1) Discuss the optimal Binary search Tree (OBST) and construct for 4 nodes (n=4) with the keys {a1, a2, a3, a4} = {do, if, int, while}. Let P(1:4) = (3, 3, 1, 1) and q(0:4) = (2, 3, 1, 1, 1)

Ans

This minimizes the total cost of searching for keys. nodes are arranged so that the expected cost of searching is minimized based on the access of each

The expected cost of Searching an OBST is given by:-

$$E = \sum_{i=1}^{n} (P_i \cdot depth(a_i)) + \sum_{i=0}^{n} (q_i \cdot depth(d_i))$$

where

- P_i is the probability of accessing key a_i
- q_i Probability of unsuccessful search between nodes
- d_i is the dummy node between keys a_i and a_{i+1}

Steps for constructing the OBST:-

1) Initialize the cost matrices:-
   - cost matrix C[i][j]
   - Root matrix R[i][j]

2) Base cases:- Initalize the diagonal elements in the cost matrix

3) Build OSBT by filling the matrices:-
   - Calculate the cost of each subtree by trying every node a_k
   - The root that gives the minimum cost of the subtree R[i][j]

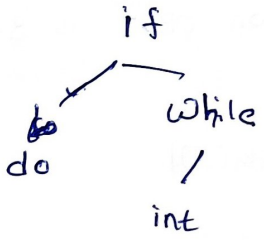
4) calculate the total cost∑ After filling cost Matrix. The Minimum cost OBST can be found in c[1][n].

Ex:-

Keys :- {a1, a2, a3, a4} = {do, if, int, while}

Probabilities of successful search P[1:4) = (3, 3, 1, 1)

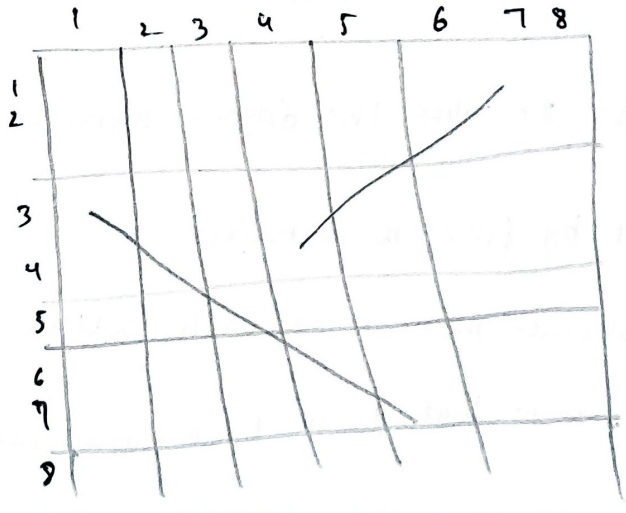
Probabilities of unsuccessful search q(0:4) = (2, 3, 1, 1, 1)



2) Write an algorithm for N-queen's problem and Provide the time and Space Complexity for 8-queen's problem

Ans:-

- We try to find all ways to place "n" not attacking queens in a chess board
- We represent solution as N-Tuple $(x_1, x_2, \ldots x_n)$ where x_i is the column of i^{th} row
- All x_i's are distinct, so our solution space is reduced from n^n to $n!$ n-Tuples



In the above diagonal (3,1) and (7,5) is normal diagonal where as (1,7) & (4,4) is opposite diagonal.

In normal diagonals Difference Between Row and Column is same

if (i,j) & (k,l) are two cells then

in normal Diagonals i-j = k-l

in opposite Diagonals i+j = k+l

which implies |j-l| = |i-k|

Algorithm:

1) For the place;

```
1) Algorithm place(k,i)
2) //Returns true if a queen can be placed in kth row and ith column. otherwise it returns
3) // false. x[] is a global array whose first (k-1) values are been set
4) //abs(r) returns the absolute value of r
5) {
6)   for j:=1 to k-1 do
7)       if((x[j]=i) or (Abs(x[j]-i)=Abs(j-k)))
8)                 then return false;
9)   return true;
10) }
```

2) Algorithm N-Queen is developed using the concept of Back tracking

```
1) Algorithm N queen(k,n)
2) //using Backtracking, this procedure prints all
3) //Possible placement of n queens on an nxn
```

4) n Chess board so that they are non attacking

```
5) {
6)   for i:=1 to n do
7)   {
8)     if place(k,i) then
9)     {
10)      x[k]:=i;
11)      if (k=n) then write (x[1:n]);
12)      else Nqueens (k+1, n);
13)    }
14)
15)  }
```

Time complexity: $O(N!)$

Space complexity: $O(N^2)$

3) Find all Possible Subsets of the set w that sum to m. Let w = {5,7,10,12,15, 18,20} and m = 35 and draw the Portion of the state space tree Generated useing back tracking

Ans:

1) sort the set : Helps to Optimize the backtracking process by stopping early when sum reaches m

2) Recursive Backtracking:

- use a recursive function that explores all subsets
- start from the first element

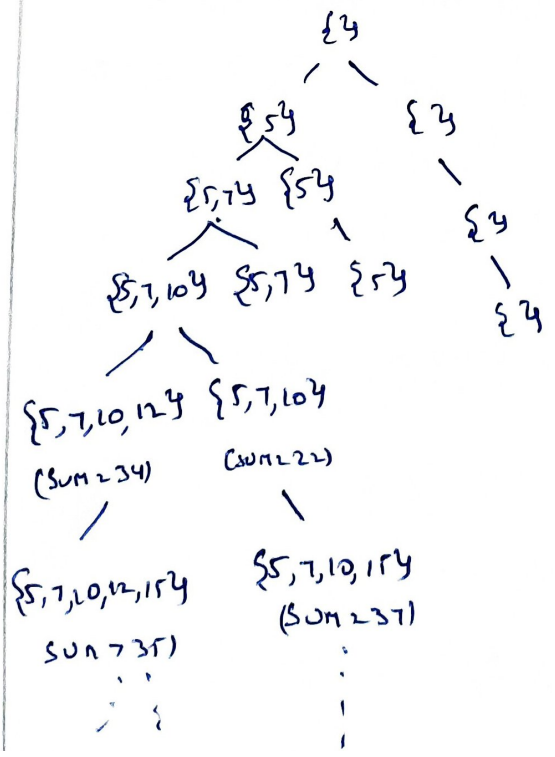
3) State space tree:

- We continue exploring down the tree until we either find a subset that sums to m or backtrack when adding

Algorithm:

```
Algorithm SumofSub(S, k, r)
{
    x[k]:=1
    if (s+w[k]=m) then write(1:k);
    else if (s+w[k]+w[k+1] ≤ m) then SumofSub(s+w[k], k+1, r-w[k]);
    if ((s+r-w[k]≥m) and (s+w[k+1] ≤ m)) then
    {
        x[k]:=0;
        SumofSub(s, k+1, r-w[k]);
    }
}
```

State space tree:



The subsets are:

1) {5, 10, 20}
2) {7, 10, 18}
3) {5, 12, 18}

These are all possible subsets of w that sum to 35.

4) Write a non-deterministic algorithm for sorting an array

Ans: In an non-deterministic algorithm for sorting an array, we use a hypothetical non-deterministic machine than can "guess" the correct an order of elements and verify if the order is sorted..

Algorithm:

1) This guess an arrangement A' of the elements A
2) Verify if A' is sorted in non-decreasing order
3) if A' is sorted, then output A' as the sorted Array.
4) if A' is not sorted, go back to step-1 and guess a new arrangement

```
Algorithm NonDeterministicsort(A);

        if A' is sorted in non-decreasing order;
            return A'
        else
            fail
```

Time complexity : $O(n)$