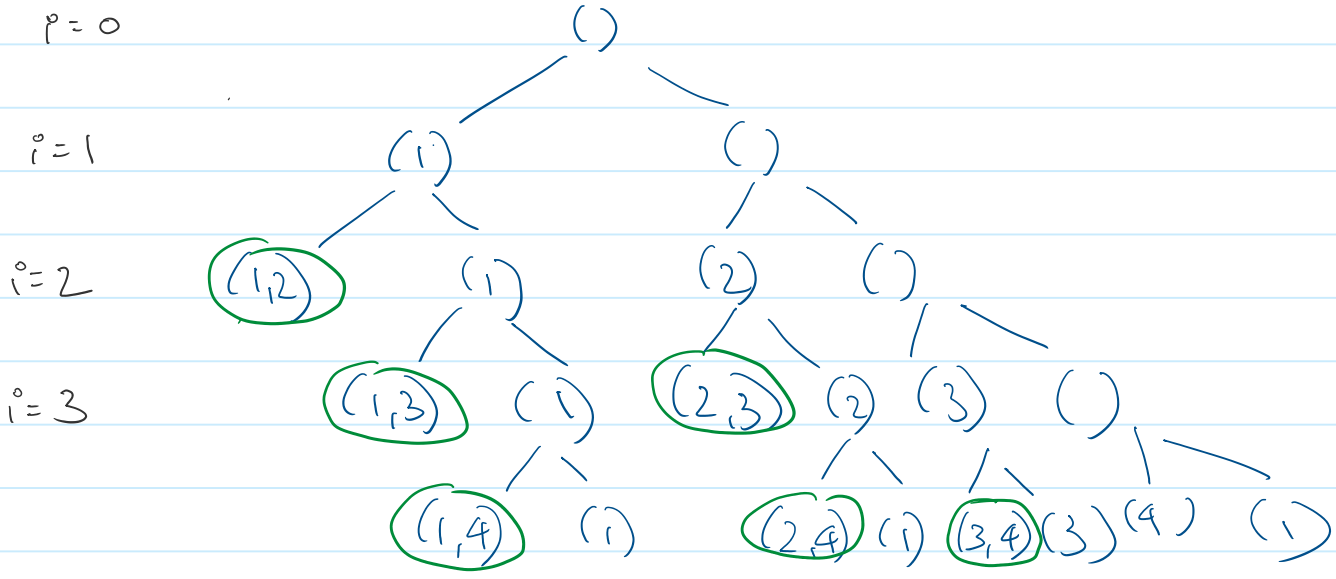
#77. Combinations

- Given two integers, n and k, return all possible combinations of 'k' numbers out of range [1, n].

n = 4, k = 2 [1, 2, 3, 4]

[[1,2], [1,3], [1,4], [2,3], [2,4], [3,4]]

// Brute force



Six combinations, max height of tree = k

Base: if (i > n) return
if (k == 0) ans.push();

// Steps: - Select a element i.e first element, at every position you have two options, either include in subset or ignore it.
- make recursive calls to find the combinations, as demonstrated in decision tree.

// Pseudocode :

```
helper (i, n, k, subset[], ans [][]){
  if (i > n) {                                  // base case
      if ( k == 0) ans.push(subset);
      return
  }
  subset.push(i);                               // include ith ele;
  help(i+1, n, k-1, subset, ans);

  subset.pop();                                 // skip ith ele;
  help(i+1, n, k, subset, ans);
}

combination ( n, k) {
  subset [];
  ans [][];
  help (1, n, k, subset, ans);
  return ans;
}
```

T.C = Exponential

S.C = Stack space + subset + ans

// Slight improvement :

- modify base case :

```
if (k == 0)  ans.push (subset)
    return;
if (i > n) return.
if (k > n - i + 1) return;
```

#289. Game of life

- Given an board of size m x n, each cell has a state
  1 : live
  0 : dead

  1: will live if there are 2 or 3 live neighbours
     else die.
  0: with 3 live neighbours live
     else die

// Brute force (using extra space)

- make a dummy matrix,
- start iterating the board, for each cell, check the all eight directions, and update the cell according to the conditions in the dummy matrix.
- after this, copy the dummy matrix in original board.

0	1	0	⇨	0	0	0
0	0	1		1	0	1
1	1	1		0	1	1
0	0	0		0	1	0

```
dum[][];
for(i=0 → m){
  for(j=0 → n){
    count no of ones for
    current cell.
    if(a[i][j]==1 && ones<2)
      make dead;
    else if(a[i][j]==1 && (ones==2 || ones==3))
      continue;
    else if(a[i][j]==1 && ones>3)
      make dead;
    else if(a[i][j]==0 && ones==3)
      make live;
    else continue
  }
}
copy dum into original.
```

time complexity: O(m.n)
space complexity: O(m.n)

0	0	0
1	0	1
0	1	1
0	1	0

// Brute force (in-place solution)

observe possibilities:

original		new		
0	→	0	map	0 - 0
1	→	0	"	1 - 0
0	→	1	"	2 - 1
1	→	1	"	3 - 1

0/0	1/1	0/0
0/2	0/0	1/3
1/1	1/3	1/3
0/0	0/2	0/0

// traverse and update the original one. to table above.

// change 3s and 2s according to table.

```
for (r = 0 → m){
  for (c = 0 → n){
    nei = countNeighbours(r,c);
    if (board[r][c]:
        if (nei == 2 || nei == 3){
            board[r][c] = 3;
    else if (nei == 3){
        board[r][c] = 2;
    }
}

for (r = 0 → m){
for (c = 0 → n){
if (board[r][c] == 1)  make 0;
else if (board[r][c] == 3 or 2)  make 1;

countNeighbours(r, c){
  neigh = 0;
for (i = r-1 → r+1){
  for (j = c-1 → c+1){
  if ((i == r and j == c) || i < 0 || j < 0 || i == m || j == col):
        Continue
  if (board[i][j] == 1 || 3)  neigh++;
}
  Return neigh;
```

time complexity: O(m x n)

space complexity: O(1)

#1499. Max Value of Equation

- Given an array of points, containing coordinations of 2D plane, points[i] = $[x_i, y_i]$ such that $(x_i < x_j)$ for $1 <= i < j = size$.
- Also given integer k
- Return maximum value of eqn:

$$y_i + y_j + |x_i - x_j| \quad , \quad |x_i - x_j| <= k$$

$[[1,3],[2,0],[5,10],[6,-10]]$, $k = 1$

- satisfying points: (1,3) and (2,0)
  (5,10) and (6,-10)

(1,3) (2,0): $3 + 0 + |1-2| = 3 + 1 = 4$ ✓ Max Value

(5,10) (6,-10): $10 + (-10) + |5-6| = 1$

// Solution:

$x_0 y_0$, $x_1 y_1$, $x_2 y_2$, $x_3 y_3$ (j)

↳ $x_1 - x_0$ (as $x_1 > x_0$)

eqn: $y_i + y_j + |x_i - x_j|$

$y_i + y_j + x_j - x_i$

we need to add (j) coordinate.

$(x_j + y_j) + (y_i - x_i)$ → maximize this

- implementing through priority_queue