BackTracking:

EX: 8Qeen

**public** **static** **void** enumerate() {

**int**[] a = **new** **int**[8];

*enumerate*(a, 0);

}

**public** **static** **void** enumerate(**int**[] q, **int** n) {

**int** N = q.length;

**if** (n == N) {

*printQueens*(q);

*}*

**else** {

**for** (**int** i = 0; i < N; i++) {

q[n] = i;

**if** (canPlaceQueen(q, n))

*enumerate*(q, n+1);

}

}

}

**public** **static** **boolean** canPlaceQueen(**int**[] q, **int** n) {

**for** (**int** i = 0; i < n; i++) {

**if** (q[i] == q[n]) **return** **false**; // same column

**if** ((q[i]-q[n]) == (n-i)) **return** **false**;// same major diagonal

**if** ((q[n]-q[i]) == (n-i)) **return** **false**;// same minor diagonal

}

**return** **true**;

}

Recursion takes care of back tracking automatically.

There is a pattern

* Start from low and go for next
* call the function(recursively) for next
* function returns(poped from stack) only when
  + It completes; with it's all recursive method. - We are done
  + when r=n-1 = we are done partially. i.e. one solution is found
  + If no place is found then for loop terminates, so the function. Then parent function is called which makes it to search with next canPlaceQueen() value.
* Note, it doesn't search with previous (as we have already done this so not reqd, as we started with low)
* it doesn't search randomly
* it only search with next possible solution

All backtracking Algo has same pattern.

* Try to form the logic for canPlaceQueen();
* if it can place go for next with recursion
* If it can't, end the function which will trigger backtracking.
* Stopping condition for the function.