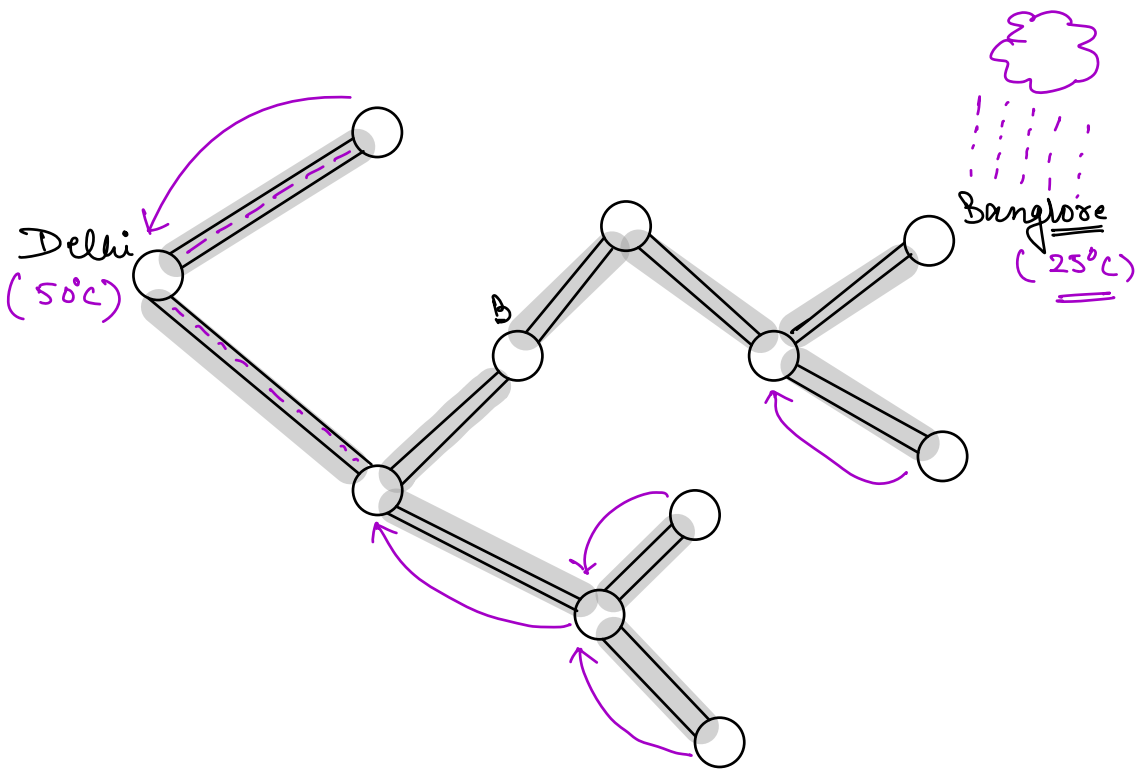
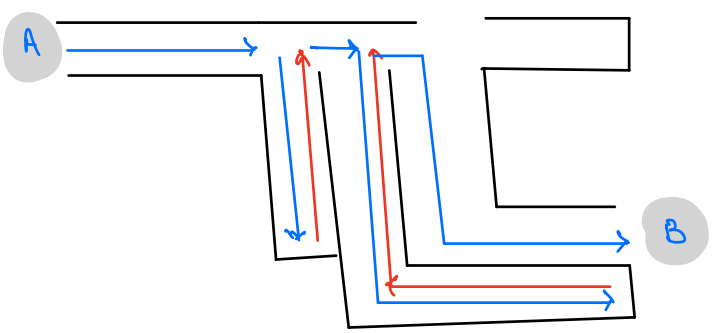
Backtracking :-

↳ Generating all possibilities to get the correct ans.





⇒ Trying all the possibilities

↳ Brute Force.

Backtracking ⇒ recursion + retrace.

Q.1 Print all N digit no's using only {1 & 2}

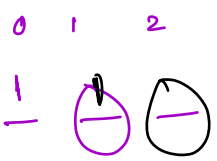
N = 1 ⇒ 1
2

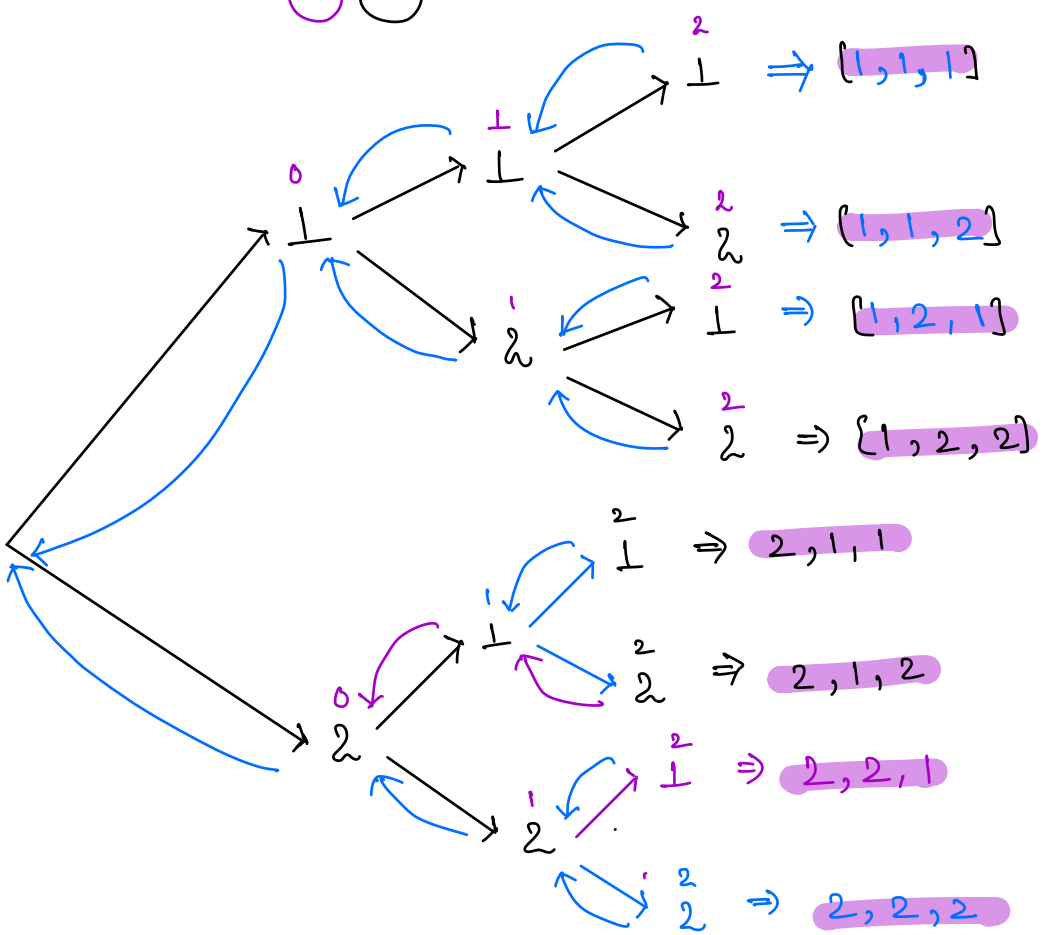
N = 2

1 1
1 2
2 1
2 2

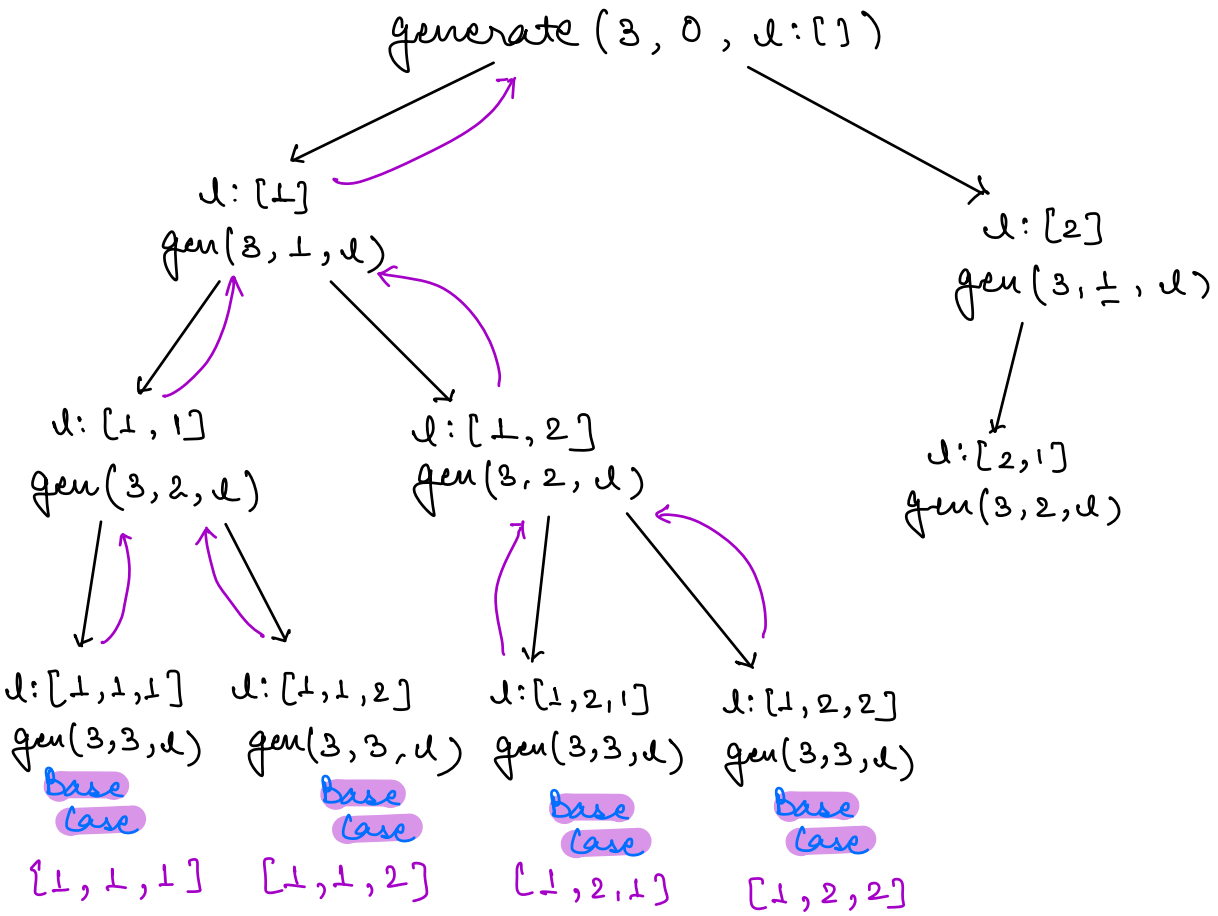
N = 3

[1, 1, 1]
[1, 1, 2]
[1, 2, 1]
[1, 2, 2]
[2, 1, 1]
[2, 1, 2]
[2, 2, 1]
[2, 2, 2]





```
void generate(N, inden, currlist){
  if(inden == N){
      print(currlist); ⇒ O(N)
      return;
  }
  currList[inden] = 1
  generate(N, inden+1, currlist);
  currList[inden] = 2
  generate(N, inden+1, currlist);
}
```



No. of fun^ calls = 2^N

TC of each fun^ call ⇒ $O(N)$

TC : $O(N \cdot 2^N)$

~~Void~~ List<List<int>> generate (N, inden, currlist) {

```
if (inden == N) {
    ans.add(currlist);
    return ;
}
currList[inden] = 1
generate(N, inden+1, currlist);
currList[inden] = 2
generate(N, inden+1, currlist);
```

}

↳ Clone the list & then add. ⇒ O(N)

Note: Shallow copy vs Deep copy ✓

ans: [[1,1,2], [1,1,2], . . .

TC : $O(N \cdot 2^N)$

Q: Print all N digits no's (as lists) using {1, 2, 3, 4 & 5}

```
Void generate (N, inden, currlist) {
    if (inden == N) {
        print(currList);
        return;
    }
    currList[inden] = 1
    generate (N, inden+1, currlist);
    currList[inden] = 2
    generate (N, inden+1, currlist);
    currList[inden] = 3
    generate (N, inden+1, currlist);
    currList[inden] = 4
    generate (N, inden+1, currlist);
    currList[inden] = 5
    generate (N, inden+1, currlist);
}

for(i = 1; i <= 5; i++) {
    currList[inden] = i;
    generate (N, inden+1, currlist);
}
```

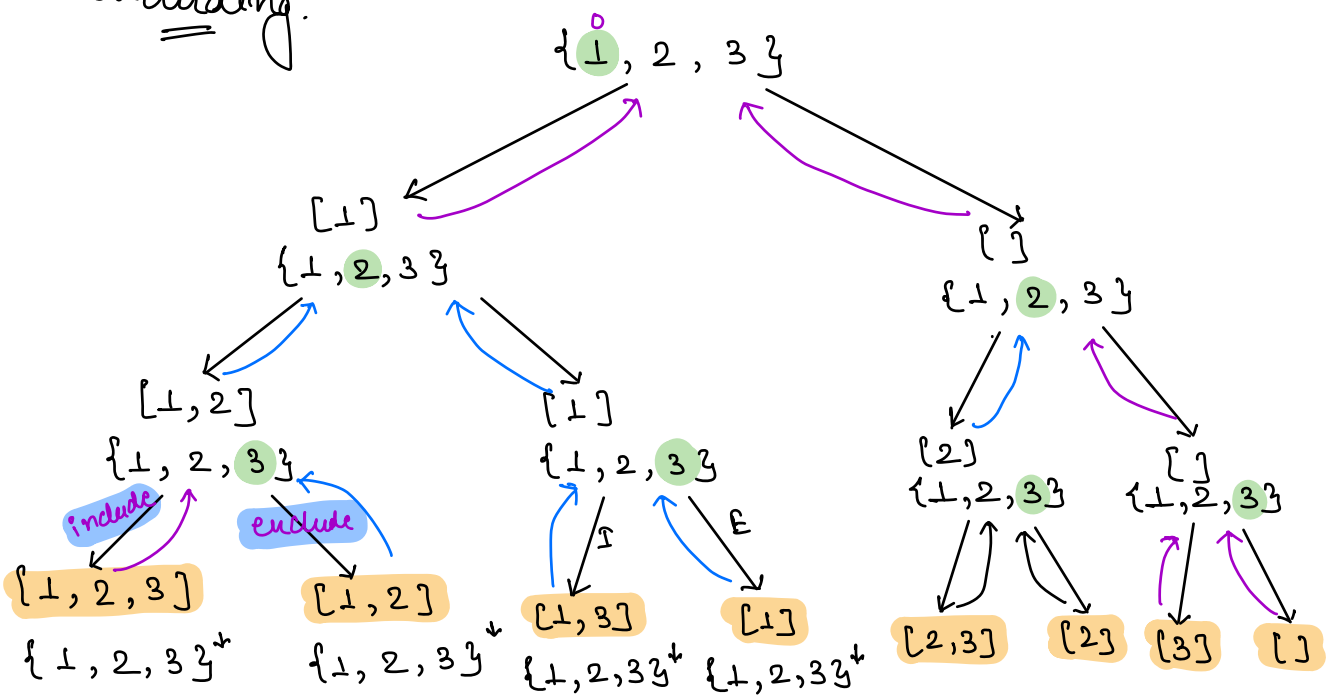
TC : $O(N \cdot 5^N)$

Q. Given an Array of size N, Generate all the subsets of it.

Amazon | MS | GS | Adobe | ....

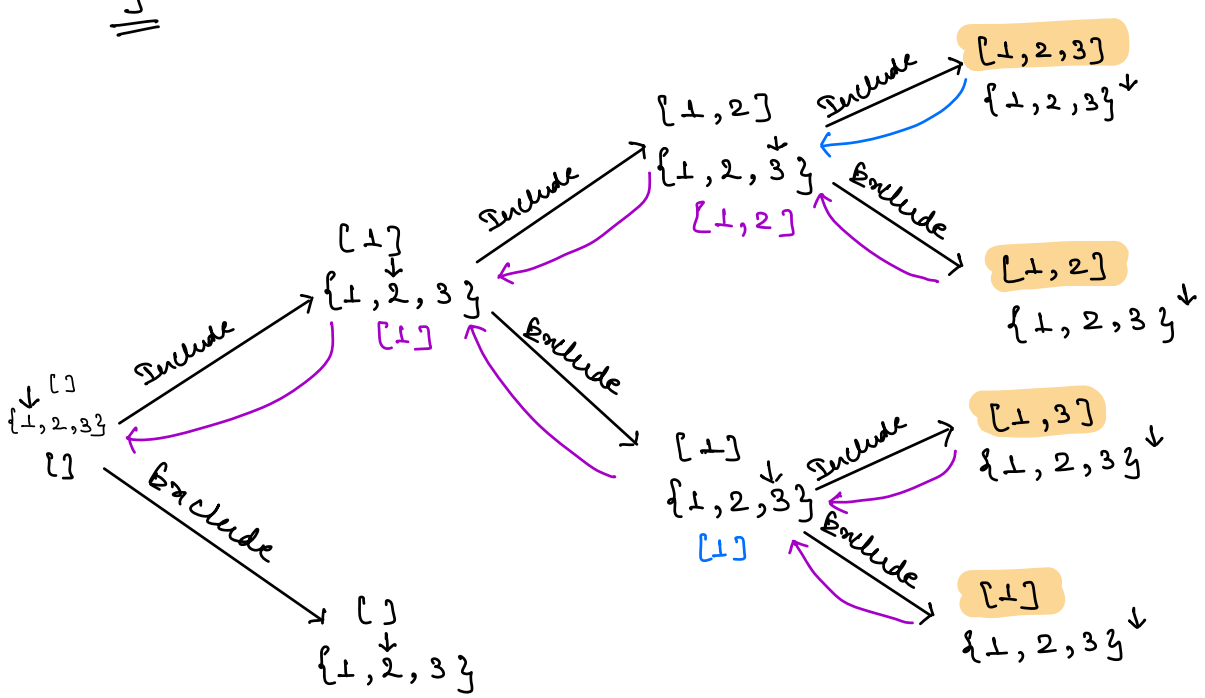
A : [1, 2, 3] →
[[ ],
[1]
[1,2]
[1,3]
[1,2,3]
[2]
[2,3]
[3]]

* Subsets / Subsequences : for every element, we have 2 choices, either to include it or to excluding.



```
generateAllSubsets(currList, indem, A[]) {
    if(indem == N) {
        ans.add(Deep copy | Clone the currlist);
        return;
    }

    // Include A[indem] in subset
    currList.add(A[indem]);
    generateAllSubsets(currList, indem+1, A);
    // Delete the last added element (A[indem])
    // from the currList
    // Exclude A[indem] in the subset.
    currList.pop()
    generateAllSubsets(currList, indem+1, A);
}
```



TC: $O(N \cdot 2^N)$, SC: Recursion Stack + currlist : $O(N)$
$O(N) + O(N)$

Note for backtracking problems, constraint will be very small.

$N <= 20$.

— * —