is encountered pop the top two operands, perform that operation and push the answer back to the stack.
6. **Towers of Hanoi**: Recursively transfer n-1 discs to middle from source. Transfer nth disc from source to destination. At last, recursively transfer n-1 discs to destination from middle.
7. **Implement Queue using Stack:** Take two stacks say A and B. Always push elements in A in enqueue operation. In case of dequeue, if B is not empty pop B, else for all elements pop A and push B till A is empty and then pop B.
8. **Implement Stack using Queue**: Take two queues A and B. Keep one of the queues always empty. For push operation, if A is empty then add in B else add in A. For pop operation, add all elements of non-empty queue to empty queue except the last element and return the last element.
9. **Euclid GCD:** If n % m=0 return m else return GCD (m, n % m )
10. **Heapify:** It is used to balance the heap after certain changes. It traverses down and finds the right position of input index say i. First it finds the greater value child then it checks if it is greater than the ith value or not. If yes then that child node moves up and traversing continues else stops.
11. **Dijkstra’s:** Take two arrays for distance and previous node. Initially all distance is infinite except source which is 0 and previous node is -1. Take a minheap and add all the vertices with distance as key. While minheap is not 0 perform delete min and perform edge relaxation for all the adjacent edges. Edge relaxation is a vertex v is if distance[v] > distance[u] + cost [u, v] then modify distance[v] as distance[u] + cost [u, v], previous node as u and modify key in minheap. O ((n + e) \*log n). Fails for negative edges and negative cycle (cycle with a negative sum of edges weight).
12. **Bellman ford:** Run a loop for n-1 times and perform edge relaxation to all the edges (edges given not vertices). Now perform edge relaxation again (nth time) and if any edge pair shows decrease in distance, print negative cycle present else return distance. O(n\*e)
13. **Prims:** Take three arrays for distance, previous node and color. Initially all distance is infinite except source which is 0 and previous node is -1. Take a minheap and add all the vertices with distance as key. While minheap is not 0 perform delete min and perform edge relaxation for all the adjacent edges. Edge relaxation is a vertex v is if distance[v] > cost [u, v] and is black then modify distance[v] as cost [u, v], previous node as u and modify key in minheap.
14. **Kruskal’s:** Build a minheap for the edge costs. Now perform delete min and check if it forms a cycle or not. If not then add it into MST.
15. **Huffman coding:** Delete two min nodes. Add them to form a new node with weight as sum of both the weights. Add them to set again. Repeat this until only one remains.
16. **Job scheduling:** Profit and deadline is given. Sort profit array. Now add maximum profit to any available position between deadline day to day 1 (traverse from D to 1).
17. **Matrix chain multiplication:**
18. **Fractional knapsack:** Make an array and calculate profit/weight ratio. Sort the array and calculate profit by including maximum profit/weight ratio first.
19. **0/1 knapsack:** Run inner loop (i) from 0 to size to knapsack given. Run the outer loop (j) from 0 till the number of items given. Now if the weight of item < j then (i, j) = (i-1, j) value else (i, j) = maximum of (i-1,j) and (i-1, j-weight[i]) + profit[i].
20. **All pair shortest path (Floyd Warshal):** In one loop (i) take vertices one by one. Now traverse the rest of graph in two loops (j,k) except paths of vertex i. If the distance of path (j,k) > distance of path (j,k) via I then replace it.
21. **DFS graph:** Take a visited array and initialize every vertex as 0. Now for all vertices if visited value is 0 calls DFS function. In DFS function change visited as 1. Traverse all adjacent vertices of the vertex if visited is 0 recursively call DFS function for that vertex.
22. **BFS graph:** Take a visited array and initialize every vertex as 0. Now for all vertices if visited value is 0 calls BFS function. In BFS function change visited as 1. Insert the vertex in queue and run a loop till queue is empty. Traverse all adjacent vertices of the vertex we get by dequeue of queue, if visited is 0, change visited as 1 and add it in queue.

**Definitions:**

**Full BT:** Two or Zero children.

**Complete BT:** All previous levels are filled. And all nodes in current level are on left.

**Perfect BT:** All levels are complete.

**Balanced BT:** Height = O (log n). example: AVL trees.

**Degenerate BT:** Linked list

**Subsequence:** All possible combinations such that the index order of those elements is always increasing. 2^N-1

**Subarray:** Contiguous subsequence. N\*(N+1)/2

**Subset:** Subsequence plus empty sequence plus no order maintenance.

**Project**

I have done various projects on web development. My most recent project is a website on daily journal. It’s a website where we can write blogs or daily journal. It is a project on Backend web development and the tools I used in the same are HTML, CSS, EJS, JavaScript, NodeJS, ExpressJS, MongoDB and Mongoose.

**Situation (Why?):** It was given to me as an assignment on a course of web development by Angela Yu.

**Task (Plan):** First I planned to make a simple site with a compose, about, contact and display page with title and content.

**Action (Execution):** I completed that successfully.Then I observed that the difference in sizes of content making it look bad. Then I decided to display only the first 100 words of each content and if the content is greater than the 100 words a Read more option will be displayed that will redirect to a new page with the content and topic of that particular post. The url extension used is the topic name. But cannot have special characters in my url so I used lodash to eliminate the special characters.

**Result:** I worked fine.

I have also done a project on meme website which allows any user to post and enjoy memes. I used the same technologies as used in the daily journal website i.e. HTML, CSS, EJS, JavaScript, NodeJS, ExpressJS, MongoDB and Mongoose. Here only the top 100 memes will be displayed along with the name who posted it and a caption with the most recent post at the top.

Apart from this I have done a project on Simon game which is a memory game to remember a sequence. It’s a project on front end web development where I used HTML, CSS, JavaScript. There are four boxes and I allotted a color, sound and number to each box [1,2,3,4]. I used a random number and used to add it in an array. Every time when a correct sequence is entered the new random number is added and displayed else the game is over and is restarted.