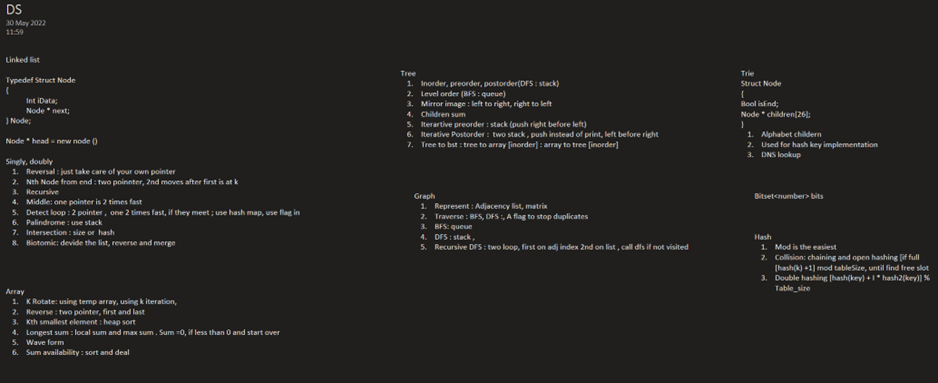


Multiset(balanced BST) : used to find min and max in sliding windows

[Sliding Window Maximum (Maximum of all subarrays of size K) - GeeksforGeeks](https://www.geeksforgeeks.org/sliding-window-maximum-maximum-of-all-subarrays-of-size-k/)(check heap soln), store index with element, while getting max check for index , if less pop



Graph:

Both adj list and Adjacency matrix can used for BFS and DFS

I have gathered around 20 of these coding problem patterns that I believe can help anyone learn these beautiful algorithmic techniques and make a real difference in the coding interviews.

1) Sliding Window

2) Islands (Matrix Traversal)

3) Two Pointers

4) Fast & Slow Pointers

5) Merge Intervals

6) Cyclic Sort

7) In-place Reversal of a LinkedList

8) Tree Breadth-First Search

9) Tree Depth First Search

10) Two Heaps

11) Subsets

12) Modified Binary Search

13) Bitwise XOR

14) Top ‘K’ Elements

15) K-way Merge

16) Topological Sort

17) 0/1 Knapsack

18) Fibonacci Numbers

19) Palindromic Subsequence

20) Longest Common Substring

------

0. Count the occurrence of a number X in a sorted array

find the first occurance using binary search, then move left and right

1. K largest elements from a big file or array

heap sort , devide into small and heap sort and then merge

2. Reverse a Linked List in groups of given size

have prevStart, prevEnd, curSTart, curEnd, NextStart, grpSize and use normal reversal

3. Implement a stack with push(), pop() and min() in O(1) time

Have a variable with min , 2(curMin)-prevMin [Design a stack that supports getMin() in O(1) time and O(1) extra space - GeeksforGeeks](https://www.geeksforgeeks.org/design-a-stack-that-supports-getmin-in-o1-time-and-o1-extra-space/)

4. Add two numbers represented by linked lists

normal addition with carry for DLL, SLL use a stack

5. Convert a Binary tree to DLL

using BFS(level order traversal), for recursive have HEAD as global variable

6. Stock span problem //consecutive day for which the stock price is highest========

bruteforce (n^2), using stack to save index and pop if price is less (n) https://www.geeksforgeeks.org/the-stock-span-problem/,

7. Next larger element ====

bruteforce, using stack to save the largest num,

8. Edit distance //https://www.geeksforgeeks.org/edit-distance-dp-5/

DP: if letter are same m[i][j]= m[i-1][j-1], else 1+ min(m[i-1][j], m[i][j-1],m[i-1][j-1])

9. Maximum of all subarrays of size k //https://www.geeksforgeeks.org/sliding-window-maximum-maximum-of-all-subarrays-of-size-k/

brute force, using heap sort for k element

10. Pythagorean Triplet

brute force, an sorted square array -> from end take the number -> find the sum

11. Print a Binary Tree in Vertical Order

preorder traversal to calculate vertical order and push to map and then print from map

12. Level order traversal

use queue

13. Smallest window in a string containing all the characters of another string // https://www.geeksforgeeks.org/find-the-smallest-window-in-a-string-containing-all-characters-of-another-string/

14. Find the number of islands(covid spread like)

15. Detect and Remove Loop in a Linked List

fast and slow ptr, count nodes in loop, two pointer (k apart) to find start of loop

16. Check if a binary tree is BST or not

check the complete min max (by going extreme left/right)

17. Boolean Parenthesization(xxxxxx)

18. Arrange given numbers to form the biggest number https://www.geeksforgeeks.org/given-an-array-of-numbers-arrange-the-numbers-to-form-the-biggest-number/#:~:text=Given%20an%20array%20of%20numbers,998764543431%20gives%20the%20largest%20value.

19. Implement LRU Cache

hash + DLL

20. Maximum difference between node and its ancestor in Binary Tree //recursion

21. Covid spread : https://www.geeksforgeeks.org/maximum-time-required-for-all-patients-to-get-infected/

Topic Wise

Arrays :

—-----------------------------------------

1. Subarray with given sum

brute force, sliding window concept, **DP**

subarraysum(int a[], int n, int sum)

**{**

If sum == 0 return 1;

If n ==0 return 0;

Return subarraysum(a, n-1, sum) !|subaraysum(a, n-1, sum - a[n-1]));

}

2. Count the triplets

one pnc formula,sort and try

3. Kadane’s Algorithm

local and global max

curr\_max = max(a[i], curr\_max+a[i]);

max\_so\_far = max(max\_so\_far, curr\_max);

4. Missing number in array

using sum

binary search variation

5. Merge two sorted arrays

merge sort

6. Rearrange array alternatively

two pointer and new array

7. Number of pairs

brute force,

8. Inversion of Array

* Do sel/bubble/insertion sort and count all swaps
* Use heap/quick sort and count all swaps

sum of smaller number on left for all index, use sorting also

9. Sort an array of 0s, 1s and 2s

counting sort

10. Equilibrium point

* Left sum, right sum, pivot(start with arr[0] as pivot)

two pointer

11. Leaders in an array

scan from right , keep the max/leader till now

12. Minimum Platforms

13. Reverse array in groups

normal

14. K’th smallest element

heap sort

15. Trapping Rain Water

* (min(max on left, max on right) - ele) : do sum for all ele //find prev and next max

16. Pythagorean Triplet

sorted square array, one loop for sum and second loop with two pointer to find sum

17. Chocolate Distribution Problem

* Sort , run a loop for len and fin the arr with min diff

18. Stock buy and sell

* Buy on day 0, sell on day before where increasing order break,buy on reduced and repeat

19. Element with left side smaller and right side greater

* Create two array lextmax and right min and use them

20. Convert array into Zig-Zag fashion

sort and then place one large one small

21. Last Index of 1

from right

22. Spirally traversing a matrix

seperate func to print a row (once j increasing , nxt decreasing base on i odd or even)

23. Largest Number formed from an Array

radix style sort and the concatenate

* String concatenation to compare two number

String :

—------------------------------

1. Reverse words in a given string

double reversal, take word and add at start

2. Permutations of a given string

3. Longest Palindrome in a String : DP

4. Recursively remove all adjacent duplicates

5. Check if string is rotated by two places

str1[i] == str2[i+2 %n]

str1[i+2%n] == str2[i]

6. Roman Number to Integer

7. Anagram

Sort or count freq

8. Remove Duplicates

9. Form a Palindrome

10. Longest Distinct Characters in the string

11. Implement Atoi

12. Implement strstr

13. Longest Common Prefix

Sort and find between first and last

Linked List :

—----------------------------

1. Finding middle element in a linked list

fast and slow pointer

2. Reverse a linked list

normal

3. Rotate a Linked List

4. Reverse a Linked List in groups of given size

normal

5. Intersection point in Y shaped linked lists

length and cal

6. Detect Loop in linked list

fast and slow

7. Remove loop in Linked List

count loop size , go to end and null it

8. n’th node from end of linked list

fast and slow pointer

9. Flattening a Linked List : mergesort

10. Merge two sorted linked lists

mergesort

11. Intersection point of two Linked Lists

size

12. Pairwise swap of a linked list

13. Add two numbers represented by linked lists : use stack for SLL

14. Check if Linked List is Palindrome

stack

15. Implement Queue using Linked List

normal

16. Implement Stack using Linked List

normal

17. Given a linked list of 0s, 1s and 2s, sort it

counting sort

18. Delete without head pointer

copy the data of next node and delete nxt node

Stack and Queue :

—----------------------------------------

1. Parenthesis Checker

2. Next larger element

3. Queue using two Stacks

4. Stack using two queues

5. Get minimum element from stack : use two stack, second for min (or 2x-min push for <min, while pop min =2min-y for if y< min

6. LRU Cache

7. Circular tour

8. First non-repeating character in a stream: use a vector to push non rep char

9. Rotten Oranges : covid spread

10. Maximum of all subarrays of size k : heap sort of k, use set or multiset which is a balanced bst

Tree :

1. Print Left View of Binary Tree : level order traversal with two q (one q per level), print the first item of each q

2. Check for BST : inorder traversal should be sorted, pass the range to function and check for max and min

3. Print Bottom View of Binary Tree : create horizontal depth map<int, vector> , print the last ele

4. Print a Binary Tree in Vertical Order : horizontal map

5. Level order traversal in spiral form :

* Create level order map and print in alternate order
* Use two stack, (odd even : right left) to push

6. Connect Nodes at Same Level: use two q or 1 q(looping is with size)

7. Lowest Common Ancestor in a BST====find the first node between n1 and n2

8. Convert a given Binary Tree to Doubly Linked List: iterative traversal

9. Write Code to Determine if Two Trees are Identical or Not: any traversal

10. Given a binary tree, check whether it is a mirror of itself: same and identical tree but pass alternate left and right

11. Height of Binary Tree

12. Maximum Path Sum====

13. Diameter of a Binary Tree===

14. Number of leaf nodes

15. Check if given Binary Tree is Height Balanced or Not

16. Serialize and Deserialize a Binary Tree: use -1 as marker in preorder

Iterative preorder : With one stack , push right before left

iterative inorder : one stack , push left till end, print top , if right child push it along with all left

iterative postorder :

1. Push root to first stack.

2. Loop while first stack is not empty

2.1 Pop a node from first stack and push it to second stack

2.2 Push left and right children of the popped node to first stack

3. Print contents of second stack

Lowest common ancestor: write a search func,

if search(node->left, p) && search(node->left, q) {findCommonAncestor(node->left, p, q)}

if search(node->right, p) && search(node->right, q) {findCommonAncestor(node->right, p, q)}

else return Node;

Heap :

1. Find median in a stream

2. Heap Sort

3. Operations on Binary Min Heap

4. Rearrange characters

5. Kth largest element in a stream

6. Merge K sorted linked lists

7. Kth largest element in a stream

Recursion :

1. Flood fill Algorithm

2. Number of paths

3. Combination Sum – Part 2

4. Special Keyboard

5. Josephus problem

Hashing :

Hashing : https://www.geeksforgeeks.org/hashing-data-structure/?ref=shm

% the easiest hash function

double hashing

Open addressing :linear probing(hash(x) + 1) % S, quadratic probing(hash(x) + 1\*1) % S, double hashing(hash(x) + i\*hash2(x))

chaining to avoid collision

rehashing : resizing of hash if load factor is more

Solution patern

----------------

Mostly searching through Hash map

create multiple hash

create reverse hash

Find all pairs (a, b) in an array such that a % b = k

Convert an array to reduced form | Set 1 (Simple and Hashing)

1. Relative Sorting

2. Sorting Elements of an Array by Frequency

3. Largest subarray with 0/K sum : https://www.geeksforgeeks.org/number-subarrays-sum-exactly-equal-k/

4. Common elements

5. Find all four sum numbers ,store pair sum for all pair in hash map<sum, vector<pair<i,j>>>

6. Swapping pairs make sum equal

7. Count distinct elements in every window

8. Array Pair Sum Divisibility Problem : hash of remainder freq---- similar prob with hash of remainder and index(to find subarray)

9. Longest consecutive subsequence

10. Array Subset of another array

11. Find all pairs with a given sum

12. Find first repeated character

13. Zero Sum Subarrays

14. Minimum indexed character

15. Check if two arrays are equal or not

16. Uncommon characters

17. Smallest window in a string containing all the characters of another string : sliding window----

18. First element to occur k times

19. Check if frequencies can be equal

20. vertical traversal and sum in a binary tree

Graph :

adjacency matrix and adjacency list

1. Depth First Traversal : recursive, stack, have a visited array[n]

2. Breadth First Traversal : iterative and q

3. Detect cycle in undirected graph : dfs , if found visited during DFS and not parent

4. Detect cycle in a directed graph : dfs , with two visited/ret bool vector

5. Topological sort : modified dfs with stack push at end

* Order of algo
  + MST: kushals, prims
  + Min dist from src to all vertex: dijkstras, bellman ford
  + Min dist between all pair: floyd warshal
  + Kushal: MST(min spanning tree), sort edge, pick small , if not form a cycle include
    - Represent graph as list<pair<int, pair<int,int>>>
    - [Kruskal's Algorithm - javatpoint](https://www.javatpoint.com/kruskal-algorithm) , to find loop , check parent of parent
  + Prims:(adj matrix)MST, three array is In MST[v]=bool, dist[v] = max, parent[v] start with any vertex(ex 0),include it, dist[0] = 0 , outer loop : 0 to v, fin min dist[], include, inner loop: 0-> v update dist for all unincluded vertex
  + Dijkstras(a[][], src) : min path to all(greedy), same as prims , include min dist V if not already, update all vertex distance
  + Bellman ford: outerloop of v inner of e ,
  + Floyd warshal: min dist between all pair: 3 loop (0 to v) dist[i][j] = disk[i][k] + disk[k][j] (k outmost loop)

Graph

==========

Depth first traversal

int visited[v] = {0};

int Graph[][], int VCount

DFS()

{

//iterate over row

for(int i =0; i< VCount; i++)

{

if(!Visited[v])

DFSUtil(V);

}

}

DFSUtil(int V)

{

Visited{v] = 1;

cout << v;

//iterate over col

for(j = 0; j < vCount; j++)

{

if(Graph[V][j] != 1

continue;

if(!Visited[j])

DFSUtil(V);

}

}

DFSUtilIterative(int V)

{

stack<int> st;

st.push(V);

While(!st.empty())

{

int curV = st.pop();

cout << curV;

Visited[curV] = 1;

//iterate over col

for(j = 0; j < vCount; j++)

{

if(Graph[curV][j] == 1)

{

if(!Visisted[j])

st.push(j);

}

}

}

}

--------------------------------------

BFS

int visited[v] = {0};

int Graph[][], int Vcount

BFS()

{

//iterate over row

for(int i =0; i< VCount; i++)

{

if(!Visited[v])

BFSUtil(V);

}

}

BFSUtil(int V)

{

Queue<int> q;

q.insert(V);

While(!Q.empty())

{

int curV = q.remove();

cout << curV;

Visited[curV] = 1;

//iterate over col

for(j = 0; j < vCount; j++)

{

if(Graph[curV][j] == 1)

{

if(!Visisted[j])

q.insert(j);

}

}

}

}

------------------------------------

Topological sort //modified DFS , push to stack at the end

TS()

{

//iterate over row

for(int i =0; i< VCount; i++)

{

if(!Visited[v])

TSUtil(V);

}

}

TSUtil(int V)

{

Visited{v] = 1;

//iterate over col

for(j = 0; j < vCount; j++)

{

if(Graph[V][j] != 1

continue;

if(!Visited[j])

TSUtil(V);

}

st.push(v);

}

-------------------------

Cycle in UnDirected Graph // If you find any visited node in DFSUtil, check if its the parent, if not then there is a cycle

DFS()

{

//iterate over row

for(int i =0; i< VCount; i++)

{

if(!Visited[v])

if(IsCyclic(V, -1))

return true;

}

}

bool IsCyclic(int V, int parent)

{

Visited{v] = 1;

//cout << v;

//iterate over col

for(j = 0; j < vCount; j++)

{

if(Graph[V][j] != 1)

continue;

if(!Visited[j])

if(IsCyclic(j, v))

return true

else

if(j != parent)

return true

}

return false;

}

-----------

Cycle in diected graph //modified DFS, here parent is not just prev node but a whole bool [], cycle if there is a edge from node to its ancestor

DFS()

{

//iterate over row

for(int i =0; i< VCount; i++)

{

int parent[] = {0};

if(!Visited[v])

if(IsCyclic(V, parent))

return true;

}

}

bool IsCyclic(int V, int parent[])

{

Visited[v] = 1;

Parent[V] = 1

//cout << v;

//iterate over col

for(j = 0; j < vCount; j++)

{

if(Graph[V][j] != 1)

continue;

if(!Visited[j])

if(IsCyclic(j, v))

return true

else

if(parent[j] == 1)

return true

}

Parent[V] = 0

return false;

}

------------------

Graphs algorithms

MST : min spanning tree , kushals and prims

Min dist from src to all vertex: djikstras, bellman ford

Min dist between all pair: floyd warshal

Kushals: sort the edges , consider edges 1 by 1 and include if not forming cycle (super parent of both are not same)

q<pair<int, int>, int> : sort

msp<pair<int, int>, int>

int parent[] = // i for i

edgeInMsp = 0

int findParent(int v)

{

while(parent[v] != v)

{

v = parent[v];

}

return v

}

Krushals

{

for (edge in q)

{

if(findParent(q.first.first) ! = findParent(q.first.second)

{

msp.push(edge);

parent[q.first.second] = q.first.first; // to decide on who is parent use a rank array

edgeInMsp++;

if(edgeInMsp == v-1)

break

}

}

}

Prims(int src)

------

dist[] = {INT\_MAX to all}

InMST[] = {0}

dist[src = 0] = 0;

for(k: 0 to v-1)

i = find the min in dist, not already in MST

InMST[i] = 1;

for(j: 0 to v-1)

if(InMST[j] == false && dist[j] > dist[i] + g[i][j])

dist[j] = dist[i] + g[i][j]

dijkstras

-----------

dist[] = {INT\_MAX to all}

InMST[] = {0}

dist[src = 0] = 0;

for(k: 0 to v-1)

i = find the min in dist, not already in MST

InMST[i] = 1;

for(j: 0 to v-1)

if(InMST[j] == false && dist[j] > dist[i] + g[i][j])

dist[j] = dist[i] + g[i][j]

Bellman Ford

-------------

dist[src] = 0

for(v : vertex)

for(E : edge)

if(dist[e.sec] > dist[e.first] + e.weight)

dist[e.sec] = dist[e.first] + e.weight

Floyds

-------

3 loop

for(k: 0 to V-1)

for(i: 0 to v-1)

for(j: 0 to v-1)

if(graph[i][j] > graph[i][k] + graphk][j])

graph[i][j] = graph[i][k] + graph[k][j]

6. Find the number of islands : similar to dfs + covid spread

7. Implementing Dijkstra: same as prims , include min dist V if not already, update all vertex distance

8. Minimum Swaps :

9. Strongly Connected Components : topological sort on normal and then DFS on transpose graph starting with stack pop

10. Shortest Source to Destination Path : Dijkstras

11. Find whether path exist : dijkstras

12. Minimum Cost Path : dikstras

13. Circle of Strings

14. Floyd Warshall

15. Alien Dictionary

16. Snake and Ladder Problem

prims : greedy , start with one vertex, include path with min weight that connect two set

kushals: sort all edges , include min if not forming cycle, Graph has edge list implementtion

Google Mock interviews

-----------------------

Given a time (in 24-hour format) with missing digits marked as '?', we want to replace all of the question marks with digits (0-9) in such a way as to obtain a valid time. The earliest possible time is 00:00 and the latest valid time is 23:59.

Write a function that, given a string in the format "HH:MM", returns an integer denoting the number of valid times that can be obtained by replacing the question marks.

11:?0 -6

1?:?1 -

??:??

Based on second digit, compli 6 10

2nd digit

0-3 -> 3

4-9 -> 2

1st digit 0 -> 10

1 ->10

2 -> 4

? -> 24 POSSIBILITY

No of ?

Places of ?

1?

Based on second digit, compli 6 10

2nd digit

0-3 -> 3

4-9 -> 2

1st digit 0 -> 10

1 ->10

2 -> 4

? -> 24 POSSIBILITY

2?

Place 12 -> 24

13 -> 1st digit logic \*6

14 -> 1st digit logic \*10

23-> 2nd logic \* 6

24->2nd logic \* 10

34 - 6 \*10

3?

123 -> 24\* 6

124 ->24 \*10

234-> 2nd digit logic\*60

134-> 1st digit logic\*60

4?:24 \*60

Int iAns =1

? -> place

iAns \*= digitsLogic

2nd digit

0-3 -> 3

4-9 -> 2

Int FirstDigLogic(char iSecondDigit)

{

switch(iSecondDigit)

{

Case ‘0’: return 3;

Case ‘1’: return 3;

Case ‘2’: return 3;

Case ‘3’: return 3;

Case…

Case’?’ : return 24

}

}

Int SecondDigLogic(char iFirstDigit)

{

}

Int ThirdDigLogic()

{

return 6;

}

Int ForthDigLogic()

{

return 10;

}

Int main()

{

String strInput;

Int iAns = 1;

Int i = 1 // counter

For (char c : strInput)

{

If (c == ‘?’)

Switch (i)

{

Case 1:

iAns\*= FirstDigitLogic(strInput[1]);

Break;

Case 2:

iAns\*= FirstDigitLogic(strInput[1]);

break;

…

}

i++;

}

}

=============================

[Google SDE Sheet: Interview Questions and Answers - GeeksforGeeks](https://www.geeksforgeeks.org/google-sde-sheet-interview-questions-and-answers/)

Reverse a Linked List: Write a function to reverse a singly linked list.

void Reverse(Node\*\* ptrHead)

{

if(\*head == nullptr)

return;

Node\* ptrPrev = nullptr;

Node\* ptrCur = \*ptHead;

Node\* ptrNext = nullptr;

while(ptrCur)

{

ptrNext = ptrCur->next;

ptrCur->next = ptrPrev;

ptrPrev = ptrCur;

ptrCur = ptrNext;

}

\*ptrHead = ptrPrev;

}

=======================

Find the Longest Substring Without Repeating Characters: Given a string, find the length of the longest substring without repeating characters.

an array to store lastIndefof[ch]

LongestSubWithoutRepeatingChar(string strInput)

{

unordered\_map<char, int> mapLastIndex;

int i = 0, res = 0, index = 0;

for(char ch : strInput)

{

i = max(i, mapLastIndex[ch] + 1)

res = max(res, index - i +1);

mapLastIndex[ch] = index;

index++;

}

}

LongestSubWithoutRepeatingChar(string strInput) //n^2

{

int start = 0 , end = 0, curSize = 0, maxSize = 0, numOfFreqGT1 = 0;

unordered\_map<char, int> mapFreq;

int len = strInput.size();

while(end < len)

{

map[strInput[end]]++;

if(map[strInput[end]] > 1)

{

numOfFreqGT1 = 1;

curSize = end - start + 1;

if (curSize > maxSize)

maxSize = curSize;

}

end++;

While(start < end && numOfFreqGT1)

{

map[strInput[start]]--;

if(map[strInput[end]] == 1)

{

numOfFreqGT1--;

}

start++;

}

}

return maxSize;

}

==============

Implement a Stack using Queues: Implement a stack data structure using two queues. The stack should support the usual push and pop operations.

class Stack

{

queue<int> q1;

queue<int> q2;

int icurq = 1;

}

void STACK::push(int i)

{

if(iCur == 1)

q1. push(i);

else

q2.push(i);

}

int STack::pop()

{

if(iCur == 1)

{

if(q1.empty())

return 0;

for(int i = 0 ; i < q1.size() - 1; i++)

{

int item = q1.pop()

q2.push(item)

}

icur = 2;

return q1.pop()

}

else

{

if(q2.empty())

return 0;

for(int i = 0 ; i < q2.size() - 1; i++)

{

int item = q2.pop()

q1.push(item)

}

iCur = 1;

return q2.pop()

}

}

=======================================

Find the Median of Two Sorted Arrays: Given two sorted arrays, find the median element of the combined array formed by merging the two arrays.

https://www.geeksforgeeks.org/median-of-two-sorted-arrays-of-different-sizes/

FindMedian(vector<int> vec1, vector<int> vec2)

{

int n = vec1.size();

int m = vec2.size();

if (m > n)

findMedian(vec2, vec1)

int start = 0, end = n;

median = (m + n+ 1)/2;

while (start <= end)

{

leftAIndex = (start + end)/ 2;

leftBIndex = median - leftA;

leftA = vec1[leftAIndex];

leftB = vec2[leftBIndex];

rightA = vec1[leftAIndex + 1];

rightB = vec2[leftBIndex + 1];

if(leftA <= rightb && leftB <= RightA)

{

if((m+n)%2 != 0)

return max(leftA,leftB);

else

return (max(leftA,leftB) + min(rightA , rightB) /2);

}

else

if(leftA > rightB)

end = mid -1;

else

start = mid+ 1;

}

}

===========

Implement a Trie (Prefix Tree): Implement a trie with insert, search, and startsWith methods.

class trieNode{

public:

bool m\_bIsEnd;

trieNode\* m\_ptrChild[26];

trieNode(): m\_bIsEnd(false){

for(int i = 0; i < 26; i++)

{

m\_ptrChild[i] = nullptr;

}

}

}

class trie{

public:

trieNode\* root = nullptr;

void insert( string strInput);

bool search( string strInput);

bool startsWith(string strInput);

}

void trie::insert(string strInput)

{

if(root == nullptr)

{

root = new trieNode();

}

treiNode\* cur = root;

for(const auto ch : strInput)

{

if(!cur->m\_ptrChild[ch-'a'] )

cur->m\_ptrChild[ch-'a'] = new trieNode();

cur = cur->m\_ptrChild[ch-'a'];

}

cur->m\_bIsEnd = true;

}

bool trie::search(string strInput)

{

if(root == nullptr)

return flase;

trieNode\* cur = root;

for( const auto ch : strInput)

{

if(cur->m\_ptrChild[ch -'a'] == nullptr)

return false;

cur = cur->m\_ptrChild[ch -'a']

}

if(cur->m\_bIsEnd == true)

return true;

return false;

}

bool trie:: startsWith(string strInput)

{

if(root == nullptr)

return flase;

trieNode\* cur = root;

for( const auto ch : strInput)

{

if(cur->m\_ptrChild[ch -'a'] == nullptr)

return false;

cur = cur->m\_ptrChild[ch -'a']

}

return true;

}

=======

Check if a Binary Tree is a Binary Search Tree: Given a binary tree, determine if it is a valid binary search tree (BST).

IsBST(Node\* root, int min/inT\_MIN/, int Max/INT\_MAX/)

{

if(root->data < min || root->data > max)

return false;

if(! ISBST(root->left, min, root->data))

return false;

if(! ISBST(root->right, root->data, max))

return false;

return true;

}

=======

Serialize and Deserialize a Binary Tree: Design an algorithm to serialize and deserialize a binary tree into a string and back to a tree.

void serialize(node\* root, File\* file)

{

if(root == null)

{

file << "-1 ";

return;

}

file << root->val;

serialize(root->left, file)

serialize(root->right, file)

}

void Deserialize(node\*\* root, File\* file)

{

int read = 0;

file >> read;

if (read == -1)

return;

\*root = new Node(read);

Deserialize(\*root->left, file)

Deserialize(\*root->right, file)

}

=======

Find the Kth Largest Element in an Array: Find the kth largest element in an unsorted array. Note that it is the kth largest element in the sorted order, not the kth distinct element.

use priority q

=====

Given a Matrix of 0s and 1s, find the Biggest Square of 1s: Given a binary matrix of 0s and 1s, find the size of the biggest square of 1s in the matrix.

int BiggestSq(int ip[][], int m, int n)

{

int DP[m][n];

for(0 to n)

DP[0][i] = ip[0][i]

for(0 to m)

DP[m][0] = ip[m][0]

int iMax = 0;

for(i :1 to m)

for(j: 1 to n)

{

if(IP[i][j] == 0)

DP[i][j] = 0;

else

{

DP[i][j] = 1 + min(DP[i-1][j-1], DP[i-1][j], DP[i][j-1])

if(DP[i][j] > imax)

iMax = DP[i][j];

}

}

}

=====

Find the Intersection of Two Linked Lists: Write a program to find the node at which the intersection of two singly linked lists begins.

Node\* intersection(Node\* root1, Node\* root2)

{

int iLen1 = len(root1);

int iLen2 = len(root2);

int iDiff = 0;

Node\* ptrBig, ptrSmall;

if(len1 > len2)

{

iDiff = iLen1 - iLen2;

ptrBig = root1;

ptrSmall = root2;

}

else

{

iDiff = iLen2 - iLen1;

ptrBig = root2;

ptrSmall = root1;

}

for(i = 0; i < iDiff; i++)

{

ptrBig = ptrBig->next;

}

while(ptrBig && ptrSmall)

{

if(ptrBig == ptrSmall)

return ptrBig;

ptrBig = ptrBig->next;

ptrSmall = ptrSMall->next;

}

}

============================

find a element in rotated array

after finding pivot , based on condition either search in first arr or sec arr that will save from so much manipulation of index

FindRotPoint(int arr[], int min//0, int max //n-1)

{

while(min< max)

{

mid = min + (max - min)/ 2;

if(arr[mid-1] > arr[mid] )// stop when prev ele is larger

return mid;

if(arr[min] > arr[mid])

max = mid -1;

if(arr[max] < arr[mid])

min = mid+ 1;

}

}

Find(int arr[],int n, int k)

{

int start = RotPt = FindRotPoint(int arr[], int min//0, int max //n-1);

if (arr[Rotpt] <=k && k <= arr[n-1])

binarySearch(arr, k, rotPt, n-1)

else

binarySearch(arr, k, 0, rotPt- 1)

}

\*/

//wrong aproach

/\*

Find(int arr[],int k, int n, int start, int end)

{

len = n;

while(len)

{

mid = (start + len / 2) % 7;

if(arr[mid] == k) return mid;

if(arr[mid] > k)

{

prevSTart = start

start = mid + 1 % 7;

if(start - prevStart > 0)

len =len - (start - prevStart)

else

len = (prevStart - 1 - start) + 1

}

else{

int end = ((mid - 1) >= 0) ? (mid - 1) : n-1;

len = end - start + 1;

}

}

}

Find(int arr[],int n, int k)

{

int start = RotPt = FindRotPoint(int arr[], int min//0, int max //n-1);

int end = ((start - 1) >= 0) ? (start - 1) : n-1;

int index = Find(int arr[],int k, int start, int end)

}

\*/

Google interview 23 jul

1. Find average of cont input running time.
   1. Remove spikes i.e. big 25% numbers
2. Input is a signed int sorted array, output should be square of all ele , sorted, time complexity : O(n)
   1. Soln : cal square of neg num , push\_front to a list
   2. Cal sq of pos num , push back to a list
   3. Merge two list