Date: / /
Page:

# Day - 86

## Subsequence

| 1 | 2 | 3 |
|---|---|---|

=) Subsequence means printing all their combination i.e.

{{1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}, {3}}

while maintaining the order.

=) Here, we are printing any no. as —

Ex: we are printing 1 —

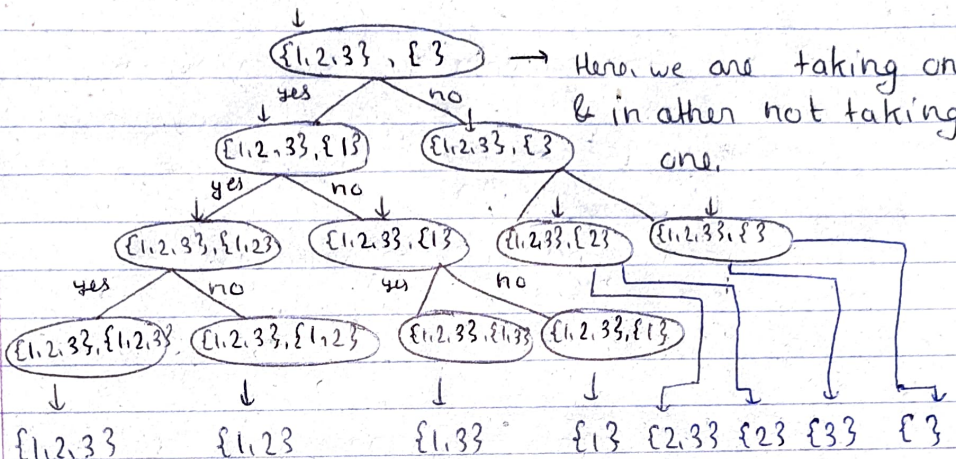
that means we are taking 1 & discarding other two.

=) So, we can represent all the combination as —

( 1 2 3 )

0 0 0 → {}
0 0 1 → {3}
0 1 0 → {2}
0 1 1 → {2,3}
1 0 0 → {1}
1 0 1 → {1,3}
1 1 0 → {1,2}
1 1 1 → {1,2,3}

=) So, here we will use recursion.




→ Here, we are taking one & in other not taking one.

⇒ Here, we are asking every element to come in or not.

Code

```
void subseq(int arr[], int index, int n,
            vector<vector<int>> &ans, vector<int> temp){
    if(index == n){
        ans.push_back(temp);
        return;
    }
    subseq(arr, index+1, n, ans, temp);
    temp.push_back(arr[index]);
    subseq(arr, index+1, n, ans, temp);
}
int main(){
    int arr[] = {1,2,3};
    vector<vector<int>> ans; → n
    vector<int> temp; → n
    subseq(arr, 0, n, ans, temp);
    // Print
}
```

Date : / /
Page :

⇒ So, the time complexity is equal to the no. of nodes in the recursion tree.

⇒ Nodes = 1S = $2^{n+1} - 1 = 2^n$

T.C. = $O(2^n)$

So, total space complexity is —

$O(2^n * n + n^2 *)$

⇒ $O(2^n * n)$

⇒ For more optimizing space, we can also pass the temp array as reference.

Then S.C. → O(n).

* Print the subseq. of "abc"

⇒ Do the same procedure as we do in the last question.

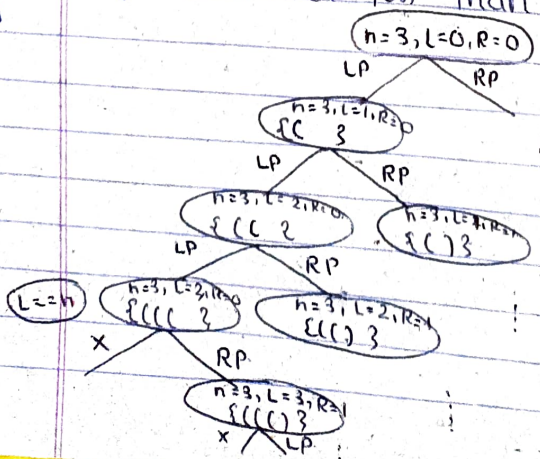
* Generate Parentheses:

⇒ we have to generate valid parenthesis.

n = 2, → ()() , (())

⇒ Here, we can see that right parenthesis is always equal to or less than left parenthesis.

⇒



```
void parenthesis( int n, int left, int right ,
  vector <string> &ans , string &temp ){
    if( left + Right == 2*n){
          ans.push_back(temp);
          return;
    }
    if( left < n){
        temp.push_back('c');
        parenthesis (n, left+1, right, ans, temp);
        temp.pop_back();
    }
    if( right < left){
        temp.push_back (')');
        parenthesis (n, left, right+1, ans, temp);
        temp.pop_back();
    }
}
```