RECURSION +BT

Q1)Print in the descending order

```
#include<bits/stdc++.h>
using namespace std;
void fun(int n){
 if(n>0){
    cout<<n;
    fun(n-1);
 }
}
int main(){
   int x;
   cin>>x;
   fun(x);
}
```

Q2)Print in the ascending order

```
#include<bits/stdc++.h>
using namespace std;
void fun(int n){
 if(n>0){
    fun(n-1);
    cout<<n;

 }
}
int main(){
   int x;
   cin>>x;
   fun(x);
}
```

Q3) the difference in adding

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n){
 if(n>0){
    return fun(n-1)+n;//adding the previous case
 }
 return 0;
}
int main(){
   int x;
   cin>>x;
   cout<<fun(x);
}
```

Q4)Tree recursion

```
#include<bits/stdc++.h>
using namespace std;
void fun(int n){
   if(n>0){
      cout<<n;
      fun(n-1);
      fun(n-1);
   }
}
int main(){
   int x;
   cin>>x;
   fun(x);
}
```

Q5)Indirect recursion

```
#include<bits/stdc++.h>
using namespace std;
void funB(int n);//declaring first
void funA(int n){
   if(n>0){
      cout<<n;
      funB(n-1);
   }
}
void funB(int n){
   if(n>1){
      cout<<n;
      funA(n/2);

   }
}
int main(){
   int x;
   cin>>x;
   funA(x);
}
```

Q6)nested recursion

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n){
   if(n>100){
      return n-10;
   }else{
      return fun(fun(n+11));
   }
}
int main(){
   int n;
   cin>>n;
   cout<<fun(n);
}
```

Q7)sum of first n natural numbers

Case1:

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n){
 if(n==0){
return 0;
}
else{
    return fun(n-1)+n;//adding the previous case
 }
 return 0;
}
int main(){
   int x;
   cin>>x;
   cout<<fun(x);
}
```

Q8)factorial of a number

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n){
 if(n==0){
   return 1;
}
    return fun(n-1)*n;
}
int main(){
   int x;
   cin>>x;
   cout<<fun(x);
}
```

Q9)power

```
#include<bits/stdc++.h>
using namespace std;
int fun(int m,int n){
 if(n==0){
   return 1;
}
else{
    return fun(m,n-1)*m;
 }
return 0;
}
int main(){
   int x,y;
   cin>>x;
   cin>>y;
   cout<<fun(y,x);
}
```

Q10) Taylor series

We learned about all sum, factorial and exponents and now we have to add all the things here to get the result

```
#include<bits/stdc++.h>
using namespace std;
int fun(int m,int n){
 static int r,p=1,f=1;//static because they retain there values during calls
 if(n==0){
   return 1;
}
else{
    r=fun(m,n-1);
    p=p*m;
    f=f*n;

    return r+p/f;
 }
return 0;
}
int main(){
   int x,y;
   cin>>x;
   cin>>y;
   cout<<fun(y,x);
}
```

Through horners rule:

Here we decrease the number of multiplications

```
#include<bits/stdc++.h>
using namespace std;
int fun(int m,int n){
 static int s=1;
 if(n==0){
   return s;
}
s=1+m/n*s;
return fun(m,n-1);
 }

int main(){
   int x,y;
   cin>>x;
   cin>>y;
   cout<<fun(x,y);
}
```

Q11)fibbonaci series

Normal approach

Through the addition of two functions we cover all

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n){
```

```
 static int s=1;
 if(n<=1){
   return n;
}

return fun(n-2)+fun(n-1);
 }

int main(){
   int x,y;
   cin>>x;
   cout<<fun(x);
}
```

Pro:

Through memorization we use arrays to store the value of the calls that has been already called once.

```
#include<bits/stdc++.h>
using namespace std;

int fun(int n){
   int F[10];//for reference lets 10
   fill(F, F + 10, -1);

 if(n<=1){
    F[n]=n;
   return F[n];
}else{
   if(F[n-2]==-1){//if not seen put the value
      F[n-2]=fun(n-2);
   }
   if(F[n-1]==-1){
      F[n-1]=fun(n-1);
   }
   return F[n-1]+F[n-2];
}

return 0;
 }

int main(){
   int x,y;
   cin>>x;
   cout<<fun(x);
}
```

Q12)print NCR(permutation)

We use the concept of pasacals triangle here it's a important case to visualise to get thing from the bottom to the top

```
#include<bits/stdc++.h>
using namespace std;

int fun(int n,int r){
  if(r==0 || n==r){
     return 1;
  }else{
     return fun(n-1,r-1)+fun(n-1,r);
  }
  return 0;
 }

int main(){
   int x,y;
   cin>>x;
   cin>>y;
   cout<<fun(x,y);
}
```

Q13)Palindrome check

Here we traverse from both the sides and check simultaneously. And more importantly traversing through the string/array

```
#include<bits/stdc++.h>
using namespace std;

int fun(string str ,int start ,int end){
  if(start>=end){
     return true;
  }
  return(str[start]==str[end])&&fun(str,start+1,end-1);
 }

int main(){
   int x,y;
   string str;
   cin>>str;
   int n=str.length();
   cout<<fun(str,0,n-1);
}
```

Q14)sum of digits

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n){
 static int s=1;
 if(n<=9){
   return n;
}

return fun(n/10)+n%10;
 }

int main(){
```

```
   int x,y;
   cin>>x;
   cout<<fun(x);
}
```

15)Rope cutting problem

Heres an popular interview problem to verify which of the following possibilities is true and return yes if its true..

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n,int a,int b,int c){
 if(n==0){
    return 0;
 }
 if(n<0){
    return -1;
 }
 int res = max(fun(n-a, a, b, c),max(fun(n-b, a, b,
c),fun(n-c, a, b, c)));
 if(res==-1){
 return -1;//corner case
 }
 return res+1;
 }

int main(){
   int x,y,z,a,b,c;
   cin>>x;
   cin>>a>>b>>c;
   cout<<fun(x,a,b,c);
}
```

16)Generate all subsets

```
#include<bits/stdc++.h>
using namespace std;
void fun(string s,string curr,int index){
  if(index==s.length()){
     cout<<curr;
     return;
  }
  fun(s,curr,index+1);//part1 recursive
  fun(s,curr+s[index],index+1);//part2 recursive

 }

int main(){
   string s;
   cin>>s;
   fun(s,"",0);
}
```

Q17)Tower of Hanoi

Note the path followed

```
#include<bits/stdc++.h>
using namespace std;
void fun(int n,int a,int b,int c){
  if(n>0){
     fun(n-1,a,c,b);
     cout<<"("<<a<<","<<c<<")";
     fun(n-1,b,a,c);
  }

 }

int main(){
  int i,n;
  cin>>n;
  fun(n,1,2,3);
}
```

Q18)Josephus problem

Here the question assumes that it starts from 0 if it starts from 1 we create a function and add 1 to it we will get the following results.

```
#include<bits/stdc++.h>
using namespace std;
int fun(int n,int k){
if(n==1){
   return 0;
}else{
   return (fun(n-1,k)+k)%n;
}

 }

int main(){
  int i,n,k;
  cin>>n>>k;
  cout<<fun(n,k);
}
```

Q19)subset sum problem

```
#include<bits/stdc++.h>
using namespace std;
int fun(vector<int>v,int n,int sum){
if(n==0){
   return (sum==0)?1:0;//how it stores all values
}
 return fun(v,n-1,sum)+fun(v,n-1,sum-v[n-1]);

 }

int main(){
  int i,n,k,temp;
  cin>>n>>k;
  vector<int>v;
  for(int i=0;i<n;i++){
```

```
      cin>>temp;
      v.push_back(temp);
   }

   cout<<fun(v,n,k);
}
```

Q20)permutations and backtracking

```
#include<bits/stdc++.h>
using namespace std;
void fun(string s,int i){//intially its 0
if(i==s.length()-1){
    cout<<s;
    return;//stopping unneccsary calls
}
 for(int j=i;j<s.length();j++){
  swap(s[i],s[j]);
  fun(s,i+1);
  swap(s[i],s[j]);//backtracking
 }
 }

int main(){
  string s;
  cin>>s;

  fun(s,0);
}
```

BACKTRACKING

Q1) 0-1 knapsack problem

```
class Solution {
public:
   int knapSackHelper(int W, vector<int>& wt, vector<int>& val, int i, vector<vector<int>>& dp) {
      // Base case: if no items left or capacity is 0
      if (i == wt.size() || W == 0)
         return 0;

      // Check if the solution already exists in dp table
      if (dp[i][W] != -1)
         return dp[i][W];

      // If the weight of the item is more than the current capacity, skip it
      if (wt[i] > W)
         return dp[i][W] = knapSackHelper(W, wt, val, i + 1, dp);

      // Include the current item or skip it
      return dp[i][W] = max(val[i] + knapSackHelper(W - wt[i], wt, val, i + 1, dp),
                            knapSackHelper(W, wt, val, i + 1, dp));
   }

   int knapSack(int W, vector<int>& wt, vector<int>& val) {
      int n = wt.size();
      vector<vector<int>> dp(n, vector<int>(W + 1, -1));
      return knapSackHelper(W, wt, val, 0, dp);
   }
};
```

Q2)

https://leetcode.com/problems/burst-balloons/

```
class Solution {
public:
   int maxCoinsHelper(vector<int>& nums, int left, int right, vector<vector<int>>& dp) {
      if (left + 1 == right) return 0; // No balloons to burst in this interval

      if (dp[left][right] != -1) return dp[left][right]; // Already computed

      int maxCoins = 0;
      for (int k = left + 1; k < right; ++k) {
         int coins = nums[left] * nums[k] * nums[right];
         coins += maxCoinsHelper(nums, left, k, dp);
         coins += maxCoinsHelper(nums, k, right, dp);
         maxCoins = max(maxCoins, coins);
      }

      dp[left][right] = maxCoins; // Memoize the result
      return maxCoins;
   }

   int maxCoins(vector<int>& nums) {
      int n = nums.size();
      nums.insert(nums.begin(), 1); // Add 1 at the beginning
      nums.push_back(1); // Add 1 at the end
      vector<vector<int>> dp(n + 2, vector<int>(n + 2, -1)); // DP table
      return maxCoinsHelper(nums, 0, n + 1, dp); // Compute max coins
   }
};
```

Q3)
https://leetcode.com/problems/palindrome-partitioning/

This is a classic bt template question

```
class Solution {
public:
   int n;

   bool isPalindrome(string &s, int l, int r) {

      while(l < r) {
         if(s[l] != s[r])
            return false;
         l++;
         r--;
      }

      return true;

   }

   void backtrack(string &s, int idx, vector<string> curr, vector<vector<string>> &result) {

      if(idx == n) {
         result.push_back(curr);
         return;
      }

      for(int i = idx; i<n; i++) {

         if(isPalindrome(s, idx, i)) {

            curr.push_back(s.substr(idx, i-idx+1));

            backtrack(s, i+1, curr, result);

            curr.pop_back();

         }
      }
   }
   vector<vector<string>> partition(string s) {
      n = s.length();
      vector<vector<string>> result;
      vector<string> curr;
      backtrack(s, 0, curr, result);
      return result;

   }
};
```

Q4) **https://leetcode.com/problems/word-search/description/**

```
class Solution {
public:
   int l, m, n;
   vector<vector<int>> directions{{0, 1}, {0, -1}, {1, 0}, {-1, 0}};
   bool find(vector<vector<char>>& board, int i, int j, string &word, int idx) {
      if(idx >= l)
         return true;

      if(i < 0 || i >= m || j < 0 || j >= n || board[i][j] != word[idx])
         return false;

      char temp = board[i][j];
      board[i][j] = '$';

      for(auto& dir : directions) {
         int i_ = i + dir[0];
         int j_ = j + dir[1];

         if(find(board, i_, j_, word, idx+1))
            return true;
      }

      board[i][j] = temp;
      return false;
   }

   bool exist(vector<vector<char>>& board, string word) {
      m = board.size();
      n = board[0].size();
      l = word.length();
      if(m*n < l)
         return false;

      for(int i = 0; i<m; i++) {
         for(int j = 0; j<n; j++) {
            if(board[i][j] == word[0] && find(board, i, j, word, 0)) {
               return true;
            }
         }
      }

      return false;
   }
};
```

5) https://leetcode.com/problems/non-decreasing-subsequences/

```
class Solution {
public:
    int n;

    void backtrack(vector<int>& nums, int idx,
vector<int>& curr, vector<vector<int>>& result) {
        if(curr.size() > 1)
            result.push_back(curr);

        unordered_set<int> st;
        for(int i = idx; i<n; i++) {
            if((curr.empty() || nums[i]  >= curr.back())
&& st.find(nums[i]) == st.end()) {

                curr.push_back(nums[i]);
                backtrack(nums, i+1, curr, result);
                curr.pop_back();

                st.insert(nums[i]);
            }
        }
    }

    vector<vector<int>>
findSubsequences(vector<int>& nums) {

        n = nums.size();
        vector<vector<int>> result;
        vector<int> curr;
        backtrack(nums, 0, curr, result);
        return result;
    }
};
```

6) https://leetcode.com/problems/fair-distribution-of-cookies/

```
class Solution {
public:
    int result = INT_MAX;
    int n;
    void solve(int idx, vector<int>& cookies,
vector<int>& children, int k) {
        if(idx == cookies.size()) {

            int ans = *max_element(begin(children),
end(children));
            result = min(result, ans);
            return;
        }

        int candy = cookies[idx];
        for(int i = 0; i<k; i++) {
            children[i] += candy;

            solve(idx+1, cookies, children, k);

            children[i] -= candy;
        }

    }
    int distributeCookies(vector<int>& cookies, int
k) {
        n = cookies.size();
        vector<int> children(k);
        solve(0, cookies, children, k);

        return result;
    }
};
```

7) https://leetcode.com/problems/maximum-number-of-achievable-transfer-requests/description/

try using for loop

```
class Solution {
public:
    int m;
    int result = INT_MIN;

    void solve(int idx, int count, int n, vector<int>&
resultant, vector<vector<int>>& requests) {
        if(idx == m) {

            int allZero = true;
            for(int &x : resultant) {
                if(x != 0) {
                    allZero = false;
                    break;
                }
            }

            if(allZero) {
                result = max(result, count);
            }
            return;
        }


        int from = requests[idx][0];
        int to   = requests[idx][1];

        resultant[from]--;
```

```
        resultant[to]++;
        solve(idx+1, count+1, n, resultant, requests);

        resultant[from]++;
        resultant[to]--;
        solve(idx+1, count, n, resultant, requests);
    }

    int maximumRequests(int n,
vector<vector<int>>& requests) {
        m = requests.size();

        vector<int> resultant(n, 0);

        solve(0, 0, n, resultant, requests);

        return result;
    }
};
```

8)

https://leetcode.com/problems/combinations/

```
class Solution {
public:
    vector<vector<int>> result;

    void solve(int start, int n, int k, vector<int>&
temp) {
        if(k == 0) {
            result.push_back(temp);
            return;
        }

        for(int i = start; i <= n; i++) {
            temp.push_back(i);
            solve(i+1, n, k-1, temp);
            temp.pop_back();
        }
    }

    vector<vector<int>> combine(int n, int k) {
        vector<int> temp;

        solve(1, n, k, temp);

        return result;
    }
};


//Khandani Backtracking Approach (without for
loop)

class Solution {
public:
    vector<vector<int>> result;

    void solve(int start, int n, int k, vector<int>&
temp) {
        if(k == 0) {
            result.push_back(temp);
            return;
        }

        if(start > n)
            return;

        temp.push_back(start);
        solve(start+1, n, k-1, temp);
        temp.pop_back();
        solve(start+1, n, k, temp);
    }

    vector<vector<int>> combine(int n, int k) {
        vector<int> temp;

        solve(1, n, k, temp);

        return result;
    }
};
```

9)

https://leetcode.com/problems/permutations/

```
class Solution {
public:
    vector<vector<int>> result;
    int n;

    void solve(int idx, vector<int>& nums) {

        if(idx == n) {
            result.push_back(nums);
            return;
        }
        for(int i = idx; i < n; i++) {
            swap(nums[i], nums[idx]);

            solve(idx+1, nums);

            swap(nums[i], nums[idx]);
        }
    }

    vector<vector<int>> permute(vector<int>&
nums) {
        n = nums.size();
```

```
        solve(0, nums);
        return result;
    }
};
//pattern-2
class Solution {
public:
    vector<vector<int>> result;
    unordered_set<int> st;
    int n;

    void solve(vector<int>& temp, vector<int>&
nums) {

        if(temp.size() == n) {
            result.push_back(temp);
            return;
        }

        for(int i = 0; i<n; i++) {
            if(st.find(nums[i]) == st.end()) {
                temp.push_back(nums[i]);
                st.insert(nums[i]);

                solve(temp, nums);

                st.erase(nums[i]);
                temp.pop_back();
            }
        }


    }

    vector<vector<int>> permute(vector<int>&
nums) {
        n = nums.size();

        vector<int> temp;

        solve(temp, nums);

        return result;
    }
};
```

10)
https://leetcode.com/problems/permutations-ii/

```
class Solution {
public:
    int n;
    vector<vector<int>> result;

    void backtrack(vector<int>& temp,
unordered_map<int, int>& mp) {

        if (temp.size() == n) { //we got all numbers
            result.push_back(temp);
            return;
        }

        for (auto& [num, count] : mp) {

            if (count == 0)
                continue;

            // Do something
            temp.push_back(num);
            mp[num]--;

            // Explore it
            backtrack(temp, mp);

            // Undo it
            temp.pop_back();
            mp[num]++;
        }
    }

    vector<vector<int>>
permuteUnique(vector<int>& nums) {
        n = nums.size();
        unordered_map<int, int> mp;

        // count the occurrence of each number
        for (int& num : nums) {
            mp[num]++;
        }

        vector<int> temp;
        backtrack(temp, mp);

        return result;
    }
};
```

Q11)
https://leetcode.com/problems/combination-sum/

```
class Solution {
public:
    vector<vector<int>> result;

    void solve(vector<int>& candidates, int target,
int start, vector<int>& temp) {
        if(target == 0) {
```

```
            result.push_back(temp);
            return;
        }

        for(int i = start; i < candidates.size(); i++) {
            if(candidates[i] <= target) {
                temp.push_back(candidates[i]);
                solve(candidates, target - candidates[i], i,
temp); // not i+1 because we can reuse same
elements
                temp.pop_back();
            }
        }
    }

    vector<vector<int>>
combinationSum(vector<int>& candidates, int
target) {
        vector<int> temp;
        solve(candidates, target, 0, temp);
        return result;
    }
};
```

Q12)

https://leetcode.com/problems/combination-sum-ii/

```
class Solution {
public:
    vector<vector<int>> result;

    void solve(vector<int>& candidates, int target,
int start, vector<int>& temp) {
        if(target == 0) {
            result.push_back(temp);
            return;
        }

        for(int i = start; i < candidates.size(); i++) {
            if(i > start && candidates[i] == candidates[i-
1]) {
                continue; // skip duplicates
            }
            if(candidates[i] <= target) {
                temp.push_back(candidates[i]);
                solve(candidates, target - candidates[i], i
+ 1, temp); // i+1 because we can't reuse the same
element
                temp.pop_back();
            }
        }
    }

    vector<vector<int>>
combinationSum2(vector<int>& candidates, int
target) {
        vector<int> temp;
        sort(candidates.begin(), candidates.end());
        solve(candidates, target, 0, temp);
        return result;
    }
};
```

Q13)

https://leetcode.com/problems/subsets-ii/

```
class Solution {
public:
    vector<vector<int>> result;

    void solve(vector<int>& nums, int start,
vector<int>& temp) {
        result.push_back(temp);

        for(int i = start; i < nums.size(); i++) {
            if(i > start && nums[i] == nums[i-1]) {
                continue; // skip duplicates
            }
            temp.push_back(nums[i]);
            solve(nums, i + 1, temp);
            temp.pop_back();
        }
    }

    vector<vector<int>>
subsetsWithDup(vector<int>& nums) {
        vector<int> temp;
        sort(nums.begin(), nums.end());
        solve(nums, 0, temp);
        return result;
    }
};
```

Q14)

https://leetcode.com/problems/combination-sum-iv/

```
class Solution {
public:
    int n;
    vector<vector<int>> t;

    int solve(int idx, vector<int>& nums, int target) {
        if (target == 0)
            return 1;

        if (idx >= n || target < 0)
            return 0;

        int result = 0;
```

```
        if (t[target][idx] != -1)
            return t[target][idx];

        for (int i = idx; i < n; i++) {
            int take_i = solve(0, nums, target - nums[i]);
            result += take_i;
        }

        return t[target][idx] = result;
    }

    int combinationSum4(vector<int>& nums, int
target) {
        n = nums.size();
        t = vector<vector<int>>(target + 1,
vector<int>(n, -1));
        return solve(0, nums, target);
    }
};
```

Dp approach

```
class Solution {
public:
    int n;
    vector<vector<int>> t;

    int solve(vector<int>& nums, int target, int idx) {
        if (target == 0)
            return 1;

        if (idx >= n || target < 0)
            return 0;

        if (t[target][idx] != -1)
            return t[target][idx];

        int take_idx = solve(nums, target - nums[idx],
0);
        int reject_idx = solve(nums, target, idx + 1);

        return t[target][idx] = take_idx + reject_idx;
    }

    int combinationSum4(vector<int>& nums, int
target) {
        n = nums.size();
        t.resize(target + 1, vector<int>(n, -1));

        return solve(nums, target, 0);
    }
};
```

Note the difference between the two

Q15)
https://leetcode.com/problems/letter-combinations-of-a-phone-number/

```
class Solution {
public:
    vector<string> result;

    void solve(int idx, string &digits, string &temp,
unordered_map<char, string> &mp) {

        if(idx == digits.length()) {
            result.push_back(temp);
            return;
        }


        char ch = digits[idx];
        string str = mp[ch];

        for(int i = 0; i<str.length(); i++) {

            //Do
            temp.push_back(str[i]);
            solve(idx+1, digits, temp, mp);
            temp.pop_back();

        }

    }

    vector<string> letterCombinations(string digits) {
        if(digits.length() == 0)
            return {};

        unordered_map<char, string> mp;

        mp['2'] =  "abc";
        mp['3'] = "def";
        mp['4'] = "ghi";
        mp['5'] = "jkl";
        mp['6'] = "mno";
        mp['7'] = "pqrs";
        mp['8'] = "tuv";
        mp['9'] = "wxyz";

        string temp = "";

        solve(0, digits, temp, mp);

        return result;
```

```
    }
};
```

Q16)
https://leetcode.com/problems/path-with-maximum-gold/

Try after dfs

Q17)
https://leetcode.com/problems/the-number-of-beautiful-subsets/

```
class Solution {
public:
    int result;
    int K;
    void solve(vector<int> &nums, int idx, unordered_map<int, int> &mp) {
        if (idx == nums.size()) {
            result++;
            return;
        }

        // not_take
        solve(nums, idx + 1, mp);

        // checking if we can take it or not
        if (!mp[nums[idx] - K] && !mp[nums[idx] + K]) {
            mp[nums[idx]]++;
            solve(nums, idx + 1, mp);
            mp[nums[idx]]--;
        }
    }

    int beautifulSubsets(vector<int>& nums, int k) {
        result = 0;
        K = k;
        unordered_map<int, int> mp;

        solve(nums, 0, mp);

        return result - 1; // -1 because we don't want to count the empty subset in the result
    }
};
```

18)
https://leetcode.com/problems/unique-paths-iii/

```
class Solution {
public:
    int m, n;
    int emptyCells;
    int result = 0;
    vector<vector<int>> directions{{1, 0}, {-1, 0}, {0, 1}, {0, -1}};

    void dfs(vector<vector<int>>& grid, int curr_count, int i, int j) {
        if(i < 0 || i >= m || j < 0 || j >= n || grid[i][j] == -1) {
            return;
        }

        if(grid[i][j] == 2) {
            if(curr_count == emptyCells) {
                result++;
            }
            return;
        }


        grid[i][j] = -1;
        for(vector<int> dir:directions) {
            int i_ = i + dir[0];
            int j_ = j + dir[1];
            dfs(grid, curr_count+1, i_, j_);
        }
        grid[i][j] = 0;
    }

    int uniquePathsIII(vector<vector<int>>& grid) {
        m = grid.size();
        n = grid[0].size();
        emptyCells = 0;
        result = 0;

        int start_x = 0;
        int start_y = 0;

        for(int i = 0; i<m; i++) {
            for(int j = 0; j<n; j++) {
                if(grid[i][j] == 0)
                    emptyCells++;

                if(grid[i][j] == 1) {
                    start_x = i;
                    start_y = j;
                }
            }
        }

        emptyCells += 1; //walk over every non-obstacle square exactly once.
        int curr_count = 0;

        dfs(grid, curr_count, start_x, start_y);
```

```
        return result;
    }
};
```

Rest
josephus problem and grid questions try after
graphs