Coins of various values are placed on the cells of an n × m chess board. Let the upper left corner cell be (1, 1) and the lower right cell be (n, m); cell (i, j) has coins valued at cij . A robot starts at cell (1, 1) and can move only to the right or down on the board.

1. Give a dynamic programming algorithm expressed recursively without memoization to determine the path the robot should follow to maximize the total value of the coins collected as the robot wanders on the board from cell (1, 1) to cell (n, m).

Analyze the time required and give corresponding pseudocode.

code:

```
def recur_no_mem(cell,i,j):


        if i==len(cell)-1 and j==len(cell[0])-1:
                return cell[len(cell)-1][len(cell[0])-1]
        if i>=len(cell)-1: # only can move right
                return cell[i][j]+recur_no_mem(cell,i,j+1)
        if j>=len(cell[0])-1:# only can move down
                return cell[i][j]+recur_no_mem(cell,i+1,j)

        return max(cell[i][j]+recur_no_mem(cell,i,j+1),cell[i][j]+recur_no_mem(cell,i+1,j))

  def main(cell):

        return(recur_no_mem(cell,i=0,j=0))
```

about time required:

proof:

Draw the recursion-tree (1.1), because we can get the height of tree H=m+n

so ==>. T(N)=O($2^{(m+n)}$)

or use Mathematical :

because the longest path (the numbers of the robot moves) is the sum of the two dimensions of the cell ===> N=m+n

T(0)=1

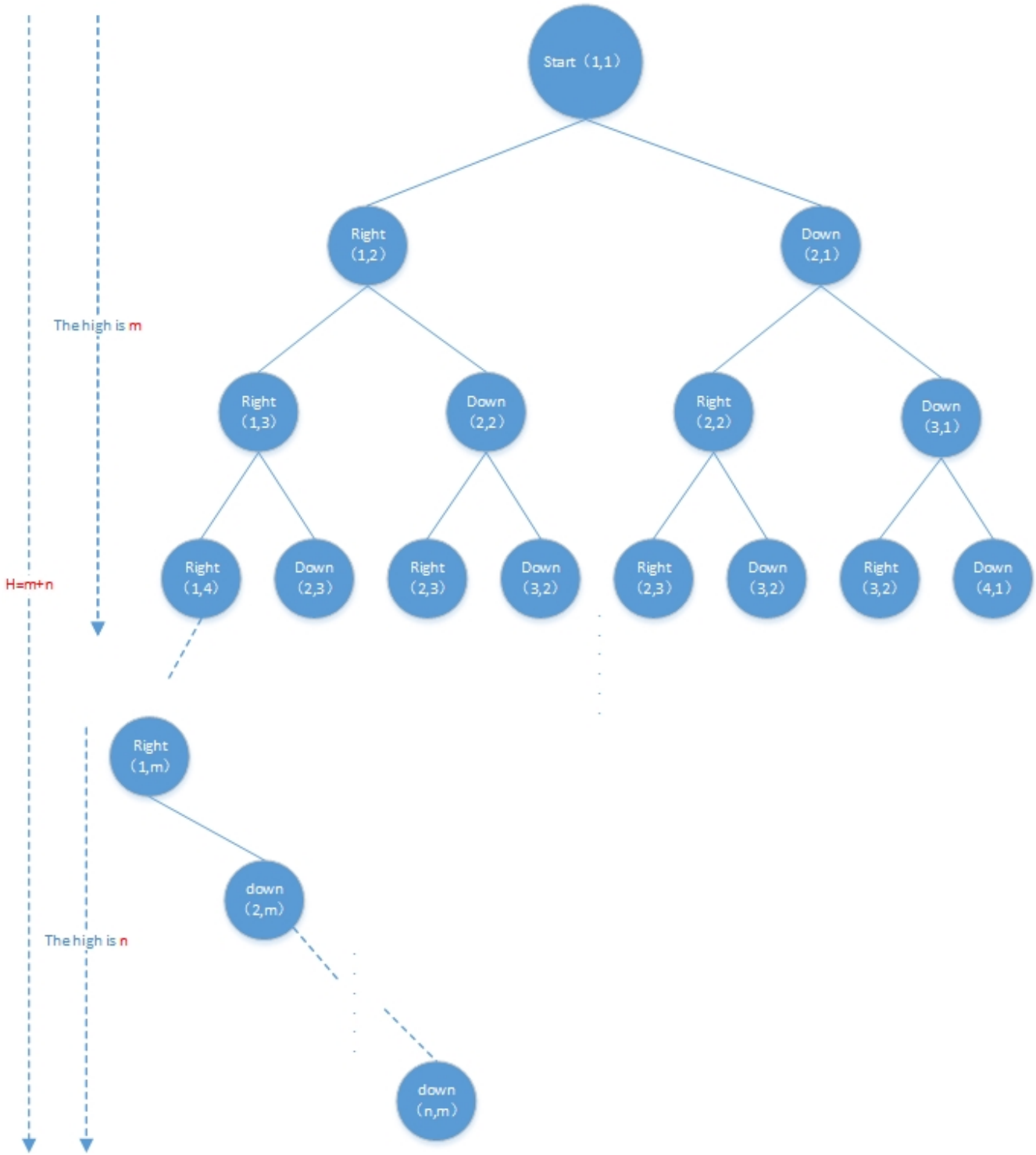
T(N)=1+ $\sum_{i=1}^{N} T(i)$ and T(N-1)=1 + $\sum_{i=1}^{N-1} T(i)$ ==>T(N)=2T(N-1)

so T(N)=O(2^{N})

N=m+n ==> T(N)=O($2^{(m+n)}$)

QED.

(1.1)


Start（1,1）
Right（1,2）
Down（2,1）
Right（1,3）
Down（2,2）
Right（2,2）
Down（3,1）
Right（1,4）
Down（2,3）
Right（2,3）
Down（3,2）
Right（2,3）
Down（3,2）
Right（3,2）
Down（4,1）
Right（1,m）
down（2,m）
down（n,m）
The high is m
The high is n
H=m+n

2. Give the algorithm iteratively with memoization.
Analyze the running time required and specify the corresponding pseudocode.

code:

```
def iteratively_withmem(cell):

                                                                    cost
1       for i in range(0,len(cell)):                                n

2           for j in range(0,len(cell[0])):                         m

3               if i==0 and j==0:
4                   continue
5               if i==0:
6                   cell[i][j]+=cell[i][j-1]
7               elif j==0:
8                   cell[i][j]+=cell[i-1][j]
9               else:
10                  cell[i][j]=max(cell[i][j-1]+cell[i][j],cell[i-1][j]+cell[i][j])

11          return cell[len(cell)-1][len(cell[0])-1]
```

about time required:

because the cell's (width,height) is (n, m)
code line 1 and line 2

==> T(N)=O(n*m)