PMB

UNIT IV (Short Notes)

1. Linear search
2. Binary search
3. Bubble sort
4. Selection sort
5. Insertion sort
6. Denide concure
7. ~~Gr~~ Greedy method
8. Dynamic
9. Backtracking

Scanned by CamScanner

Example (i) First Point

Algorithm

(Linear search)

- A linear array DATA with 'N' elements & specific Item of Information are given. This Algorithm finds the location of Item in Array Data or sets LOC = 0

Scanned by CamScanner

1. [Initialize] set K=1 & LOC=0

2. Repeat step 3 & 4 while LOC=0 & K<N

3. IF Item = DATA[K] then set LOC: =K

4. Set K= K+1

5. [Sucussful?]

IF LOC =0 then:

Write: Item is not in the array Data

Else:

Write: LOC is the location of Item

[End of IF structure]

6. Exit

- The complexity of search Algorithm is given by the number C of comparisons betn Item & DATA [K]

- We seek C(n) for worst & averag- case.

① Worst case -

- Worst case occures when Item is the last element in the array DATA

Scanned by CamScanner

or is not there at all.

$$C(n) = n$$

$C(n) = n$ worst-case complexity of Linear search.

Average case

number of comparisions can be any numbers 1, 2, 3, ... n & each number occure probability $p = \frac{1}{n}$ the

$$C(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n}$$

$$(1 + 2 + 3 + \cdots + n) \cdot \frac{1}{n}$$

$$\frac{n(n+1)}{2} \cdot \frac{1}{n} = \frac{n+1}{2}$$

$$\frac{n+1}{2}$$

- Average case complexity approximetly half the no of element in the DATAlist

Scanned by CamScanner

* Binary Search

Imp

A → Array
n → Input size
Item → Item to be search

A

0	1	2	3	4	5	6	7	8	9	10	11
1	2	3	4	5	6	7	8	9	10	11	12

12	14	18	20	22	26	28
1	2	3	4	5	6	7
l	mid	h	mid			h

$$mid = \frac{1+7}{2} = \frac{l+h}{2} \quad 8/2 = 4$$

Item = 18 ?
Find

table

l	h	mid
1	7	4
1	3	2
3	3	3

Item < A[mid]

h = mid - 1
h = 4 - 1
h = 3

$$mid = \frac{1+3}{2} = 2$$

Scanned by CamScanner

A[mid] == Item

A[mid] < Item

$l = mid + 1$

$l = 3$

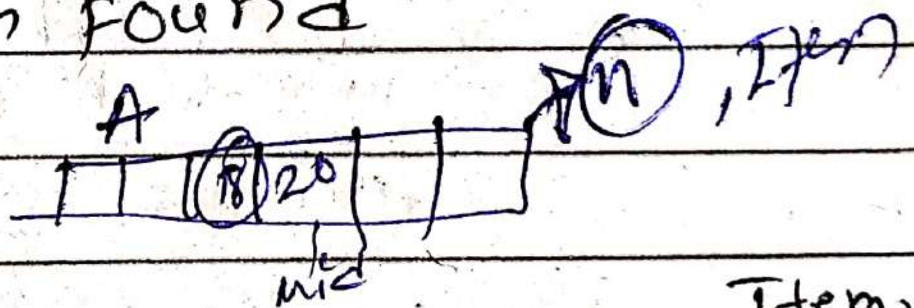
$l = 3 \quad h = 3 \quad mid = 3$

if (A[mid] == Item)

yes

Item found

Algorithm




```
int BinarySearch (A, n, Item)
{
        l = 1 , h = n
        while ( l ≤ h )
        {
```

Scanned by CamScanner

Page No.
Date

```
mid = ( (l+h) / 2 );
  if( Item == A[mid] )
      return mid;
  if( Item < A [mid] )
          h = mid-1;
  else
          l = mid+1;
}
return 0
}
```

Description

- Binary search also known as half interval search logarithmic search or binary chop.

- It devide the array in two two subarray.
- Binary search looks for perticular Item by comparing with middle of Array.
- If match found Index of Item is Returned.

- If middle Item is greater than Item Then Item is searched sub-array to left (before mid)

→ If middle Item is less than Item Then Item is searched sub-array to right

→ Repeat this process until at middle we get element

Scanned by CamScanner

* # Bubble sort

- "Bubble sort is simplest sorting Algorithm, that works by repeatedly swapping the Adjcent elements if they are in wrong order"

Example
- Array consists Following elements

n=5

A: | 5 | 1 | 4 | 2 | 8 |

Pass 1:

1	5	4	2	8

- compair first 2 element
5>1 swap & continue

1	4	5	2	8

(5>2) swap

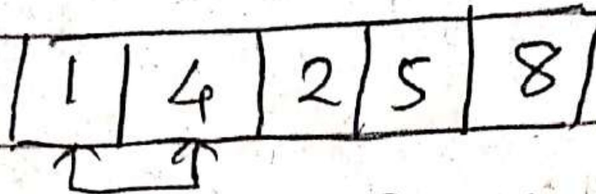
1	4	2	5	8

5 not greter than 8 so donot swap

1	4	2	5	8

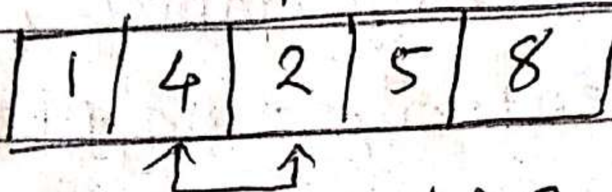
Scanned by CamScanner

Pass 2:

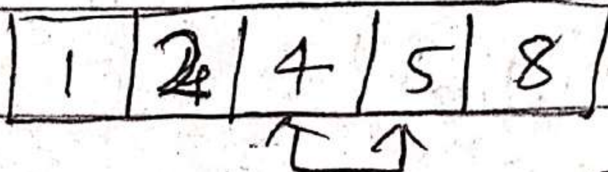


Repeat process For next Iteration

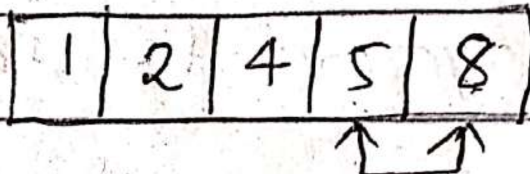
4 is sorted (no change)



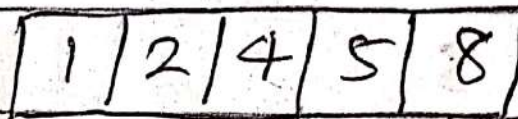
4 > 2 swap



5 is sorted (no change)

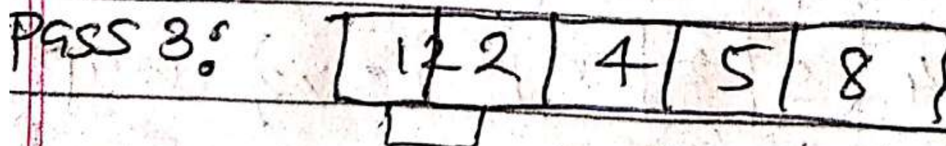


8 is sorted no change

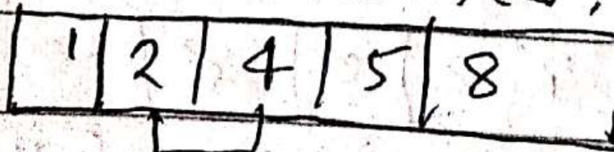


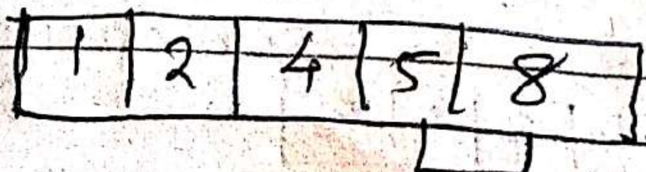
- our Array Alredy sorted but our Algorithm does not know. It require one more swap to know

Pass 3:



2 sorted, no change





Sorted

Scanned by CamScanner

* Selection Sort - min log / swap min, i (sorted & unsorted Array, passes)

Imp

- selection sort is simple sorting Algorithm
- sorting Algorithm is an in-place comparision based algorithm in which list is divided into two parts.
- sorted part at left end & unsorted part at right end.
- Initialy sorted part is empty & unsorted part is entire list.
- smallest element is selected from unsorted array & swapped with left most element.

Scanned by CamScanner

- that elements becomes part of sorted array.
- This process contineous moving unsorted array boundry one element to right.

Note -
- not suitable for large Dataset.

[Avg & worst complexity = $O(n^2)$]

Example

$n=6$

0	1	2	3	4	5
7	4	10	8	3	1

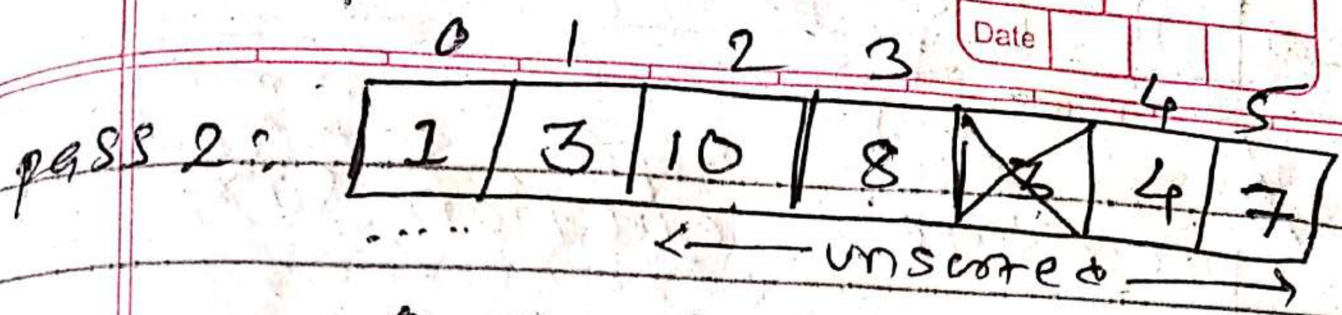
pass 1:

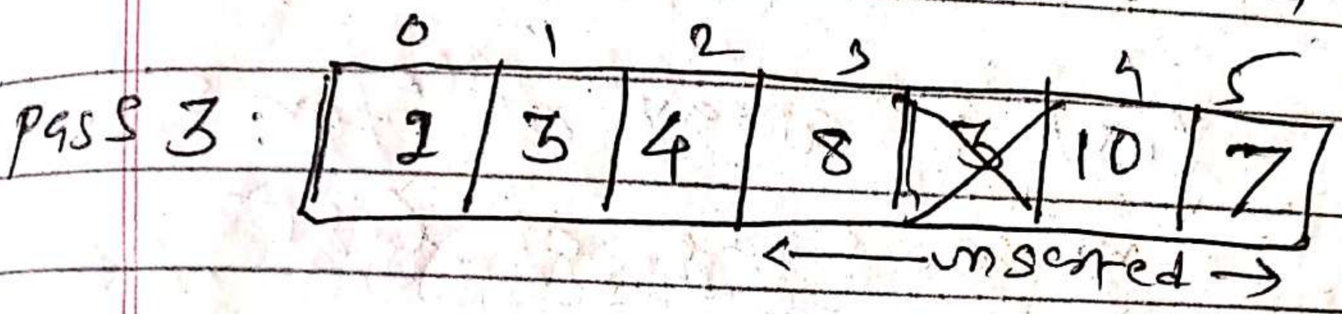
0	1	2	3	4	5
1	4	10	8	3	7

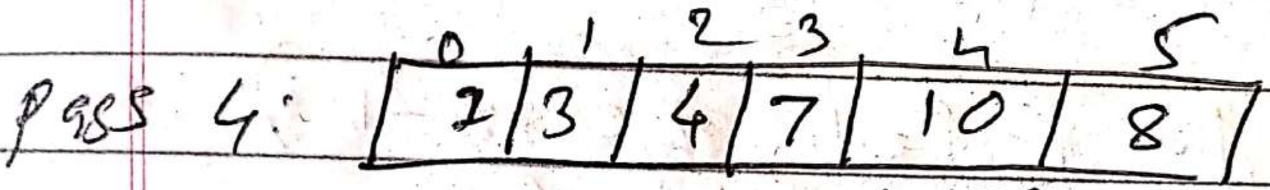
← sorted — unsorted

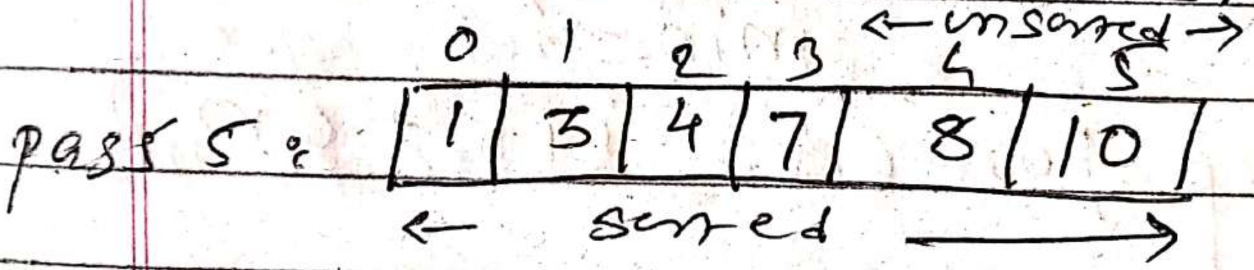
Find smallest element 1 & swap with 7 & shift counter to 1

Scanned by CamScanner









⇒ sorted At pass 5

(n-1) steps to sort selection sort.

⇒
- Find smallest ✓
- swap with leftmost element of unsorted array ✓
- Shift counter to +1 ✓

Scanned by CamScanner

procedure selection sort

list : array of items
n : size of list

```
for ( i=1 to n-1 )
    min = i ;
    for ( j = i+1 to n )
        if ( list[j] < list [min
            min = j ;
        end if
    end For
    if indexmin != i the
        swap list[min] &
             list [i]
    end IF
end For
```

Scanned by CamScanner

Page No.
Date

* Insertion sort

- Insertion sort works in the similar way as we sort cards in our hand in a card game.
- We assume, first card is alredy sorted.
- Then we select unsorted card or number
- unsorted number or card is greater than the card in hand or left side number It is placed on right
- otherwise (if right side number is less placed) left side.

& so on.

- " Insertion sort Algorithm insert unsorted element to proper place "

Scanned by CamScanner

Example

Suppose Array having elements

A :

15	10	20	5
1	2	3	4

n = 4

Pass 1 :

(15)	10	20	5

We assume first no 15 is sorted

Pass 2 :

10	15	(20)	5

- 10 is less than 15 so Insert 10 before 15

Pass 3 :

10	15	(20)	5

no change bcz 20 is greater than 15

Pass 4 :

5	10	15	20

∴ (5) is inserted to proper position i.e before 10

5 compeir with its left elements put proper place Acc. to sorting.

Scanned by CamScanner

Devide & conquer

↳ "Design or Approch to solve a problem"

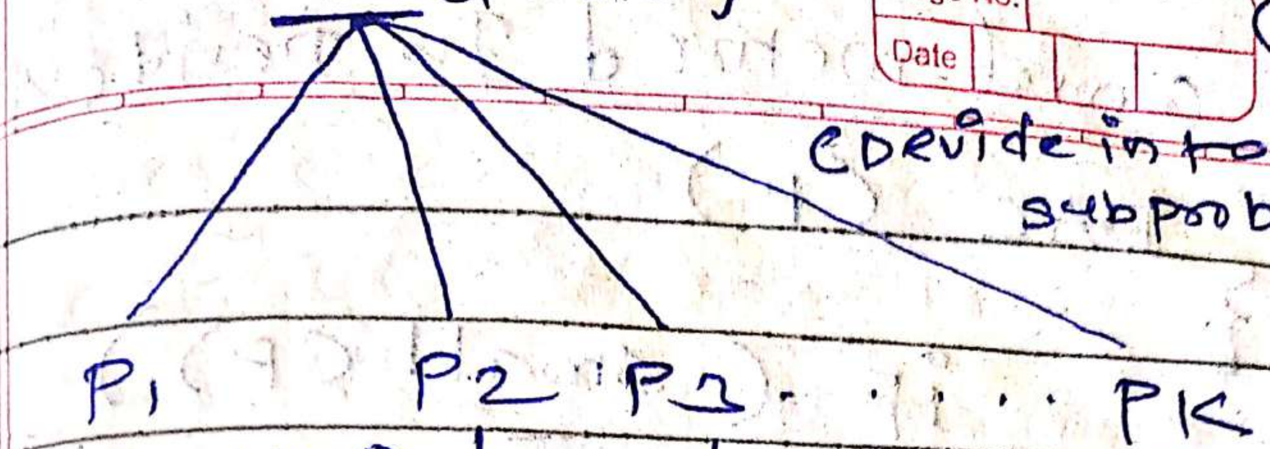
↳ solving computional problem

↳ problem is given size is 'n'

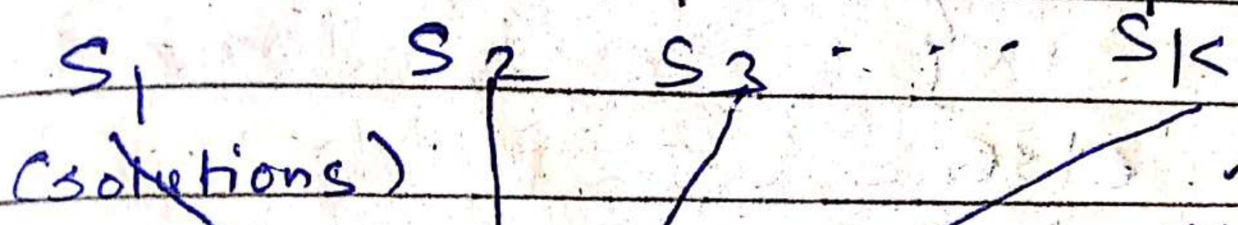
"problem is devided into subproblem & make solⁿ per subproblem & combine All subproblem solution to get original problem solⁿ"

Scanned by CamScanner

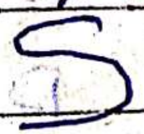
P (problem) size n (large)



(sub-problems)



(combine sol to get solution)



Note : sub problem is same as problem. P - sort & S. Prob - sort

Genral method for Devide & conquer

```
DAC (P)
{
    if ( small (P) )
    {
            S (P);
    }
    else
    {
        ① devide P into P1, P2, ... Pk
        ② Apply DAC(P1), DAC(P2), ...
                DAC(Pk)
        ③ Combine ( DAC(P1), DAC(P2)
                    .... )
    }
}
```

R-ecall

Scanned by CamScanner

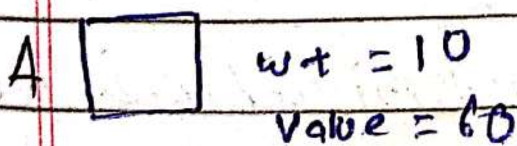
* Greedy Algorithm (Approch)

"Greedy is Algorithm paradigm that builds up sol^n pieace by pieace, always choosing the next peace that offers the ~~next~~ most obvious & immediate benefit."

- "so problems where choosing localy optimal also leads to global sol^n are best fit for Greedy"

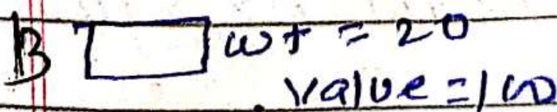
e.g. Fractional Knapsack

Take A, B & 2/3rd of C



full

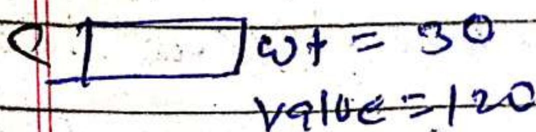
Total weight = 10 + 20 + 30 × 2/3 = 50



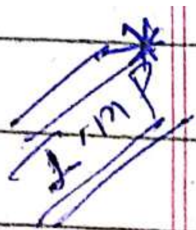
full

Capacity = 50

2/3 rd



Total value = 60 + 100 + 120 * 2/3 = 240



* Dynamic Method (programming)

" Dynamic programming is ~~plain~~ mainly optimization over [plain Recursion]".
- where we see Recursive solⁿ that has repeated calls for same Input, we can optimize using dynamic programming"

- "The idea is simply to store the result of ~~subprogram~~ subproblem so that we do not have to "re-compute" them when needed later"

- This simple optimization Reduces time complexities from exponential to polynomial"

e.g If we write simple Recursive solⁿ for Fibonacci Numbers, we get exponential time.

Scanned by CamScanner

complexity & if we optimize it by storing solⁿ of subproblems, time complexity reduces to linear"

[1]

```
int Fib(int n)
{
    if (n<=1)
        return n;
    return Fib(n-1) + Fib(n-2);
}
```

↳ Recursion: Exponential

```
F[0] = 0
F[1] = 1
For (i=2; i<=n; i++)
{
    F(i) = F(i-1) + F[i-2];
}
return F(n)
```

↳ Dynamic Prog : Linear

imp

Dynamic programming	Greedy method (Algorithm)
① Dynamic prog use to obtain optimal solution	① Greedy Algorithm Also used to get optimal sol^n
② Less efficient	② more efficient
③ Ex – 0/1 Knapsack	③ Fractional Knapsack
④ It guaranteed that Dynamic Prog. genrate optimal solution using Principle of optimality	④ There is no such guarantee.
⑤ Dynamic Prog. Follows Top Down Approch	⑤ Greedy method Follows bottom up Approch
⑥ we calculate optimal solution by considering previous steps e.g backtracking Recursion	⑥ we make best choice at moment. It lead global optimal solution
⑦ It does not work forward fashon (backward)	⑦ It works on Forward Fashon

Scanned by CamScanner

* Backtracking

— Backtracking is Algorithmic Technique for solving problem recursively, by trying to build solution incremently, one peace at a time, removing those solution that fail to satisfy the constraints at any point of time.

ie genral Algorithmic technique that consider searching every possible combination inorder to solve computational problem"

There are 3 types of problem in backtracking

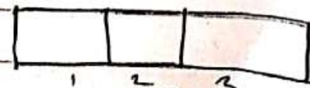
1. Decision problem – In this we search for feasible solution.
2. optimization problem – we search for best solution.
3. Enumeration problem – Find all feasible solutions

Example

– It is like conditional (constraint) probability

suppose two Boys & one Girl they have to arrange in 3 positions or chairs

B_1 B_2 G_1
boys Girls



positions

$n = 3$

Scanned by CamScanner

- We arrange them in 3! ways
i.e = 6 ways (All ways)
- But constraint is (Girl could not come in between two boys) (! in middle(G))
then tree is (only 4 paths)
i.e 4 ways only

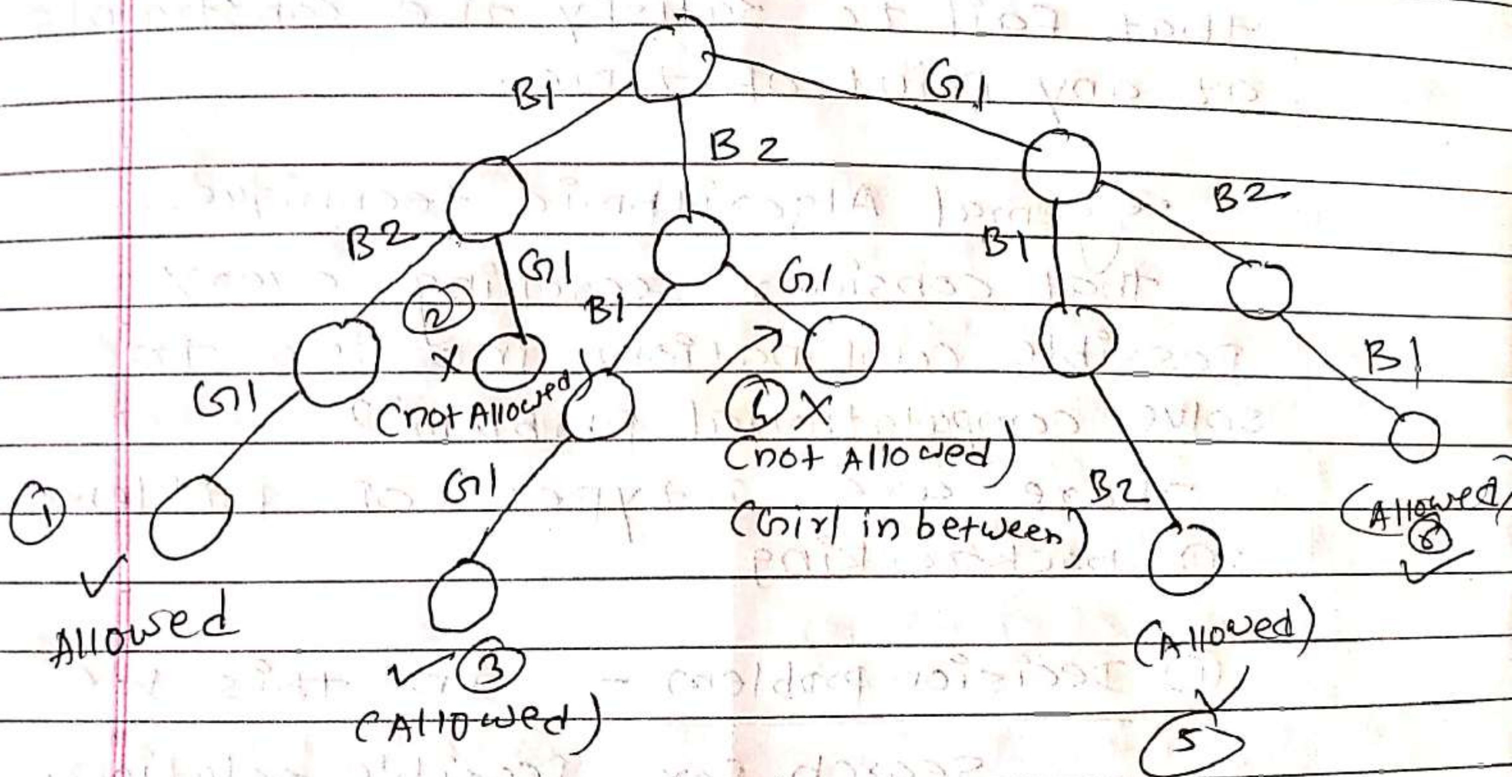


Fig: show total (6) ways but according to condition only
✓ only (4) Allowed.
✓ i.e Back tracking going previous & checking condition.

Scanned by CamScanner