Page No.
Date

30/12

Pointers :-

* Symbol table : The table in which variable & the memory location where the variable is stored is saved.

&i → Address of i.

* Pointers are variables which store the address of another variable.

int *p = &i ;

*p → dereference.

* Memory allocated for pointers → 8 bit

* Always initialise the pointer with zero.

Pointer arithmatic :-

→ p++ , p = p+1 → go to the next integer.
+8 bytes.

Scanned by CamScanner

Page No.
Date

OOPS :

→ ~~Abstraction~~ Encapsulation :-
- → clubbing of data + functions.
- → Achieved using classes.

→ Abstraction : hiding the unnecessary details.

→ Inheritance :
- → Reusability.

Access modifiers
- → Private
- → Public
- → Protected & only child class can use it.

Inheritance syntax :-

```
class car : public Vehicle {

};
```

* private properties are never inherited.
* when access modifier is public.
  - Public → public
  - Protected → protected.
* Protected members cannot be used directly they can only be used inside the derived class.

For protected
- Public → Protected
- Protected → Protected

For Private
- Public → Private
- Protected → Private

* By default it is private.

Scanned by CamScanner

Page No.
Date

Constructor call : A()
B()
Destructor call ~B()

SIEVE of Eratosthenes.

For all numbers a : from 2 to sqrt(n)
if a is unmarked then
a is prime.
for all multiples of a (a<n)
mark multiples as composite.

TRICKS with Bits :-

1. x & (x-1) will clear the lowest set bit of x
2. x & ~(x-1) extracts the lowest set bit of x.
3. x & (x + (1 << n)) = x

Scanned by CamScanner

Page No.
Date

NUMBER THEORY :

→ Sieve of eratosthenes

Time complexity :

$i = 2$ to $\sqrt{N}$

$j = i * i$, $i++$

$$\frac{N}{2} + \frac{N}{3} + \frac{N}{5} + \dots$$

$$= N\left(\frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \frac{1}{7} + \frac{1}{11} + \dots\right)$$

$$= O(N \log \log N)$$

Modulo Arithmetic :

modulo $(10^9 + 7)$

* $(a+b)\%m = (a\%m + b\%m)\%m$.

** For negative numbers

$a\%m = (a+m)\%m$.

Scanned by CamScanner

Page No.
Date

HASHMAPS :-

→ Map between keys & values.

→ Map (BST) - O(logn) vs Unordered-Map. O(1) (Hash table)

```
pair<string, int> p("abc", 1);
ourmap.insert(p);
ourmap["def"] = 2;
```
} 2 methods to insert.

// find or access :

```
ourmap["abc"];
ourmap.at("abc");
```
} access.

** if we use at & the key is not present then it will through an exception.

*** when we use [] then it will insert with a count 0.

// check presence

```
if (ourmap.count("ghi") > 0)   // count.
```

→ `ourmap.size()` // to find size.

→ `ourmap.erase("ghi")` // erase / remove

⇒ ITERATOR :

```
unordered_map<string, int>:: iterator it = map.begin
while (it != map.end()) {
    it ++;
}
```

it → first (key) it → second (value)

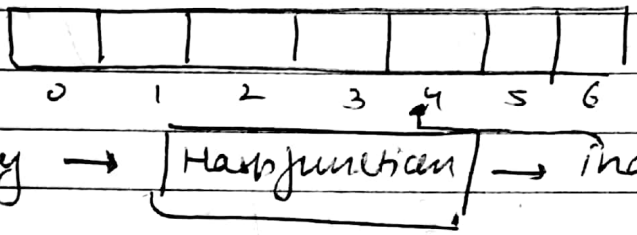
```
vector<int>:: iterator it = v.begin();
```

Scanned by CamScanner

→ map.find("abc") // gives an iterator.

→ We can pass an iterator to the erase function.
We can also pass a range.
eg: map.erase(it, it+3);

* Bucket Array :



* Hash Function → hash code
→ compression Function.

→ Collision Handling
↳ open Addressing → Linear probing
→ Double hashing
↳ quadratic probing
↳ closed Addressing
(seperate chaining).

** SORT A MAP BY Value :-

```
template <typename T1, typename T2>;
struct less_second {
    typedef pair<T1, T2> type;
    bool operator() (type const a, type const b) {
        return a.second < b.second;
    }
};
```

Scanned by CamScanner

Page No.
Date

```
vector < pair < string, int >> V (m.begin(), m.end());
sort ( V.begin(), V.end(), less-second < string, int >() );
```

* copy to a vector :-

```
copy ( X.begin(), X.end(), back_iterator(v));
```

first_iterator last_iterator.

BIT MANIPULATION :-

→ Used for compression.
→ Used for. Data encription.

\# Left shift operator :

number << no-of-places (x)

N = 8 1

0 0 0 0 1 0 0 0

0 0 0 1 0 0 0 0

Number is multiplied by 2^x.

8 << 1

$N << i \Rightarrow N = N * 2^i$

\# Right shift operator

number >> no. of places (x)

N=8 N>>1

0 0 0 0 1 0 0 0

0 0 0 0 1 0 0 0

↑

if number is +ve add 0
otherwise add 1.

Scanned by CamScanner

0 0 0 0 0 0 0 0
1 1 1 1 1 0 1
+1
1 1 1 1 1 1 1 0

Page No.
Date

Number gets divided by 2.

$N = N/2^i$.

0 0 0 1
0 0 1 0

Bitwise AND :

$x \& 0 \rightarrow 0$

$x \& 1 \rightarrow x$.

Bitwise OR :

$x | 1 = 1$

$x | 0 = x$.

~ → Not (compliment). / 1's compliment.

XOR $x \wedge 1 = \sim x$

$x \wedge 0 = x$.

→ Flip ith bit :- $n \wedge (1 << i)$

→ Set $n | (1 << i)$

→ unset $n \& (1 << i)$

→ check if number is odd or even.

$n \& 1$

→ powers of 2 or not.

```
while (N%2 == 0) {
    N = N/2;
}
if (N == 1)
    return true
else false.
```

$n \& (n-1)$

0 ✓ if 2^i

1 Not powers of 2.

Scanned by CamScanner

Page No.
Date

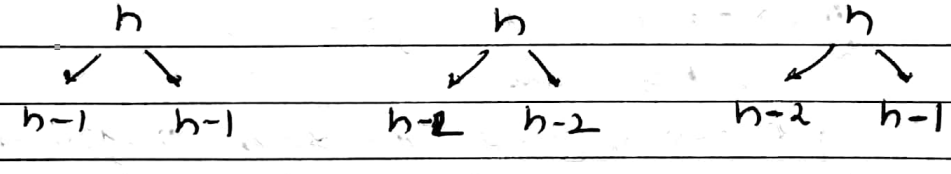
Trees :-

① Diameter of a binary tree :

max(lheight + rheight + 1 , max(ldiameter, rdiameter))

Maximum of
1. left diameter
2. Right diameter
3. lheight + rheight + 1

② Balance binary tree :

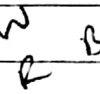


Possibility of $h-1 \to x$, $h-2 \to y$.

h = 0	= 1
h = 1	= 1

W B

FLAGS :-




```
(N, prev, prevprev)
    if (prev == 'B' && prevprev == 'W')
            (N-1, 'R', 'B')
    if (prev == 'B' && prevprev == 'R')
            (N-1, 'W', 'B')
    if (prev == 'W')
            (N-1, 'B', 'W');
            (N-1, 'R', 'W');
```

Scanned by CamScanner

Page No.
Date

BACKTRACKING :

→ RAT IN THE MAZE :.

Variations :
→ Is there a path.
→ Print all paths.

* We maintain another 2D array which keeps track of all the visited places.

```
bool haspath(int **arr, int n){
    solution : { new 2D array }
    return haspathHelper(arr, n, solution, 0, 0);
}
helper(int** arr, int n, int** solution, int x, int y){
    if(x == n-1 && y == n-1)
        return true.
    if(x<0 || y<0 || x>=n || y>=n ||
        input[x][y] == 0 || solution[x][y] == 1){
            return False
    solution[x][y] = 1;
    if(helper(arr, n, solution, x-1, y)
        return true;
    if→ do this for (x, y-1)
    if                (x+1, y)
    if                (x, y+1)
    solution[x][y] = 0;
    return False.
}
```

→ Print all paths :

Scanned by CamScanner

→ In the base case print the solution matrix.

```
if (x == n-1 && y == n-1)
    solution[n][y] = 1;
    for(int i=0; i<n; i++){
        for(int j=0; j<n; j++){
            cout << solution[i][j] << " ";
    cout << endl;
    return;
}.
```

→ Other base case.
→ Then call for all 4 directions.

* **SUDOKU :**

- if (i == n-1 && j == n-1)

1. All boxes are filled we are at n-1, n-1 : true.
2. Reached a box which is already filled : true.
3. " " " " not filled.
   → Find all the options
   → Try all the options & call forward.

* No. of numbers without duplicates :

```
int num(int* input, int n, int* fact, int* freq){
    if(n==0 || n==2)
        return 0;
    int ans=0;
    for(int i=input[0]+1; i<10; i++){
        if(freq[i]>0)
            ans += fact[n-1];
    }
```

Scanned by CamScanner

```
    freq[input[0]]--;
    ans += sum(input+1, n-1, fact, freq);
    return ans;
}
```

Solve sudoku :

```
if (x==9)
    return true;
if (y==9)
    return solve(board, x+1, 0);
if (n[x][y]!=0)
    return solve(board, x, y+1);
for(int i=1; i<=9; i++){
    if(safe(board, x, y, i)){
        board[x][y]=i;
        if(solve(n, x, y+1)){
            return true;
        }
    }board[x][y]=0;
}
return False.
```

In safe() we check for row, col & inner square.

Subset Array :

```
if(n==0){
    output[0][0] = 0
    return 1;
}
```

Scanned by CamScanner

```
int so = subset(input+1, n-1, output);
for(int i=0; i<so; i++)
    output[i+so][0] = output[i][0] + 1

for(int i=0; i<so; i++){
    for(int j=1; j<=output[i+so][0]; j++){
        if(j==1) output[i+so][j] = input[0]
        else
            output[i+so][j] = output[i][j-1];
```

* Print subsets:

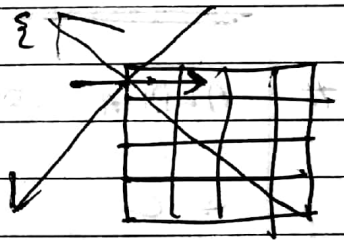
2 choices + the element is passed

OR

Not passed.

* N queens :-

```
isSafe(board, N, row, col){
    int i, j;
    for(i=0; i<col; i++)
        if(board[row][i])
            return false
```



upper diagonal on left

```
    for(i=row, j=col; i>=0 && j>=0; i--, j--)
        if(board[i][j])
            return false;
```

lower diagonal on left

```
    for(i=row, j=col; i>=0 && i<=N; i++, j--){
        if(board[i][j])
            return false;
    }
    return true;
}
```

$x = 2.0^{n}$

$y = 1$

n++ y*=x

00	01	02	03
10	11	12	13
20	21	22	23
30	31	32	33

Scanned by CamScanner

```
bool helper( board, N, col )
    if ( col == N ) {
        Print
        return true;
    }
    bool res = false;
    for (int i=0; i<N; i++) {
        if ( issafe( board, N, i, col ) {
            board[i][col] = 1
            res = res || helper(board, N, col+1)
            board[i][col] = 0
        }
    }
    return res;
}
```

DYNAMIC PROGRAMMING :-

ALPHA CODE :-

→ call for the single digit.
→ if (ch-1) * 10 + (ch) >= 10 && <= 26
    call for the 2 digits.

Scanned by CamScanner

Page No.
Date

LIS : (Longest increasing subsequence)

```
output[0] = 1;
for (int i = 1 ; i < n ; i++) {
        output[i] = 1 ;
        for (int j = i-1 ; j >= 0 ; j--) {
                if ( arr[i] > arr[j] ) {
                        output[i] = max(output[i], output[j]+1
                }
        }
}
```

Find the best by traversing once.

No. of balanced binary trees :-

$$n(h) = n(h-1) * n(h-1) + 2 * n(h-1) * n(h-2)$$

$h = 0 \Rightarrow 1$

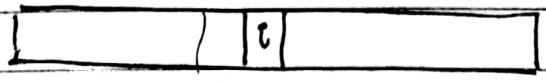
$n = 1 \Rightarrow 1$

$3 * 3 + 2 *$

No. of BST's possible :- $O(n^n)$

* $n\,BST(K-1) + nBST(n-K)$

* $nBST(n) = \sum_{K=1}^{n} n\,BST(K-1) + nBST(n-K)$



Scanned by CamScanner

Page No.
Date

for (i=2 ; i<=n ; i++)

out[i] = 0;

for (i=1 ; i<=i ; i++) {

out[i] = out[i] + (out[i-1] * out[i-i])

use modulo for all operation.

}

}

Largest Sum Subarray.

curr_max_end = 0

best_so_far = 0 .

Boredom :

dp[i] = max (dp[i-2] + i*freq[i] , dp[i-1])

Dry Run

1 2 3 5 1 2 4 11 5 6.

0 1 2 3 4 5 6 7 8 9 10 11

Freq[0 2 2 1 1 1 0 0 1 0 0 1]

dp [0 1 2 3 4 5 6 7 8 9 10 11]

0 2 4 5 6 15 15 15 23 23 23 34.

dp[2] = max (dp[0] + 2*2 , dp[1])

i-2 i-1

dp[n] =

11

1 2 3 5 1 2 4 11 5 6

34

~~Uploaded Heap Sort~~

Minimum Number of chocolates :

left to right :

```
if (arr[i] > arr[i-1])
        dp[i] = dp[i-1] + 1;
else if{
        dp[i] = 1.
```

O(n)

```
if arr[i] > arr[i+1] && dp[i] <= dp[i+1]
        dp[i] = 1 + dp[i+1].
```

→ When two arrs depends on both ways of traversing.

→ 1st go from left to right & assign the values & after that go from right to left and optimise your output.

No. of APs

```
int numofAP(int * arr, int n) {
        int ans = n+1;
        unordered_map<int,int> * map = new
                unordered_map<int,int>[n];
        for(i=0; i<n-1; i++) {
                for(j=i+1; j<n; j++) {
                        int diff = arr[j] - arr[i];
                        map[i][diff] += 1;
                        ans = (ans+1)%Mod;
                }
        }
```

Scanned by CamScanner

Page No.
Date

```
for (int i = n-3 ; i >= 0 ; i--) {
    for (int j = i+1 ; j < n ; j++) {
        int diff = arr[j] - arr[i];
        map[i][diff] += map[j][diff];
    map[i][diff] = (map[i][diff] + map[j][diff])
        ans = (ans + map[j][diff]) % mod; %mod;
    }
}
delete [] map;
return ans;
```

Min count :-

```
    0  1  2  3  4  5  6  7  8  9.
[   1  2  3                    ]
int *dp = new int[n+1]
dp[0] = 0
dp[1] = 1
dp[2] = 2
dp[3] = 3
for (int i = 4 ; i <= n ; i++) {
    dp[i] = i;
    for (int j = 1 ; j <= ; j++) {
        int temp = j*j;
        if (temp > i) break;
        else
            dp[i] = min (dp[i], 1 + dp[i - temp]);
    }
}
```

Scanned by CamScanner

Page No.
Date

- COIN Tower.

Assume yourself as 1 player.

- Longest consecutive subsequence :-

→ Creat a hashmap of all the array elements
→ For each element start counting for a sequence.

- Longest common subsequence. :-

if 1st character is same.

$1 + LCS(S_1+1, S_2+1)$

else

$\max(LCS(S_1+1, S_2), LCS(S_1, S_2+1))$;

Brute force time complexity :- exponential. 2^n

- coin change :-

```
dp[0] = 1
for(i=0 ; i=n ; i++){
    for(int j=a[i] ; j<=x ; j++){
        dp[j] = dp[j] + dp[j-a[i]] ;
```

eg:- n=3 {1,2,3} X=4

dp [1 1 2 2 4]

i=0 a[i]=1
j=1 2

Scanned by CamScanner

Page No.
Date

Q Edit distance :-

1. Insert a char.
2. delete a char
3. substitute a char.

similar to LCS ..

Q Minimum number of jumps :-

Q Matrix chain Multiplication. :-

array of n+1 size $a_0 \dots a_n$.

$m_i = a_{i-1} * a_i$

f(a, s, e) :

$k = s+1$ & $k = e-1$

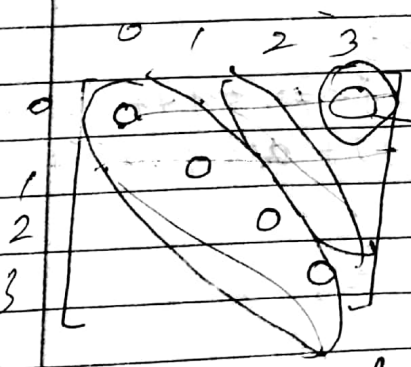
$f(a,s,e) = \min(f(a,s,k) + f(a,k,e) + a[s] * a[k] * a[e])$

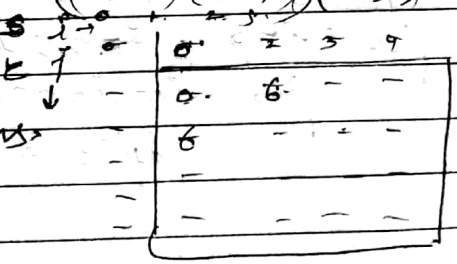
if (s == e) return 0

if (e - s == 1) return 0.

[2 3 4 5]

((2*3)(3*4))(4*5)





$dp[i][j] = \min(dp[i][j], dp[i][k] + dp[k][j] + dp[i] * a[k+1] * dp[j+1])$

Scanned by CamScanner

Page No.
Date

HEAPS And Priority queue :-

Priorit Queue :- Extension of queue (FIFO)
↳ element is removed on the basis of its priority.

min priority queue
max priority queue.

	1). Insert	2) getMax / getMin. (top)	3) removeMax removeMin. (pop).
(implementation :)			
Array (unsorted)	O(1)	O(n)	O(n)
Array (Sorted)	O(n)	O(1)	O(n)
LL (US)	O(1)	O(n)	O(n)
LL (S)	O(n)	O(1)	O(1)
BST	O(~~h~~)	O(~~h~~)	O(~~h~~)
Balanced BST	O(logn)	O(logn)	O(logn)

INBUILT : #include <queue> By default max PQ.

priority_queue < >

→ empty() → size() → push()

→ getMax() → top() → removeMax()

pop()

Concerns in BST :-

1) Balancing
2) store.

Scanned by CamScanner

Page No.
Date

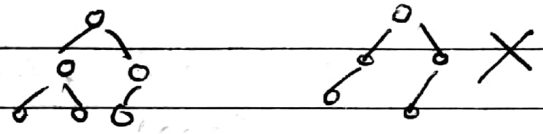
⇒ Heaps : (CBT)

1) A complete binary tree.
2) heap order property.

What is a 'complete binary tree (CBT) :

→ All levels of a binary tree must be filled except the last level.

eg :



And the last level should also be filled in left to right manner.

1 Height :

min no. of nodes with height h CBT :-

↳ $2^{h-2}+1$ × max = $2^{h+1}-1$

2^{h-1}

$$2^{h-1} \leq n \leq 2^h - 1$$

$2^{h-1} \leq n$ $\quad$ $n \leq 2^h - 1$

$h-1 \leq \log_2 n$ $\quad$ $\log(n+1) \leq h$

$h \leq \log_2 n + 1$

$$\log_2(n+1) \leq h \leq \log_2 n$$

→ The height of CBT is $\log(n)$.

Scanned by CamScanner

pq.pop_back();

Page No.
Date

→ Storing

left child : $2i+1$

right child : $2i+2$.

Parent : $\frac{\text{childindex} - 1}{2}$ $\left\{ \frac{2i}{2}, \frac{2i+1}{2} \right\}$

→ Heap Order property :-

* Min heap : the value of the root node should be less than that of the child / ~~parent~~ node.

* max heap : the value of the root node should be greater than its child element.

→ upheapify → toggling and swapping the elements. (INSERTION)

→ Down heapify :- (Deletion)

```
insert ( int data ) {
    pq.push_back(element);
    int childIndex = pq.size() - 1;
    int parentIndex = (childIndex - 1) / 2;
    while(CI > 0) {
        if (pq[childIndex] < pq[parentIndex]) {
            swap ( pq[CI], pq[PI] );
        }
        childIndex = PI;
    }
```

put this inside a while loop.

Scanned by CamScanner

Page No.
Date

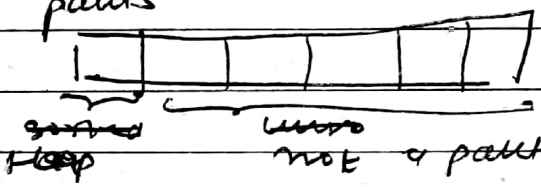
In place heap sort :-

i : cI P

→ First do insertion from 1 to n-1

→ Then do remove operation from n-1 to 0

→ Consider the array having sorted & unsorted parts.




```
ios_base :: sync_with_stdio(false);
cin.tie(NULL);
```

Scanned by CamScanner

Page No.
Date

TREES :

```
template <typename T>
class TreeNode {
public:
    T data;
    vector<TreeNode<T>*> children;
```

→ Automatic base case handling.
~~As if need is only 1 c~~
If the root doesn't have a child it automatically acts as a base case.

→ Level wise printing : use queue.

• Height of a tree :- height = 1;
height = max(height, height of tree(root→children[i]))

• Depth of a tree :-

→ Print at level K :-

```
if (k == 0) {
    cout << root->data << endl;
}
for (int i=0; i < root->children.size(); i++) {
    printatlevelk(root->children[i], k-1);
}
```

BINARY TREES :-

* Construct A tree from Preorder & Inorder.
Preorder root left Right
Inorder left root Right.

Scanned by CamScanner

Page No.
Date

- To build left subtree we need lefts inorder & preorder.
- same goes for right sub. tree.

root left right
left root right

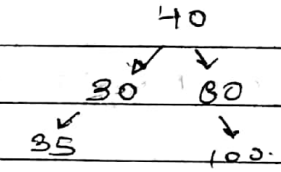
Left IN → LIOS LIOE
Right PO → RPOS - LPOE

LINE - LIOS = LPOE - LPOS
LPOS + (LINE - LIOS) = LPOE.

Tree from Postorder & Inorder,

Inorder Left Root Right
Postorder, Left Right Root.

Preorder Root Left Right
Postorder. Left Right Root.

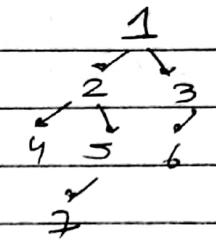


BST R
Smaller Larger.

→ Level wise linked list :-.

queue. h = N t = N

1 → NULL, 2 → 3 → NULL, 4 → 5 → 6 → NULL, 7 → NULL



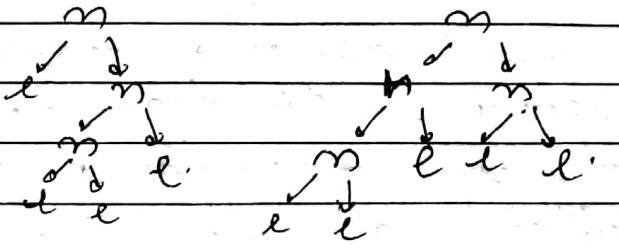
1 → NULL,
2 → 3 → NULL
4 → 5 → 6
7.

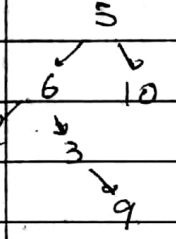
Scanned by CamScanner

Page No.
Date

Nice Binary tree :-

→ ~~Serialize~~ n l n n l l l.





n n n l l l n l l.

1 2 3 4 5

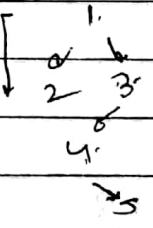
→ longest path from ~~root~~ leaf to root :-

→ call for both right & left
→ check the size of the returned vector
→ Add the root→data to the vector whose size is larger.

→ Max Sum leaf to node :-

There are 4 scenarios which are to be taken care of :-

→ root only is max
→ root + l / root + r
→ root + l + r
→ call for left & right side.



→ Boundary traversal :-

→ Print left boundary.
→ Print leaf nodes.
→ Print right boundary.

Scanned by CamScanner

Page No.
Date

BINARY SEARCH TREE :-

1. For every node n → everything in n's left subtree is less than n's data.
2. Everything in n's right subtree should be greater than n's data.

IsBST :-

Time complexity.

max : $T(n) = 2T(n/2) + kn$.

$\Rightarrow \Theta(n \log n)$

min : $T(n) = 2T(n/2) + kn$.

$\Rightarrow O(n \log n)$

$\boxed{O(nh)}$

improving time complexity :-

```
class isBSTReturn {
    public:
        bool isBST;
        int minimum;
        int maximum;
};
```

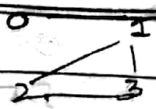
using this we can reduce the complexity by O(n)

Side view of BST :-

Scanned by CamScanner

Page No.
Date

GRAPHS :-



→ Trees are always connected.

→ Trees are cyclic.

→ Nodes → vertices connections → edges.

→ Two vertices are adjacent if they have an edge between them.

→ Degree of a vertex → the no. of edges from/coming toward an vertex.

→ Path : A collection of edges from which one can reach from one vertex to another if they don't have a direct edge between them.

→ Min no. of edges in a connected graph.
$(n-1)$

→ Complete graph : $^nC_2$

IMPLEMENTATION :-

① Edge List
② Adjacency list.
③ Adjacency Matrix.

Depth First search :-

```
cout << sv << endl;
visited[sv] = true;
for(int i=0; i<n; i++){
    if(i==sv) continue
    if(edges[sv][i]==1)
        if(visited[i]==0)
            print(edges, n, i, visited
```

Scanned by CamScanner

Page No.
Date

BFS → queue.
DFS → stack

Take input for dynamically allocated 2d array :-

```
int **edges = new int*[v];
for(int i=0; i<n; i++){
    edges[i] = new int[n];
```

- Directed graphs : A → B
  (one way edges).
- Weighted graph :.
- Eulerian path :-
  If there are exactly 2 vertex with with odd no. of degree then it has an eulerian path.

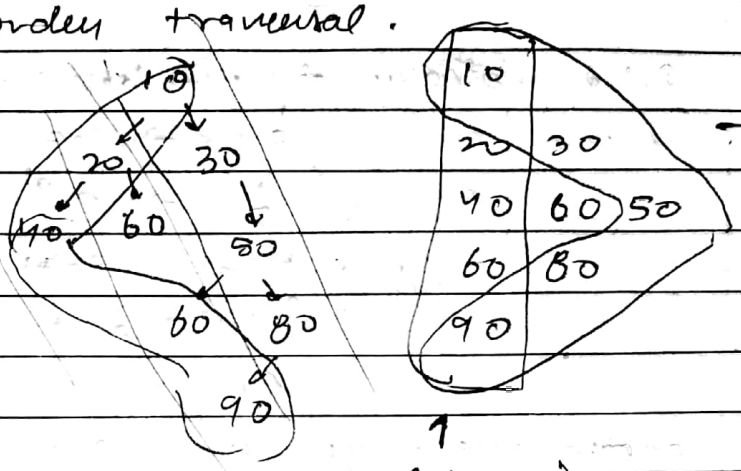
Scanned by CamScanner

2135515547O 9T6B3F2

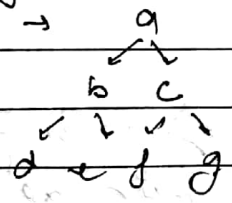
Page No.
Date

Left view of BST :-

→ use level order traversal.



→ Binary tree to doubly linked list :-



BFS :- a ⇄ b ⇄ c ⇄ d ⇄ e ⇄ f ⇄ g → Null

INorder : d ⇄ b ⇄ e ⇄ a ⇄ f ⇄ c ⇄ g

- Binary search tree class :-

Delete :- The root can be replaced by maximum element of left subtree or minimum element of right subtree.

- Lowest common Ancestor :-

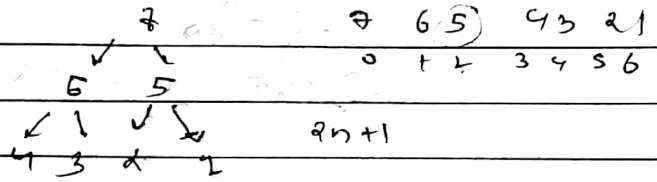
```
if (root == null)
        return root;
if ( root→data == n1 || root→data == n2)
        return root;
Node* left = LCA(root→left, n1, n2);
Node* right = LCA (root→right, n1, n2);
if ( left && right ) return root;
if(left) return left
else return right;
```

Scanned by CamScanner

Page No.
Date

- connect nodes at same level :-
  * use level order traversal.

- Print binary tree levels in sorted order.



- bottom view of a Binary tree :-

vertical order traversal.

Hd = horizontal distance.

For root hd = 0

For left hd = hd parent − 1

For right hd = hd parent + 1

// if hd is not in the definition of the tree.

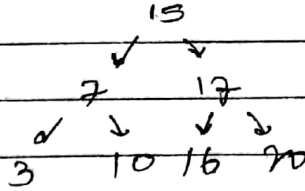
make the queue as a pair of Node* & hd.

map<int, int> ; d=0 ;

queue<pair<Node*, int>> q ;

~~queue~~ q.push (make_pair (root, d)) ;

- All nodes distance k in a binary tree.



:- Create a map storing the parent of all the nodes.

→ This will help us to create an undirected graph..

Scanned by CamScanner

→ create an map, seen array & queue.

→ Then do level order traversal starting from the ~~root~~ target node.

Diagonal traversal :-

```
queue<Node *> q;
q.push(root);
while(!q.empty()){
    Node *t = q.front();
    while(t){
        cout << t->data << " ";
        if(t->left)
            q.push(t->left);
        t = t->right;
    }
    q.pop();
}
```

Page No.
Date

9807
a b
6541
c d

Fast multiplication :-

$(10^n)ac + (10^{n/2})(ad+bc) + bd.$

Atomic Multiplication → 4

" Addition → 4.

$M(n) = M(n/2) + 2n.$

Scanned by CamScanner

Page No.
Date

GRAPHS :-

→ A tree is a graph which is connected & doesn't contain a cycle.

Spanning Tree :- A graph which is connected & undirected. The tree which connects all the vertices.

minimum ST :- (weighted graphs)

- Kruskal's Algorithm :-
  → Pick the edges one by one with min weight.
  → check whether cycle is created when we add an edge to the MST.

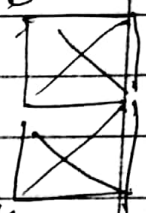
naspan → (O(V+E))

- cycle detection :-
  → Before adding an edge check whether there is a path between the 2 vertices or not.

not good as complexity is concerned.
As far as

→ Union-find Algorithm :-

	0	1	2	3	4	5
Parent →	0	1	2	3	4	5



	0	1	2	3	4	5

complexity →

Scanned by CamScanner

```
bool cycle (int v, int **edges, int parent, bool
                                              vis) {
    vis[v] = true;
    for(int i=0; i<v; i++) {
        if (edge[v][i] == 1) {
            if (!vis[i]) {
                if (cycle(i, edges, v, vis)
                    return true;
            }
            else if (i != parent)
                return true.
        }
    }
    return false;
}
```

Run this inside the main function for all connected components.

Kruskal's :

1) Take input
2) Sort the input array.
3) count = 0, i = 0, parent[v].
   while (count < n-1) {
       find parent for source & dest
       (can not be same)
       add edge.
   }

   complexity. : O(EV) + ElogE.

→ Union by Rank & path compression.

Scanned by CamScanner

Page No.
Date

PRIM'S ALGORITHM :-

Source	Parent	weight	visited
0	-1	0	
1	-1	∞	
2	-1	∞	
3	-1	∞	
.	-1	∞	
:		:	

By using minheap & adjacency matrix we can reduce its time complexity from $O(n^3)$ to $O((E+n)\log n)$

DIJKSTRA'S ALGORITHM :-
(shortest distance).

→ Visited
→ distance value.

Floyd-Warshall :-

Formula $A^k[i,j] = \min\{A^{k-1}[i,j], A^{k-1}[i,k] + A^{k-1}[k,j]\}$.

```
for(k=1 ; k<=n ; k++){
    for(i=1 ; i<=n ; i++)
        for(j=1 ; j<=n ; j++)
            A[i,j] = min(A[i,j], A[i,k]+A[k,j]);
    }
}.
```

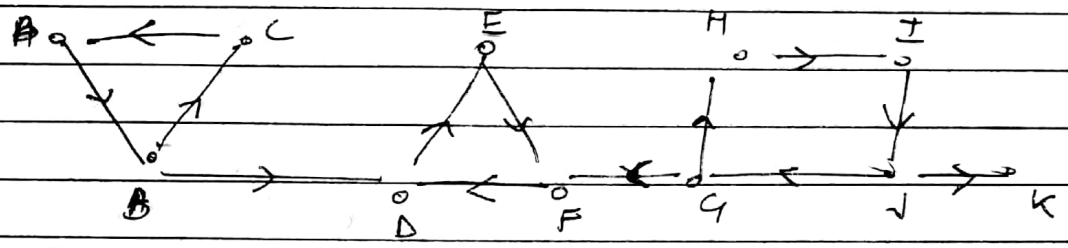
Scanned by CamScanner

Page No.
Date

Strongly connected components :-

-> All the vertices in that component is reachable from every other vertex in that component

eg :-



ABC DEF GHIJ

are strongly connected components

Kosaraju's Algorithm :- (BFS)

Stack → Set
by Finish. (visited)
Time.

Articulate Point :-
(cut vertex in a graph)

- If removing a vertex makes it an undirected graph then it is an articulate point.
- For a disconnected undirected graph it is the vertex removing which increases number of connected components.

1st → remove a vertex & then do BFS / DFS O(V x [V+E])
2nd → DFS Tree

Bridges : An edge in an undirected connected graph is a bridge if removing it disconnects the graph.

Scanned by CamScanner

Page No.
Date

→ Bi-connected Graph :-

A graph with vertex connectivity greater than 1./
A connected graph having no articulate vertex.

→ Bipartite Graph :-

A graph whose vertices can be divided into two independent sets, U & V such that every edge (u,v) either connects a vertex from U to V or a vertex from V to U.

A bipartite graph is possible if graph colouring is possible with 2 colours.

→ Colouring.

Scanned by CamScanner

Page No.
Date

Q 0/1 knapsack Problem :-

$V[i][w] = \max\{ V[i-1][w], V[i-1][w - w[i]] + p[i] \}$
(i) i j

if $j >= w[i]$

Q Minimum subset problem :-

→ similar to 0/1 knapsack :-

$i==0 \;||\; j==0 \quad dp[i][j] = 0$

$i >= a[i-1] \quad dp[i][j] = \max(dp[i-1][j], dp[i-1][j - a[i-1]] + a[i-1])$

$dp[i][j] = dp[i-1][j];$

* Next greater element (Find max/min in O(n)).

→ Using stack :-

1) Push the first element to stack.
2) Follow the steps in a loop :-
   i) Mark the current as next
   ii)

next min algo :-

```
for (int i=0; i<n; i++) {
    while (!s.empty() && arr[s.top()] >= arr[i])
        s.pop();
    if (!s.empty())
        ele[i] = s.top();
    s.push(i);
```

Scanned by CamScanner

Page No.
Date

TRIES :-

Tries vs hashmaps :-

1. We cannot search words starting with a few characters or it is difficult to implement.
2. Space optimization.

```
class TrieNode {
    char data;
    TrieNode **children;
    bool isTerminal
    TrieNode(char data) {
        this->data = data;
        children = new TrieNode*[26];
        isTerminal = False;
    }
};
```

↓ Initialize with null.

Types of TRIES :-

Scanned by CamScanner

Page No.
Date

Hamiltonian cycle :- (NP hard Problem).

→ If there is an Articulation point in a graph then Hamiltonian cycle is not possible.

→ Pendant vertex then also not possible.

```
Hamiltonian(k)
while(true){
    NextVertex(k);
    if(x[k]==0) return;
    if(k==n)
        print(x[1:n]);
    else
        Hamiltonian(k+1);
    }
}
NextVertex(k){
    while(true){
    x[k] = (x[k]+1) mod (n+1);
    if(x[k]==0) return
    if(G[x[k-1],x[k]]!=0){
        for(j=1 to k-1)
            if(x[j]==x[k]) break;
        if(j==k)
            if(k<n or (k==n) && G[x[n],x[1]]
                return;                    !=0)
    }
}
```

Scanned by CamScanner

Page No.
Date

GCD :-

→ Euclid Algorithm :-

$a > b$

$gcd(a,b) \Rightarrow gcd(b, a\%b)$;

$a\%b = a - b * [a/b]$

$r < b$

$+ \quad r \leq a-b$

$2r < a$

$a\%b < a/2$

time complexity = $\log(a)$

Extended euclid :-

```
class Triplet {
    public:
        int gcd;
        int x;
        int y;
};
Triplet gcd(a, b)
if (b==0){  myans.gcd = a;
            myans.x = 1;  myans.y = 0;
}
Triplet smallAns = gcdExtended(b, a%b);
Triplet myans;
myans.gcd = smallans.gcd;
myans.x = smallans.y;
myans.y = Smallans.x - (a/b) * Smallans.y;
return myans;
}
```

Scanned by CamScanner

Page No.
Date

Multiplicative Modulo Inverse :-

$(A \cdot B) \bmod m = 1 \qquad 1 \leq B \leq M-1$

$A \cdot B + m Q = 1$

Scanned by CamScanner

Page No.
Date

Dynamic Programming :-

① 0/1 knapsack.

```
if ( j < wt[i])
    dp[i][j] = dp[i-1][j];
else {
    dp[i][j] = max( val[i] + dp[i-1][j - wt[i]], dp[i-1][j])
}
```

② LONGEST COMMON SUBSEQUENCE :-

```
if (input1[i] == input2[j]) {
    T[i][j] = T[i-1][j-1] + 1
}else
    T[i][j] = max(T[i-1][j], T[i][j-1]);
```

③ Matrix chain Multiplication :-

	0	1	2	3
0	0	36	84	124
1		0	72	132
2			0	120
3				0

0	1	2	3
[2,3]	[3,6]	[6,4]	[4,5]

$T[i][j] = \min[dp[i][k] + dp[k+1][j] + dp[i] \times dp[k+1] \times dp[j+1],$
$dp[i][j])$

④ Subset sum Problem :-

```
if( j < input[i])  dp[i][j] = dp[i-1][j];
else  dp[i][j] = dp[i-1][j] || dp[i-1][j - input[i]];
```

Scanned by CamScanner

Page No.
Date

Q Coin Exchange problem :-

```
if i ≥ coin[j]
    T[i][j] = min[ T[i-1][j], 1 + T[i][j - coin[j]]
else
    T[i][j] = T[i-1][j].
```

Q Minimum Edit Distance :-

```
dp[i][0] = i ;
dp[0][i] = i ;

if ( str1[i] == str2[j] )
        dp[i][j] = dp[i-1][j-1];
else {
        dp[i][j] = min( dp[i-1][j], min( dp[i][j-1],
                                         dp[i-1][j-1] )
                                         +1 ;
}.
```

Q OPTIMAL BINARY SEARCH TREE :

dp[m][n] ;

0	1	2	3
10	12	16	21
4	2	6	3

	0	1	2	3
0	4	0		
1		2	10	
2			6	
3				3

Scanned by CamScanner

Page No.
Date

Q Longest palindromic subsequence :-

if (input[i] == input[j])
    T[i][j] = T[i+1][j-1] + 2
else
    T[i][j] = max(T[i-1][j], T[i][j-1]);

Q Coin Exchange :-

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

↓ without picking jth coin

↓ picking jth coin.

Q coin changing :- (ways)

if i ≥ coin[j]
    T[i][j] = T[i-1][j] + T[i][j - coin[j]]
else
    T[i][j] = T[i-1][j];

Q Rod cutting :-

Scanned by CamScanner