Unique Paths:

```
class Solution {
public:
    int countPaths(int i,int j,int m,int n){
        //base case
        if(i==m-1&&j==n-1){
            return 1;
        }

        //edge case
        if(i>=m||j>=n){
            return 0;
        }

        int downPaths==countPaths(i+1,j,m,n);
        int rightPaths==countPaths(i,j+1,m,n);
        return (downPaths+rightPaths)%(2*1e9);
    }
    int uniquePaths(int m, int n) {
        int i=0;
        int j=0;
        int paths=countPaths(i,j,m,n);
        return paths%(2*1e9);
    }
};
```

Time:O(2^n*m)(Exponential)(for every m*n cell you were having two recursive calls so)
Space:Recursive Stack Space

Unique Paths-2:

```
class Solution {
public:

     int countPaths(int i,int j,int m,int n,vector<vector<int>>&
obstacleGrid){
        //base case
        if(i==m-1&&j==n-1){
            return 1;
        }

        //edge case
        if(i>=m||j>=n||obstacleGrid[i][j]==1){
            return 0;
```

```
        }

        int downPaths==countPaths(i+1,j,m,n,obstacleGrid);
        int rightPaths==countPaths(i,j+1,m,n,obstacleGrid);
        return (downPaths+rightPaths)%(2*1e9);
    }
    int uniquePathsWithObstacles(vector<vector<int>>& obstacleGrid) {
        int m=obstacleGrid.size();
        int n=obstacleGrid[0].size();
         int i=0;
        int j=0;
        int paths=countPaths(i,j,m,n,obstacleGrid);
        return paths%(2*1e9);
    }
};
```

Time:O(2^n*m)(Exponential)(for every m*n cell you were having two recursive calls so)
Space:Recursive Stack Space

Climb Stairs:

```
#include <iostream>
#include<vector>
using namespace std;

void climbStairs(int n,vector<int>&ans){
   //base case
   if(n==0){
      for(int x:ans){
         cout<<x<<" ";
      }
      cout<<endl;
      return;
   }

   //edge case
   if(n==-1){
      return;
   }

   //one step down
   ans.push_back(1);
   climbStairs(n-1,ans);
   ans.pop_back();
```

```
    //two step down
    ans.push_back(2);
    climbStairs(n-2,ans);
    ans.pop_back();

}
int main() {
    int n=2;
    vector<int>ans;
    climbStairs(n,ans);
    return 0;
}
```

Time:O(n)
Space:Recursive stack space and vector space won't be counted because it is asked in question to use vector we haven't used any extra space

Print Maze Paths:

```
#include <iostream>
#include<vector>
using namespace std;

void printPaths(int i,int j,int m,int n,vector<char>&ans){
    //base case
    if(i==m-1&&j==n-1){
        for(char x:ans){
            cout<<x<<"";
        }
        cout<<endl;
    }

    //edge case
    if(i>=m||j>=n){
        return;
    }

    //down
    ans.push_back('D');
     printPaths(i+1,j,m,n,ans);
     ans.pop_back();


     //right
     ans.push_back('R');
     printPaths(i,j+1,m,n,ans);
```

```
    ans.pop_back();
}

int main() {
    int m=3;
    int n=2;
    int i=0;
    int j=0;
    vector<char>ans;
    printPaths(i,j,m,n,ans);

    return 0;
}
```

Time:O(2^n*m)(Exponential)(for every m*n cell you were having two recursive calls so)
Space:Recursive Stack Space

Print All Subsequences:

```
#include <iostream>
#include<vector>
using namespace std;


void printSubsequences(int i,string s,vector<char>&ans){
    //base case
    if(i==s.size()){
        for(char ch:ans){
            cout<<ch<<"";
        }
        cout<<endl;
        return;
    }

    //pick
    ans.push_back(s[i]);
    printSubsequences(i+1,s,ans);
    ans.pop_back();

    //not pick
    printSubsequences(i+1,s,ans);
}
int main() {
    string s="abc";
    int i=0;
    vector<char>ans;
```

```
    printSubsequences(i,s,ans);

    return 0;
}
```

Time:O(2^n,where n is size of string)(Exponential)(for every n array elemnts you were having two recursive calls of pick and not pick so)
Space:Recursive Stack Space

Climbing Stairs:

```
class Solution {
public:
    int climbStairs(int n) {
        //base case
        if(n==0||n==1){
            return 1;
        }

        int xwaysToReachGroudBy1step=climbStairs(n-1);
        int ywaysToReachGroudBy2step=climbStairs(n-2);
        return xwaysToReachGroudBy1step+ywaysToReachGroudBy2step;

    }
};
```

Time:O(n)
Space:recursive Stack Space(function calls that are stored in stack order)