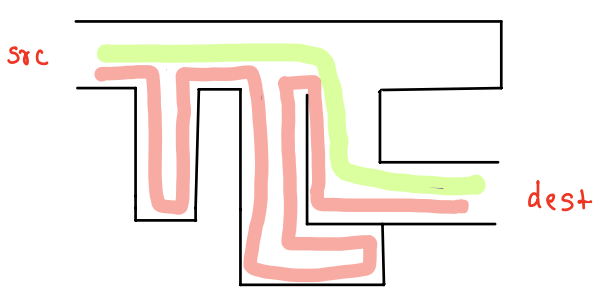
Agenda

1) Introduction to Backtracking
2) All numbers (using 1 & 2)
3) Subset Sum
4) N-Queens

Introduction

Exploring all paths with the help of recursion.

Maze :



→ In Backtracking the coming back step is very important.

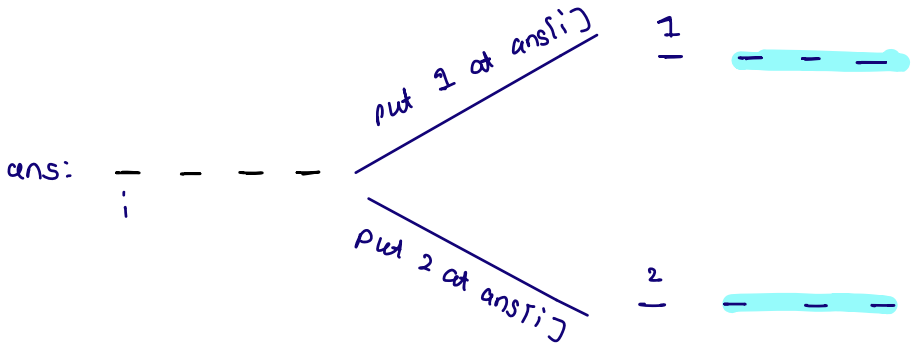
Q.1 Given N. Print all N digit numbers formed by 1 & 2 in increasing order of numbers.

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 solve (int N) {
    int [] ans = new int [N];
    helper(N, ans, 0);
}
```

```
void helper (int N, int [] ans, int idx) {
    if (i == N) {
        // print the array
        for (int val : ans) {
            System.out.print (val + " ");
        }
        System.out.println ();
        return;
    }

    // two options

    ans[i] = 1;          // put 1 at ans[i]
    helper (N, ans, i+1);

    ans[i] = 2;          // put 2 at ans[i]
    helper (N, ans, i+1);
}
```

printing of ans

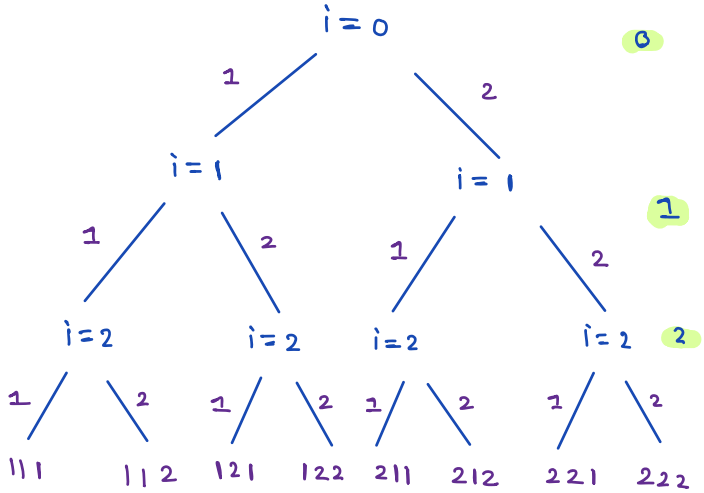
TC : $O(n * 2^n)$

SC : $O(n)$

Dry run

ans : _ _ _
0 1 2

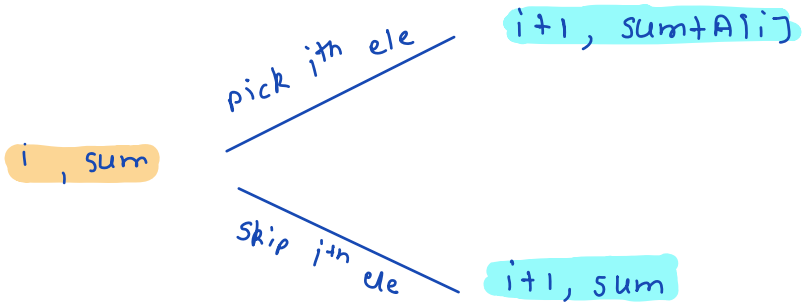
```
void helper (int N, int [] ans, int i) {
    if (i == N) {
        //print the array
        for (int val : ans) {
            sop (val + " ");
        }
        sopln ();
        return;
    }

    // two options
    ans[i] = 1;          // put 1 at ans[i]
    helper (N, ans, i+1);

    ans[i] = 2;          // put 2 at ans[i]
    helper (N, ans, i+1);
}
```



Q.2 Given an array, find count of subsets with sum = K.

A = [5, 7, 2] K = 7 ans = 2 {5, 2} {7}
0 1 2

idea: Out of all possible subsets find the subsets where sum = K.




```
int ans;
int solve(int[] A, int k) {
    ans=0;
    helper(A, k, 0, 0);
    return ans;
}

void helper(int[] A, int k, int i, int sum){
    if(i==A.length){
        if(sum==k) { ans++; }
        return;
    }
    //two options
    helper(A,k,i+1, sum+A[i]); // yes to ith element
    helper(A, k, i+1, sum); // no to ith element
}
```

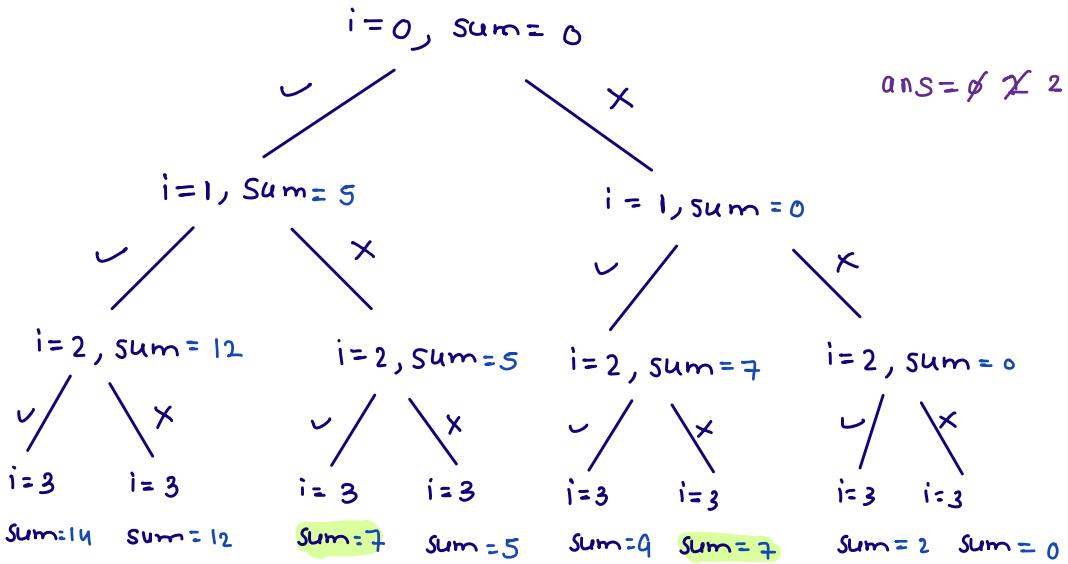
TC: O(2^n)

SC: O(n)
↳ recursive space

Dry run

```
void helper(int[] A, int k, int i, int sum){
    if(i == A.length){
        if(sum == k) { ans++; }
        return;
    }
    //two options
    helper(A, k, i+1, sum+A[i]); // yes
    helper(A, k, i+1, sum); // no
}
```

A = { 5 7 2 } k = 7
0 1 2

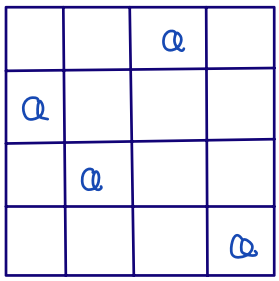


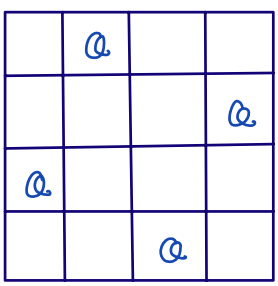
ans = ~~0~~ ~~1~~ 2

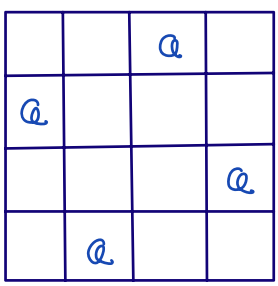
Q.3 Given N, print valid placements of N queens on a NXN board such that no two queens kill each other.

Note: If 2 queens are present in same row/col/diagonal they will kill each other.

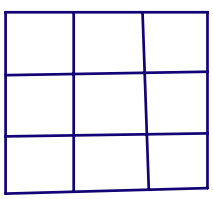
N=4







N=3, place 3 queens in a 3x3 board

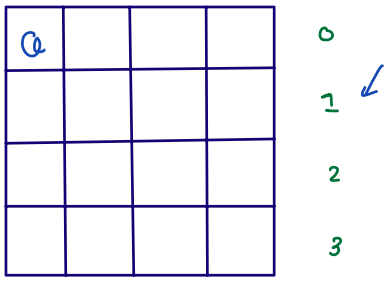


not possible

We need to place N queens on a NXN board.

↳ every row is going to contain exactly one queen.

N = 4

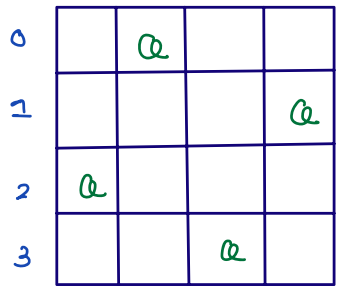


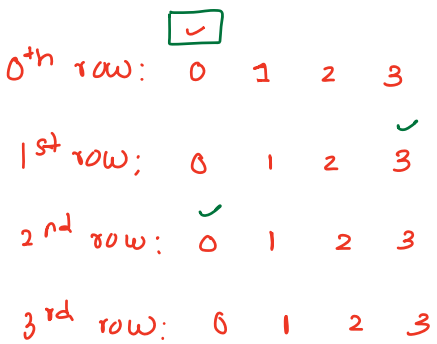
⇒ i^{th} row

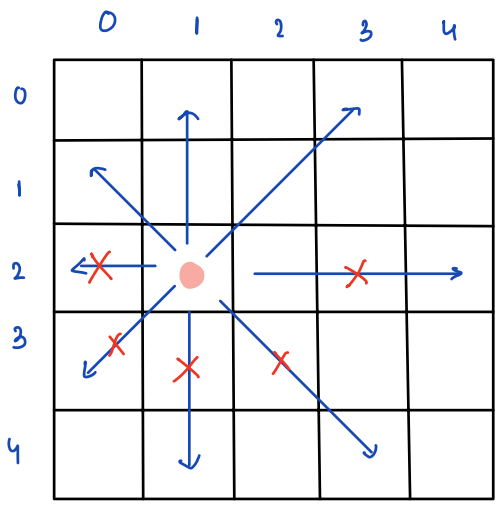
N options {j ⇒ 0 to N-1}

```
void solve (int N) {
    int [][] mat = new int [N][N];
    helper (mat, N, 0);
}

void helper (int [][] mat, int N, int i) {
    if (i == N) {
        // print mat
        return;
    }

    // N options [col : 0 to N-1]
    for (int j=0; j<N; j++) {
        if (check (mat, i, j) == true) {
            mat[i][j] = 1;
            helper (mat, N, i+1);
            mat[i][j] = 0;
        }
    }
}
```

```
void helper(int[][] mat, int N, int i) {
    if(i == N) {
        // print mat
        return;
    }

    // N options [col: 0 to n-1]
    for(int j=0; j<n; j++) {
        if(check(mat, i, j) == true) {
            mat[i][j] = 1;
            helper(mat, N, i+1);
            mat[i][j] = 0;
        }
    }
}
```

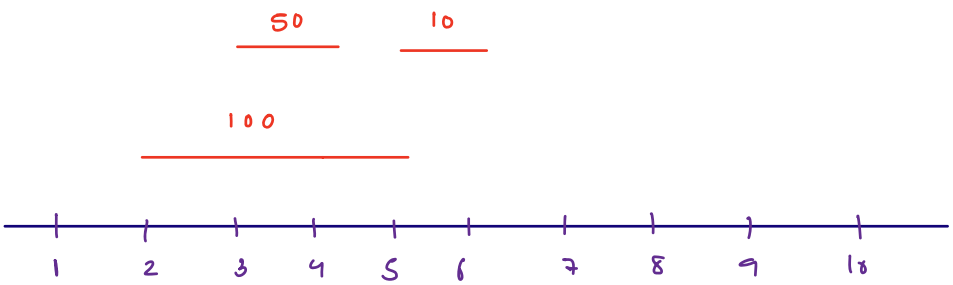






check if we can place queen at (2,1)

Doubts



3, 4, 50

2, 5, 100

5, 6, 10

ans = 0 + 50 - 50 + 100 + 10

last_pair = 5, 6, 10

(maybe list)