54. Spiral Matrix

cramp it full

```
vector<int> res;
    int ct;
    void foo(int r1, int r2, int c1, int c2, vector<vector<int>> &matrix, int tot)
    {
        for(int i=c1;i<=c2 && ++ct<=tot;i++)
            res.push_back(matrix[r1][i]);

        for(int i=r1+1;i<=r2 && ++ct<=tot;i++)
            res.push_back(matrix[i][c2]);

        for(int i=c2-1;i>=c1 && ++ct<=tot;i--)
            res.push_back(matrix[r2][i]);

        for(int i=r2-1;i>r1 && ++ct<=tot;i--)
            res.push_back(matrix[i][c1]);

    }
    vector<int> spiralOrder(vector<vector<int>>& matrix) {

        int n= matrix.size();
        int m= matrix[0].size();

        res.clear();
        ct=0;
        int r1=0,r2=n-1,c1=0,c2=m-1;
        while(r1<=r2 && c1<=c2){
            foo(r1,r2,c1,c2,matrix,n*m);
            r1++;
            r2--;
            c1++;
            c2--;
        }
        return res;
    }
```

56. Merge Intervals

```
vector<vector<int>> merge(vector<vector<int>>& intervals) {

        int n= intervals.size();

        vector<vector<int>> res;
        sort(intervals.begin(),intervals.end());
        int l= intervals[0][0];
        int r= intervals[0][1];
        for(int i=1;i<n;i++){

            if(intervals[i][0]> r){
                //intert the intervals
                res.push_back(vector<int>{l,r});
                l= intervals[i][0];
                r= intervals[i][1];
            }
            r= max(r,intervals[i][1]);
        }
        res.push_back(vector<int>{l,r});
        return res;
    }
```

75. Sort Colors

**Bhai rat le isko dobara smajhna mat dimaag hi kharab hoga non intutive hain **

```
int n= nums.size();
        int ptr1=0,ptr2=0,ptr3=n-1;

        while(ptr2<=ptr3){
            if(nums[ptr2]==1)
                ptr2++;
            else if(nums[ptr2]==0)
                swap(nums[ptr2++],nums[ptr1++]);
            else
                swap(nums[ptr2],nums[ptr3--]);
        }
        return;
```

87. Scramble String

IMPLEMENTATION BASED ON Matrix Chain Multiplication

```
class Solution {
public:
    unordered_map<string,bool> mp;
    bool is_scrambled(string a, string b)
    {
        if(a==b){
            //cout<<a<<" "<<b<<endl;
            return true;
        }

        string xx= a;
        xx.push_back('_');
        xx.append(b);

        //cout<<xx<<endl;
        if(mp.find(xx)!=mp.end())
            return mp[xx];

        vector<int> az(26,0),bz(26,0);
        for(char c: a)
            az[c-'a']++;
        for(char c: b)
            bz[c-'a']++;
        if(az!=bz)
            return false;

        int n= a.length();
        //break like in mcm format
        for(int i=1;i<=n-1;i++){

            //no swap condition
            if(is_scrambled(a.substr(0,i),b.substr(0,i)) && is_scrambled(a.substr
(i),b.substr(i)))
                return mp[xx]= true;
            if(is_scrambled(a.substr(0,i),b.substr(n-i)) && is_scrambled(a.substr
(i),b.substr(0,n-i)))
                return mp[xx]= true;
        }
        return mp[xx]= false;
    }

    bool isScramble(string s1, string s2) {

        int n= s1.length();

        //check if s1 and s2 are anagrams;
        vector<int> a(26,0),b(26,0);
        for(char c: s1)
            a[c-'a']++;
        for(char c: s2)
            b[c-'a']++;
```

```
        if(a!=b)
            return false;

        mp.clear();
        return is_scrambled(s1,s2); //return true if s1 and s2 are scrambled strin
gs
    }
};
```

122. Best Time to Buy and Sell Stock II

GREEDY int maxProfit(vector& prices) {

```
    int n= prices.size();
    int ans=0;

    int l=0,r=0;

    while(r<n){
        while(r+1<n && prices[r+1]>prices[r])
            r++;
        ans+= prices[r]-prices[l];
        l=r;

        while(r+1<n && prices[r+1]<=prices[r]){
            r++;
            l++;
        }
        r++;
    }
    return ans;
}
```

123. Best Time to Buy and Sell Stock III

int maxProfit(vector& prices) {

```
    int n= prices.size();
    int dp1[n],dp2[n];

    dp1[0]=0;
    int mnb= prices[0];
    for(int i=1;i<n;i++){
        dp1[i]= max(dp1[i-1],prices[i]-mnb);
        mnb= min(mnb,prices[i]);
    }

    dp2[n-1]=0;
    int mxs=prices[n-1];
    for(int i=n-2;i>=0;i--){
        dp2[i]= max(dp2[i+1],mxs-prices[i]);
        mxs= max(mxs,prices[i]);
    }

    int ans=dp1[n-1];
    for(int i=1;i<n;i++)
        ans= max(ans,dp1[i-1]+dp2[i]);

    return ans;
}
```

---

124. Binary Tree Maximum Path Sum

This ques is DP on tree: There is another imp variation max path sum from leaf to another leaf question is available on gfg

TRICK : make decision on ans and what to return at every node

```
int ans;
    int foo(TreeNode* root)
    {
        if(root==NULL) return 0;

        int lans= foo(root->left);
        int rans= foo(root->right);

        ans= max(ans,root->val+lans+rans);
        ans= max(ans,root->val+max(0,max(lans,rans)));

        return max(root->val,root->val+max(lans,rans));
    }
    int maxPathSum(TreeNode* root) {

        ans=INT_MIN;
        foo(root);
        return ans;
    }
```

---

152. Maximum Product Subarray

Important problem revise it thoroughly version 1 --> TLE prefix the multiplication O(n^2) complexity

```
int n= nums.size();
long long dp[n+1];
dp[0]=1;
int idx0[n+1];
idx0[0]=-1;
//dp[i]==> represents the multication till i handling the 0 case
//ct0[i] ==> represents the idx of last zero till i
for(int i=1;i<=n;i++){
    if(nums[i-1]==0){
        dp[i]=1;
        idx0[i]=i;
    }
    else{
        dp[i]=dp[i-1]*nums[i-1];
        idx0[i]=idx0[i-1];
    }
}

long long ans= -1e15*1LL;
for(int i=1;i<=n;i++){
    for(int j=i;j>=1;j--){
        if(idx0[i]<j){
            long long temp= dp[i]/dp[j-1];
            // cout<<temp<<endl;
            ans= max(ans,temp);
        }
    }
    ans= max(ans,1LL*nums[i-1]);
}
return ans;
```

version2: DP solution : O(n) complexity **Think similar to kadane's to get the crux of this problem**

```
int n= nums.size();
        ll ans= -1e15*1LL;
        ll maxpos=1,maxneg=1;

        for(int i=0;i<n;i++){
            if(nums[i]==0){
                //reset
                maxpos=1,maxneg=1;
                ans= max(ans,0*1LL);
            }
            else if(nums[i]>0){
                ans= max(ans, maxpos*nums[i]);
                maxpos= maxpos*nums[i];
                maxneg= maxneg*nums[i];
            }
            else{
                ans= max(ans,maxneg*nums[i]);
                ll temp= maxpos;
                maxpos= max(1*1LL,maxneg*nums[i]);
                maxneg= min(1*1LL,temp*nums[i]);
            }
            //cout<<maxpos<<" "<<maxneg<<" "<<ans<<endl;
        }
        return ans;
```

---

188. Best Time to Buy and Sell Stock IV

int maxProfit(int k, vector& prices) {

```
    int n= prices.size();
    if(n==0)
        return 0;

    int dp[k+1][n];

    for(int i=0;i<=k;i++){
        int mx= -1e7*1LL;
        for(int j=0;j<n;j++){
            if(i==0)
                dp[i][j]=0;
            else if(j==0)
                dp[i][j]=0;

            else{
                int cost= prices[j]+mx;
                dp[i][j]= max(cost,dp[i][j-1]);
            }
            if(i>0) mx= max(mx,dp[i-1][j]-prices[j]);
        }
    }
    // for(int i=0;i<=k;i++){
    //     for(int j=0;j<n;j++){
    //         cout<<dp[i][j]<<" ";
    //     }
    //     cout<<'\n';
    // }
    return dp[k][n-1];
}
```

221. Maximal Square

comments are in the code for explaination

```
int maximalSquare(vector<vector<char>>& matrix) {

        int n= matrix.size();
        int m= matrix[0].size();

        int dp[n][m];

        //dp[i][j] --> represents the size of largest square with all 1's and endi
ng
        //at i,j
        //matrix[i][j]==0 ===>   dp[i][j]=0
        //matrix[i][j]==1 ===>   dp[i][j]= 1+min(dp[i-1][j],dp[i][j-1],dp[i-1][j-
1])
        int ans=0;

        for(int i=0;i<n;i++){
            for(int j=0;j<m;j++){
                if(i==0||j==0)
                    dp[i][j]= matrix[i][j]-'0';
                else{
                    if(matrix[i][j]=='0')
                        dp[i][j]=0;
                    else
                        dp[i][j]= 1+min({dp[i-1][j],dp[i-1][j-1],dp[i][j-1]});
                }
                ans= max(ans,dp[i][j]);
            }
        }
        return ans*ans;
    }
```

229. Majority Element II

THIS IS BASES ON BOYER MOORE VOTING ALGORITHM

```
vector<int> majorityElement(vector<int>& nums) {

        int n= nums.size();
        //observation is that atmost 2 majority can be present
        int ct1=0,ct2=0;
        int ele1=INT_MAX,ele2=INT_MAX;

        for(int i=0;i<n;i++){
            if(nums[i]==ele1)
                ct1++;
            else if(nums[i]==ele2)
                ct2++;
            else if(ct1==0){
                ele1= nums[i];
                ct1++;
            }
            else if(ct2==0){
                ele2= nums[i];
                ct2++;
            }
            else
                ct1--,ct2--;
        }

        vector<int> res;
        //2nd pass
        int t=0;
        int p=0;
        for(int x: nums){
            if(x==ele1)
                t++;
            if(x==ele2)
                p++;
        }
        if(t>n/3)
            res.push_back(ele1);
        if(p>n/3)
            res.push_back(ele2);

        return res;
    }
```

264. Ugly Number II

This is solved using 2 approaches ***using priority queue*** start with 1 as 1st ugly number start generating next greedy numbers but greedily pick smallest ugly number(use Priority queue) time complexity >NlogN <N^2

```
int nthUglyNumber(int D) {

        //greedily pick smallest element so far and then expand it to other multip
les
        //check duplicates
        priority_queue<ll,vector<ll>,greater<ll>> pq;
        int a=2,b=3,c=5;
        pq.push(1);
        unordered_map<ll,bool> mp; //O(1) time lookup
        int ct=0;
        vector<int>res;
        while(ct<D){
            ll num= pq.top();
            pq.pop();
            if(mp[num]==true)
                continue;

            res.push_back(num);
            mp[num]=true;
            ct++;
            if(ct==D)
                return num;

            if(mp[num*a]==false);
                pq.push(num*a);
            if(mp[num*b]==false);
                pq.push(num*b);
            if(mp[num*c]==false);
                pq.push(num*c);
        }
        return 0;
    }
```

Approach 2 : DP use previously generated ugly numbers to genrate new ugly numbers This approach is not very intutive but can be understood

```
//greedy aapproach
        vector<int> vec(n);
        vec[0]=1;
        // 1st ugly number is 1
        //generate next ugly numbers in smallest pattern from previuos ugly number
s
        int i=0,j=0,k=0;
        for(int p=1;p<n;p++){

            vec[p]= min({vec[i]*2,vec[j]*3,vec[k]*5});
            if(vec[p]==vec[i]*2) i++;
            if(vec[p]==vec[j]*3) j++;
            if(vec[p]==vec[k]*5) k++;
        }
        return vec[n-1];
```

287. Find the Duplicate Number

rar le isko acche se aage peeche hote hi galat ho ja rha hain sol

```
int findDuplicate(vector<int>& nums) {

        //using floyd cycle detection algo
        int l= nums[0];
        int h= nums[0];

        do{
            l=nums[l];
            h=nums[nums[h]];
        }while(l!=h);

        l= nums[0];
        while(l!=h){
            l=nums[l];
            h=nums[h];
        }
        return l;
    }
```

307. Range Sum Query - Mutable

Notes on segment tree: max size of segment tree= 4N

time complexity to build segment tree: O(N); space complexity to build segment tree: O(N) time complexity per query: O(logN)

Why is the complexity of this algorithm O(logn)? To show this complexity we look at each level of the tree. It turns out, that for each level we only visit not more than four vertices. And since the height of the tree is O(logn), we receive the desired running time.

We can show that this proposition (at most four vertices each level) is true by induction. At the first level, we only visit one vertex, the root vertex, so here we visit less than four vertices. Now let's look at an arbitrary level. By induction hypothesis, we visit at most four vertices. If we only visit at most two vertices, the next level has at most four vertices. That trivial, because each vertex can only cause at most two recursive calls. So let's assume that we visit three or four vertices in the current level. From those vertices, we will analyze the vertices in the middle more carefully. Since the sum query asks for the sum of a continuous subarray, we know that segments corresponding to the visited vertices in the middle will be completely covered by the segment of the sum query. Therefore these vertices will not make any recursive calls. So only the most left, and the most right vertex will have the potential to make recursive calls. And those will only create at most four recursive calls, so also the next level will satisfy the assertion. We can say that one branch approaches the left boundary of the query, and the second branch approaches the right one.

Therefore we visit at most 4logn vertices in total, and that is equal to a running time of O(logn).

In conclusion the query works by dividing the input segment into several sub-segments for which all the sums are already precomputed and stored in the tree. And if we stop partitioning whenever the query segment coincides with the vertex segment, then we only need O(logn) such segments, which gives the effectiveness of the Segment Tree.

---

309. Best Time to Buy and Sell Stock with Cooldown

int maxProfit(vector& prices) {

```
    int n= prices.size();
    int dp[n+1][2];
    dp[0][1]=0;
    dp[1][0]=-prices[0];
    dp[1][1]=0;

    for(int i=2;i<=n;i++){
        dp[i][0]= max(dp[i-1][0],dp[i-2][1]+(-prices[i-1]));
        dp[i][1]= max(dp[i-1][1],dp[i-1][0]+prices[i-1]);
    }

    return dp[n][1];
}
```

---

435. Non-overlapping Intervals

Approach-1: Instead of finding the intervals to romove, we can find the maximum number of non overlapping intervals and then subtract it with total intervals ot get the answer ***same code as unweighted activity scheduling(greedy)***

```
static bool comp(vector<int> &a, vector<int> &b)
    {
        return a[1]<b[1];
    }

    int eraseOverlapIntervals(vector<vector<int>>& intervals) {

        //this is similar to maximum number of activities
        //or unweighted job scheduling
        int n= intervals.size();
        sort(intervals.begin(),intervals.end(),comp);
        int ans=1;
        int lst =intervals[0][1];

        for(int i=1;i<n;i++){
            if(intervals[i][0]>=lst){
                ans++;
                lst= intervals[i][1];
            }
        }
        return n-ans;
    }
```

DP solution Code same as LIS

```
//dp solution-1
        int n= intervals.size();
        sort(intervals.begin(),intervals.end());

        vector<int> dp(n,1);
        int ans=1;
        for(int i=1;i<n;i++){
            for(int j=i-1;j>=0;j--){
                if(intervals[i][0]>=intervals[j][1]){
                    dp[i]= max(dp[i],1+dp[j]);
                }
            }
            ans= max(ans,dp[i]);
        }
        return n-ans;
```

***Dp solution -2** Code same as weighted job scheduling

```
//dp solution-1
        int n= intervals.size();
        sort(intervals.begin(),intervals.end());

        vector<int> dp(n,1);
        int ans=1;
        for(int i=1;i<n;i++){
            int ct=1;
            for(int j=i-1;j>=0;j--){
                if(intervals[i][0]>=intervals[j][1]){
                    ct+=dp[j];
                    break;
                }
            }
            dp[i]= max(dp[i-1],ct);
        }
        return n-dp[n-1];
```

556. Next Greater Element III

```
//  12345018760  ==> 12345068710  ==> 12345060178
        long long val= (1*1LL<<31);
        val--;

        string num= to_string(n);
        int len= num.length();
        for(int i=len-1;i>=0;i--){
            if(num[i]=='9')
                continue;

            int xx=10;
            int idx=-1;
            for(int j=i+1;j<len;j++){
                if(num[j]-'0' > num[i]-'0' && num[j]-'0'<xx){
                    xx= num[j]-'0';
                    idx=j;
                }
            }

            if(idx!=-1){
                swap(num[i],num[idx]);
                sort(num.begin()+i+1,num.end());
                break;
            }
        }

        long long req= stoll(num);
        if(req>val)
            return -1;
        if(req<=n)
            return -1;

        return (int)req;
    }
```

581. Shortest Unsorted Continuous Subarray

important question to learn concepts solution ranges from n^3 --> n^2 --> n important property of sorted array used ==> for an sorted array if seg is [L,R] min(L,R) is >= max(1,L-1) && max(L,R) <= min(R+1,N)

```
int findUnsortedSubarray(vector<int>& nums) {

        int n= nums.size();

        int l=0,r=n-1;

        while(l+1<n && nums[l]<=nums[l+1])
            l++;

        if(l==n-1)
            return 0;

        while(r-1>=0 && nums[r]>=nums[r-1])
            r--;

        // at this point l and r are at candidate bounds of unosorted array

        int mx=INT_MIN,mn=INT_MAX;
        for(int i=l;i<=r;i++){
            mx= max(mx,nums[i]);
            mn= min(mn,nums[i]);
        }

        //cout<<mx<<" "<<mn<<'\n';
        l--;r++;
        while(l>=0 && nums[l]>mn)
            l--;
        while(r<n && nums[r]<mx)
            r++;

        return r-l-1;
    }
```

This solution has O(n) tc and constant space

---

714. Best Time to Buy and Sell Stock with Transaction Fee

DP

```
 int n= prices.size();
        int dp[n][2];

        //dp[i][0]---> bought state i.e have one unsold stock in hand till i
        //dp[i][1]---> sold state i.e done with 0 or more complete transaction til
l i

        dp[0][0]=-prices[0] ,dp[0][1]=0;

        for(int i=1;i<n;i++){
            // buy stock
            int netcost= dp[i-1][1]+(-1*prices[i]);
            dp[i][0]= max(dp[i-1][0],netcost);
            //sell stock
            int nt= prices[i]+ dp[i-1][0]-fee;
            dp[i][1]= max(dp[i-1][1],nt);
        }

        return dp[n-1][1];
    }
```

1024. Video Stitching

THESE ARE IMPORTANT PROBLEMS ASKES IN INTERVIEWS The following problems has similar concepts:

1. Jump Game 2
2. Minimum number of taps to open to water the garden https://leetcode.com/problems/minimum-number-of-taps-to-open-to-water-a-garden/ (https://leetcode.com/problems/minimum-number-of-taps-to-open-to-water-a-garden/)
3. Video stiching

ALL the problems can be solves using greedy or dp

Greedy O(nlongn+ NT)

```
int n= clips.size();
        int ans=0;
        int mxrange=0;
        int mnrange=0;
        sort(clips.begin(),clips.end());
        while(mxrange<T){
            int idx=0;
            while(idx<n && clips[idx][0]<=mnrange){
                mxrange= max(mxrange,clips[idx][1]);
                idx++;
            }

            if(mxrange==mnrange)
                return -1;
            ans++;
            mnrange=mxrange;
        }

        return ans;
```

DP solution (nlogn+NT)

```
sort(clips.begin(),clips.end());
        int n= clips.size();
        //at max we can use N clips
        vector<int> dp(101,101);
        dp[0]=0; // no clips required to have a total frame of len 0

        for(auto c: clips){
            int start= c[0];
            for(int i=1;i<=c[1];i++){
                dp[i]= min(dp[i],1+dp[c[0]]);
            }
        }
        return dp[T]>=10
```

1074. Number of Submatrices That Sum to Target

O(n*mn*m) solution using prefix sum in a matrix

```
//lets  try O(n*m*n*m) solution
       //using prefix sum

       int n= matrix.size();
       int m= matrix[0].size();

       ll dp[n+1][m+1];
       memset(dp,0,sizeof(dp));

       for(int i=1;i<=n;i++){
           for(int j=1;j<=m;j++){
               if(i==1 && j==1)
                   dp[i][j]= matrix[i-1][j-1];
               else if(i==1)
                   dp[i][j]= matrix[i-1][j-1]+dp[i][j-1];
               else if(j==1)
                   dp[i][j]= matrix[i-1][j-1]+dp[i-1][j];
               else{
                   dp[i][j]= dp[i-1][j]+dp[i][j-1]-dp[i-1][j-1]+matrix[i-1][j-1];
               }
           }
       }

       //now prefix is complete
       //check target sum
       int ans=0;
       for(int i=1;i<=n;i++){
           for(int j=1;j<=m;j++){
               for(int a=i;a<=n;a++){
                   for(int b=j;b<=m;b++){
                       ll sm=0;
                       sm= dp[a][b]-dp[i-1][b]-dp[a][j-1]+dp[i-1][j-1];
                       if(sm==target)
                           ans++;
                   }
               }
           }
       }
       return ans;
```

O(n*n*m) using hasing method

```
int numSubmatrixSumTarget(vector<vector<int>>& matrix, int target) {


    int n= matrix.size();
    int m= matrix[0].size();

    //using o(n*n*m) method using hasing
    for(int i=0;i<n;i++)
        for(int j=1;j<m;j++)
            matrix[i][j]+=matrix[i][j-1];

    int ans=0;
    unordered_map<int,int> mp;
    for(int i=0;i<n;i++){
        for(int j=i;j<n;j++){
            mp.clear();
            mp[0]=1;
            for(int k=0;k<m;k++){
                matrix[i][k]+= (i==j)?0:matrix[j][k];
                ans+= mp[matrix[i][k]-target];
                mp[matrix[i][k]]++;
            }
        }
    }
    return ans;
}
```

NOTE: Runtime of 1st solution is better than of second solution beacause practically 1st is taking less than than the upper bound

---

1130. Minimum Cost Tree From Leaf Values

How to identify DP in this problem? ans: because ques asks for optimal binary tree which gives the solution as making all binary tree is not feasible therefore to find the optimal solution we have to use dp

which type of dp? ans: In this question we have to find solution for the best solution in [L,R] so this is dp on segments solved by filling the table in diagonal manner.

```
int n= arr.size();

        //we will solve it using dp
        int dp[n][n];
        //This dp is called dp on segments
        for(int j=0;j<n;j++){
            for(int i=0,k=j;i<n && k<n;i++,k++){
                if(k-i==0)
                    dp[i][k]=0;
                else if(k-i==1)
                    dp[i][k]=arr[i]*arr[k];
                else{
                    dp[i][k]=INT_MAX;
                    for(int p=i;p<k;p++){
                        //cal max in [i,p] ans [p+1][k];
                        // int mx1= *max_element(arr.begin()+i,arr.begin()+p+1);
                        // int mx2= *max_element(arr.begin()+p+1,arr.begin()+k+1);
                        int mx1= INT_MIN,mx2= INT_MIN;
                        for(int a=i;a<=p;a++)
                            mx1= max(mx1,arr[a]);
                        for(int a=p+1;a<=k;a++)
                            mx2= max(mx2,arr[a]);
                        // cout<<mx1<<" "<<mx2<<endl;
                        dp[i][k]= min(dp[i][k],dp[i][p]+dp[p+1][k]+ mx1*mx2);
                    }
                }
            }
        }
        // for(int i=0;i<n;i++){
        //     for(int j=0;j<n;j++){
        //         cout<<dp[i][j]<<" ";
        //     }
        //     cout<<endl;
        // }
        return dp[0][n-1];
    }
```

1326. Minimum Number of Taps to Open to Water a Garden

This is the type of question that needed to remebered This is a little similer to jump game II where we use ladder stairs approach

only variation is along with maximizing right bound we have to ensure left bound is also maintained

```
//n^2 + nlogn solution
        vector<array<int,2>> vec;
        for(int i=0;i<=n;i++){
            vec.push_back({max(0,i-ranges[i]),min(n,i+ranges[i])});
        }

        sort(vec.begin(),vec.end());

        int mxrange=0;
        int mnrange=0;
        int idx=0;
        int ans=0;
        while(idx<=n && mxrange<n){

            int sp;
            for(int i=idx;i<=n && vec[i][0]<=mnrange;i++){
                if(vec[i][1]>mxrange){
                    mxrange= vec[i][1];
                    sp=i;
                }
            }

            if(mxrange<=mnrange)
                return -1;

            mnrange= mxrange;
            idx=sp+1;
            ans++;
        }
        return ans;
```

ANOTHER APPROACH

one awesome trick to solve this problem is to convert the question in min jump array question the fountains can be converted into jumps for [L,R] is fountain range then the we can image it as jump from L to R and , ans is minimum number of jums also consider the case when the end is not reachable

```
    int sz= ranges.size();
    vector<int> jumps(n+1,0);

    for(int i=0;i<=n;i++){
        int l= max(0,i-ranges[i]);
        int r= min(n,i+ranges[i]);

        jumps[l]= max(jumps[l],r-l);
    }

    for(int i=0;i<=n;i++)
        cout<<jumps[i]<<" ";
    cout<<endl;
    //run the ladder stairs loop
    int ans=0;
    int mxrange=0;
    int idx=0;
    while(idx<n){
        mxrange= max(mxrange,idx+jumps[idx]);
        int steps= mxrange-idx;
        //cout<<mxrange<<" "<<idx<<endl;
        if(steps==0)
            return -1;
        else
            ans++;

        int tar=mxrange;
        while(idx<tar){
            mxrange= max(mxrange,idx+jumps[idx]);
            idx++;
        }
    }
    //cout<<mxrange<<endl;
    return ans;
}
```

PLEASE CRAMP THIS FOR OR RE ITERATE JUMP GAME II BECAUSE THIS IS TRICKY AND THERE IS POSSOBILITY THAT YOU MIGHT GET STRUCK AT LOOP
