2018ICSE0621
~~saikpa~~

Date
Page

Sai Ram. K
2018ICSE0621
08/01/2021

5 - CSE - 10.

Part - B

Q.1] **Quicksort :-**

Quicksort is a divide and conquer algorithm. It picks an element as pivot and partitions the given array around the pivot.

—> **ALGORITHM :-**

Step 1 :- Partitioning the array into 2.

```
partition (a, low, high)
{
    pivot = a[low], i = low+1, j = high
    while (pivot > a[i])
        i++
    while (pivot < a[j])
        j--
    if (i<j)
        swap (a[i], a[j])
    else
        swap (a[low], a[j])
    return j
}
```

Step 2 : Sorting the array

```
qsort (a, low, high)
{
    if (low < high)
    {
        j = partition (a, low, high)
```

20181CSE0621

Saifas

```
    qsort(a, low, j-1)
    qsort(a, j+1, high)
}
```

→ Efficiency of Quicksort :-

* Best & average case => $T(n)$

$$T(n) = \begin{cases} T(1) = 0, & \text{if } n = 1 \\ T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right) + n, & \text{otherwise} \end{cases}$$

$T\left(\frac{n}{2}\right)$: Time taken for left pivot; $T\left(\frac{n}{2}\right)$: Time for right pivot; n : Time for partitioning

$$T(n) = 2T\left(\frac{n}{2}\right) + n$$

Using Master's theorem,

$a = 2 \quad b^k = 2^1$

$\boxed{a = b^k} \qquad \boxed{k = 1}$

$$\therefore T(n) = \Theta(n^k \log n)$$
$$= \Theta(n^1 \log n)$$
$$\boxed{\therefore T(n) = \Theta(n \log n)}$$

Average & Best case have same complexity

$$\boxed{\therefore T(n) = \Omega(n \log n)}$$

* Worst case :-

$$T(n) = T(0) + T(n-1) + n$$

$$T(n) = \begin{cases} T(1) = 0, & \text{if } n = 1 \\ T(n-1) + n, & \text{otherwise} \end{cases}$$

We proceed further,

$$T(n) = T(n-1) + n \quad \text{---} (1)$$

Replace n by $n-1$ in (1)

Date
Page

$= T(n-1-1) + n-1+n$

$= T(n-2) + (n-1) + n$

$= T(n-3) + (n-2) + (n-1) + n$

$\vdots$

$= T(n-n) + (n-(n-1)) + (n-(n-2)) + n$

$= T(0) + (n-n+1) + (n-n+2) + n$

$= 0 + 1 + 2 + \dots + n$

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

We consider the highest power,

Hence, $\boxed{T(n) = O(n^2)}$

—> Differences between Merge Sort & Quick Sort.

	Quick Sort	Merge Sort.
(i)	The splitting of array is in any ratio not exactly into half.	Spliting of array is also not necessarily half but uses different approach.
(ii)	Worst case complexity is $O(n^2)$	Worst case complexity is $O(n \log n)$
(iii)	Works well on smaller arrays	Works good with any size.
(iv)	Works faster than other algorithms for small dataset.	It has consistent speed on any data.
(v)	It is not stable	It is stable
(vi)	Preferred for arrays	Preferred for Linked lists
(vii)	Method of sorting is internal	Method of sorting is external.

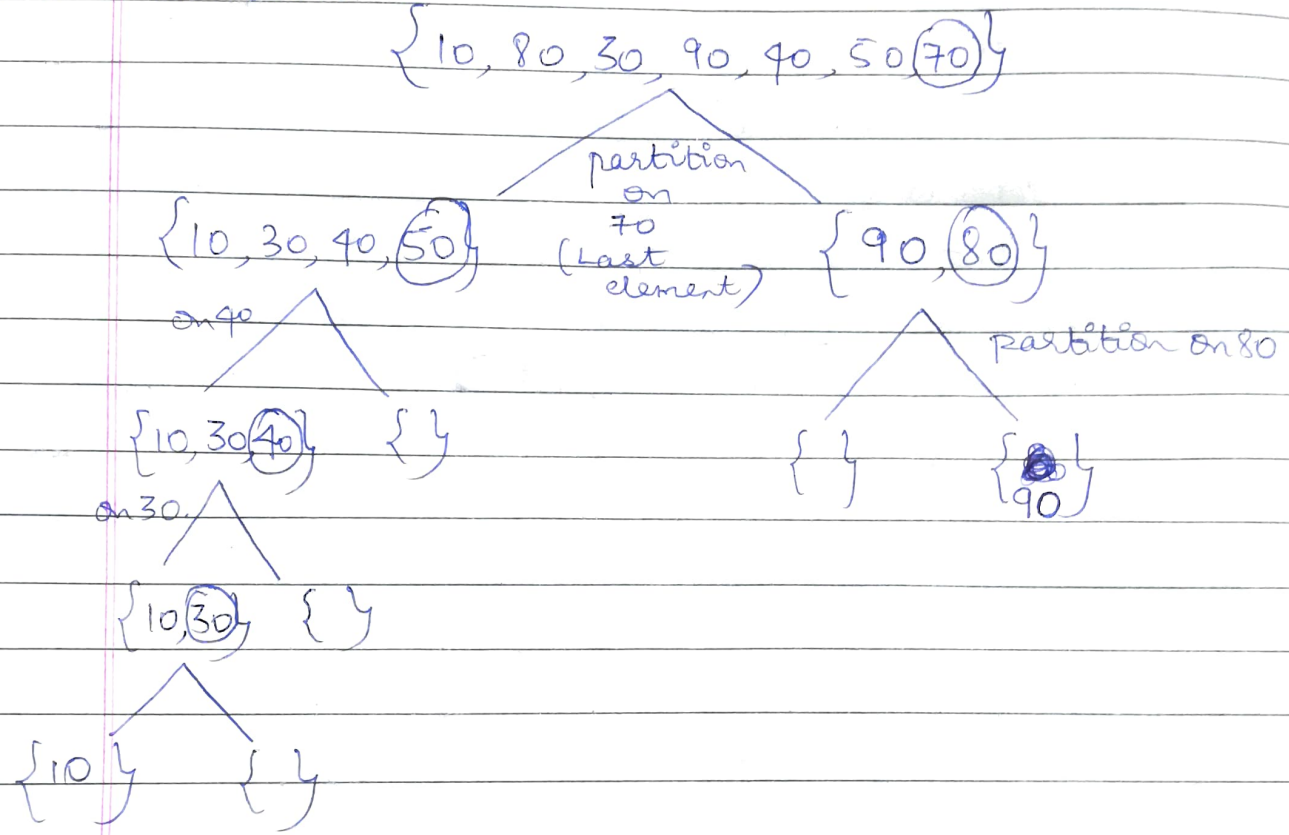
20181CSE0621

Sai

—> Example for quicksort:

Consider { {10, 80, 30, 90, 40, 50, 70} }

Step1:- Partition:



Step2:- Merging Back we get:-

{10, 30, 40, 50, 70, 80, 90}

[Merging is done by traversing.]

20181CSE0621

Date
Page

Sai Ram. K 5-CSE-10

20181CSE0621

08/01/21

Part-B

Q.2] Backtracking:-

Backtracking can be defined as an algorithmic technique for solving problems recursively by trying to build a solution one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time.

—> ALGORITHM OF N-QUEENS:-

Backtrack (X[1...i])

//Gives a template of generic backtracking algorithm.

//Input: X[1i] specifies first i promising components of a solution.

//Output: All tuples representing the problem's solution.

if X[1i] is a solution write X[1...i]

else

for each element x ∈ S;+1 consistent with X[1i] and the constraints do

X[i+1] * -X

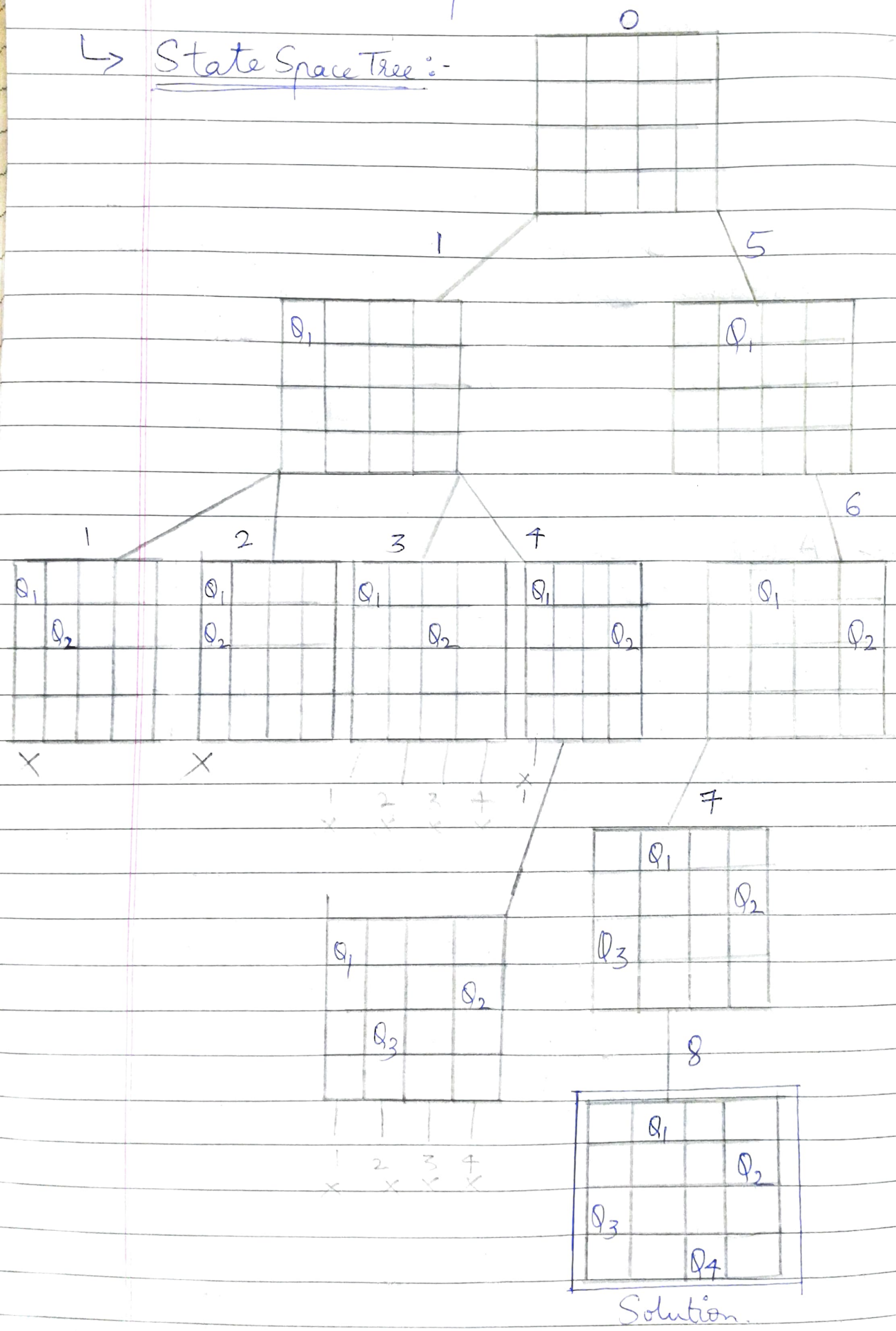
Backtrack (X[1...i+1])

2018ICSE0621

Saish

→ 4*4 queens problem:

↳ State Space Tree :-



20181CSE0621
Sairam

Date
Page

Sai Ram K
20181CSE0621
08/01/2021

5-CSE-10

Part - B

Q.3] Knapsack :-

↳ Algorithm :-

// Input : n is total number of items; W is the capacity w[ ] stores weights of each item & v[ ] stores values.

// Output : Returns total value of selected items from knapsack.

```
for (i ← 0 to n) do {
for (j ← 0 to w) do {
 table [i,0] = 0
 table [0,j] = 0 }}

for (i ← 0 to n) do {
for (j ← 0 to w) do {
  if (j < w[i]) then {
        table[i,j] ← table[i-1, j] }
else if (j >= w[i]) then
   table[i,j] ← max[table(i-1,j), (v[i+j) +
                                table[i-1,j-1])}}}

   return table[n,w]
```

Time efficiency = O(nw)

20181CSE0621

Sai

→ Given, n=4, w=[1,2,2,3] P=[18,16,6,4]
capacity = 4.

P	w
18	1
16	2
6	2
4	3

	0	1	2	3	4
0	0	0	0	0	0
1	0	18	18	18	18
2	0	18	18	34	34
3	0	18	18	34	34
4	0	18	18	34	34

Formula: $V[i,w] = \max\{V(i-1,w), V(i-1, w-w(i)) + P[i]\}$

$V[1,1] = \max\{V(0,1), V(0,1-1)+18\} = 18$
$V[1,2] = \max\{V(0,2), V(0,2-1)+18\} = 18$
$V[1,3] = \max\{V[0,3], V(0,3-1)+18\} = 18$
$V[1,4] = \max\{V[0,4], V[0,4-1]+18\} = 18.$
$V[2,1] = \max\{V[1,1], V[1,1-2]+16 = 18.$
$V[2,2] = \max\{V[1,2], V[1,2-2]+16\} = 18.$
$V[2,3] = \max\{V(1,3), V(1,3-2)+16\} = 34$
$V[2,4] = \max\{V(1,4), V(1,4-2)+16\} = 34$
$V[3,1] = \max\{V(2,1), V(2,1-2]+6\} = 18.$
$V[3,2] = \max\{V(2,2), V(2,2-2)+6\}$
$= \max(18,6) = 18.$
$V[3,3] = \max\{V(2,3), V(2,3-2)+6\} = \max(34,24) = 34$
$V[3,4] = \max\{V(2,4), V(2,4-2)+6\} = \max(34,18+6) = 34$
$V[4,1] = \max\{V(3,1), V(3,1-2)+4\} = \max(18,-) = 18.$
$V[4,2] = \max\{V(3,2), V(3,2-2)+4\} = \max(18,4) = 18$
$V[4,3] = \max\{V(3,3), V(3,3-2)+4\} = \max(34,22) = 34$
$V[4,4] = \max\{V(3,4), V(3,4-2)+4\}$
$= \max\{34,22\} = 34$

20181CSE0621
Saif

Date
Page

Hence Max value is 34
As table[4,4] = table[3,4]
& table[3,4] = table[2,4]
we dont select 4th & 3rd item.

Selecting item 1 & 2 :-
∴ Solution will be (1,1,0,0)

Profits: 34 - 16 = 18 —> 2nd item
18 - 18 = 0 —> 1st item.

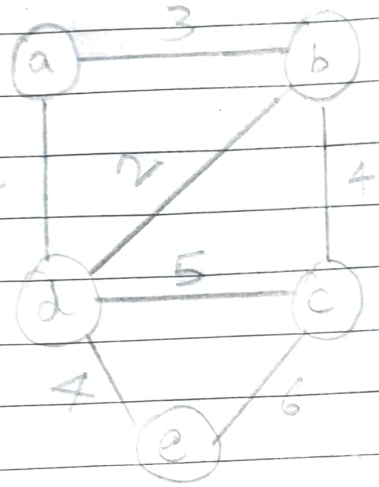
20181CSE0621

Lab-6

Sai Ram. K.
20181CSE0621
08/01/2021

Part-C

Q.1] Graph:-



↳ Dijkstra's Algorithm:-

Let start node & node x's distance be x.

Step1: Assign to every node a distance value & set it to 0 for initial node & ∞ to other nod

Step2: Set initial node as current & mark other nodes as unvisited.

Step3: Consider current node & calculate distance of neighbors.

Step5: Set unvisited nodes with smallest distance from initial node as next & continue.

Step6: When all nodes are visited the algorithm ends.

20181CSE0621

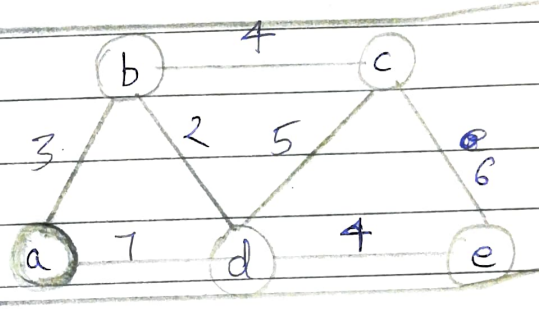
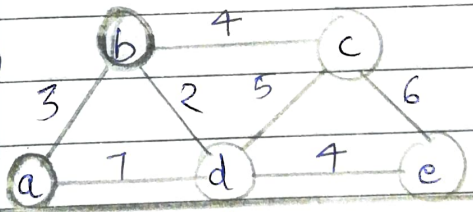
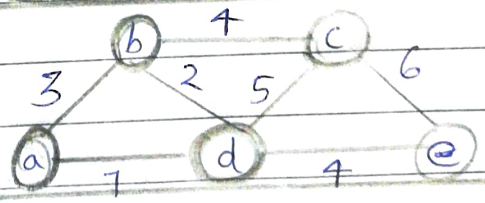
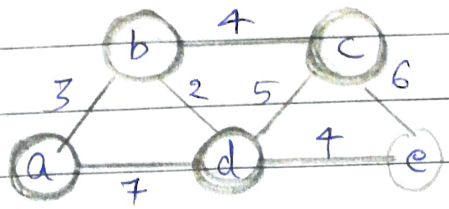
Saifa

Date
Page

Algorithm :-

Let s be source node
dist[s] ← 0
for all v ∈ V - {s} do
dist[v] ← ∞
S ← ϕ
Q ← V
while (Q ≠ ϕ) do
u ← mindist(Q, dist)
S ← S ∪ {u}
for all v ∈ neighbors[u] do
if (dist[v] > dist(u) + cost[u][v])
then dist[v] = dist[u] + cost[u][v]
return dist.

→ Finding shortest path :-

Tree Vertices	Remaining Vertices	
a(-,0)	b(a,3), c(-,∞) d(a,7), e(-,∞)	
b(a,3)	c(b,3+4), d(b,3+2) e(-,∞)	
d(b,5)	c(b,7) e(d,5+4)	
e(d,9)		

Hence, Shortest path is a -> b -> d -> c

2018ICSE0621
Sairam

Date
Page

Sai Ram. K
2018ICSE0621
08/01/2021

S-CSE-10

Part-C.

Q.2] Pseudocode for Backtracking:
Its of 2 types:-
(1) Recursive Backtracking solution
(2) Finding whether solution exists or not.

(1) Recursive:

```
void findsoln(n, other params):
    if(found a solution):
        solnfound = solnfound + 1;
        displaysoln();
    if(solnfound >= solntarget):
        system.exit(0);
    return
    for(val = first to last):
        if(isvalid(val, n)):
            applyvalue(val, n);
            findsoln(n+1, otherparams);
            removeValue(val, n);
```

(2) Finding if solution exists:

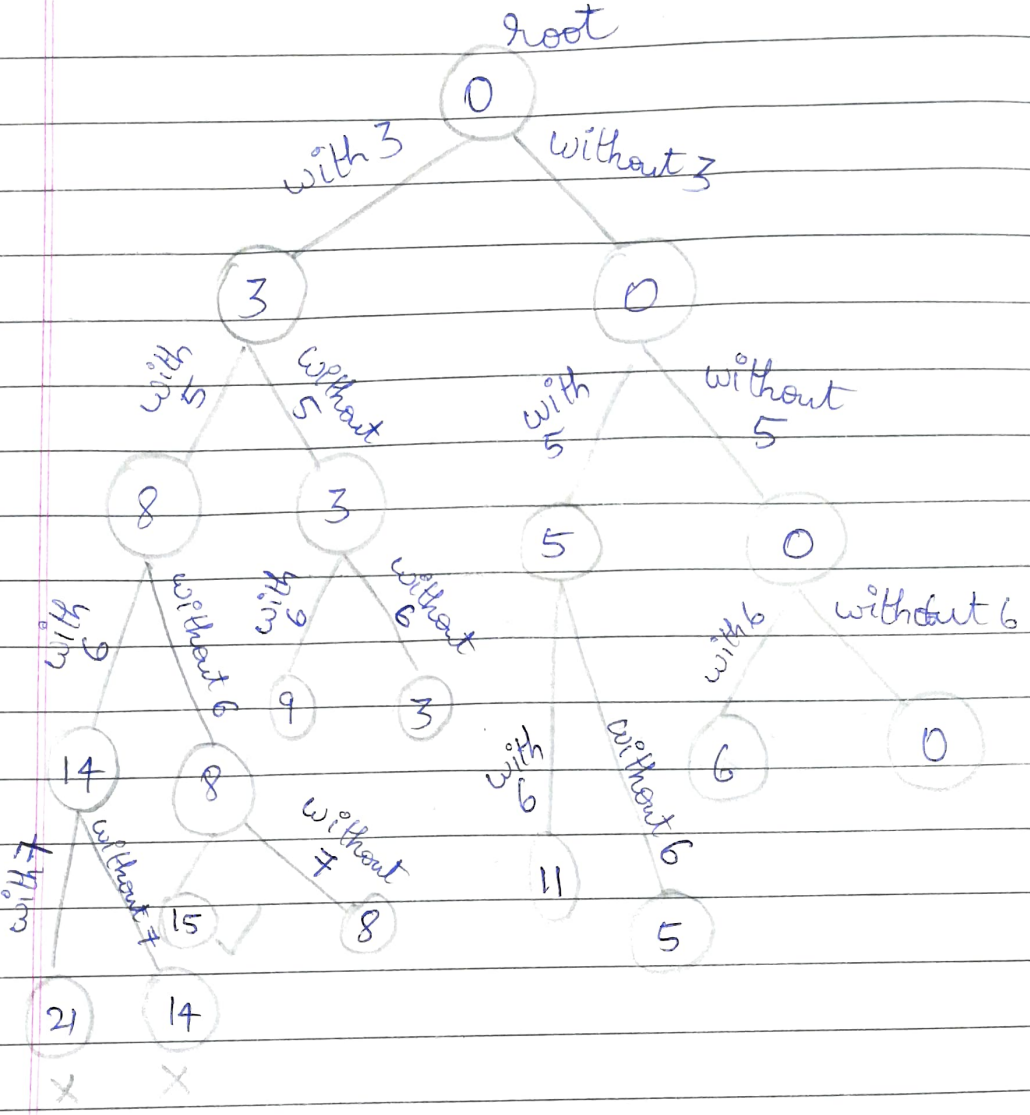
```
boolean findsoln(n, otherparams):
    if(found a solution):
        displaysolution();
    return True;
    for(val = first to last):
        if(isvalid(val, n)):
            if(findsoln(n+1, otherparams))
                return True;
            removevalue(val, n);
    return False;
```

2018ICSE0621

Saiteja

→ Given: $W = \{3, 5, 6, 7\}$

$m = 15$.



Solution:- With 3 ✓

With 5 ✓

Without 6

With 7 ✓

∴ 3 + 5 + 7 = 15