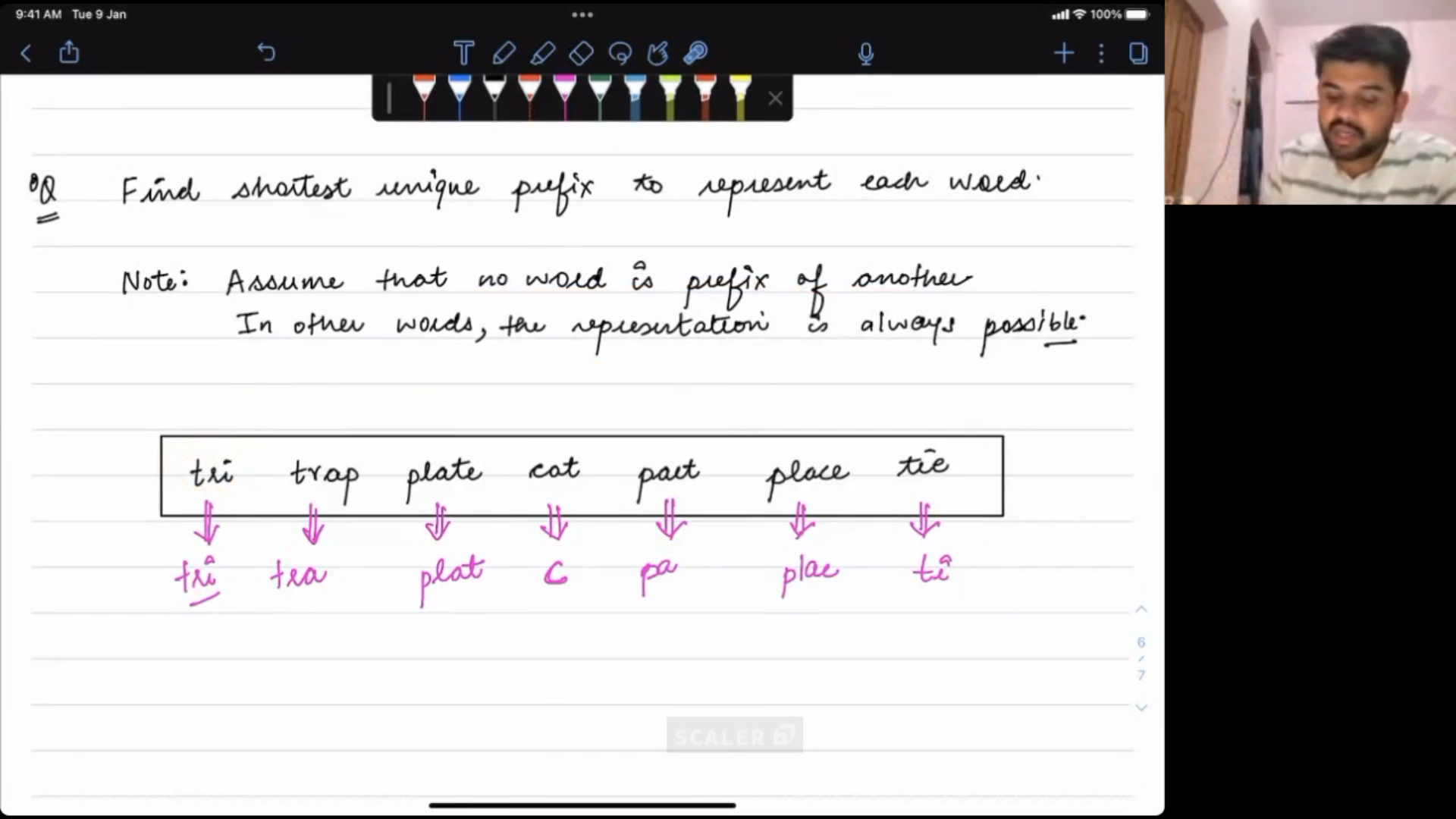
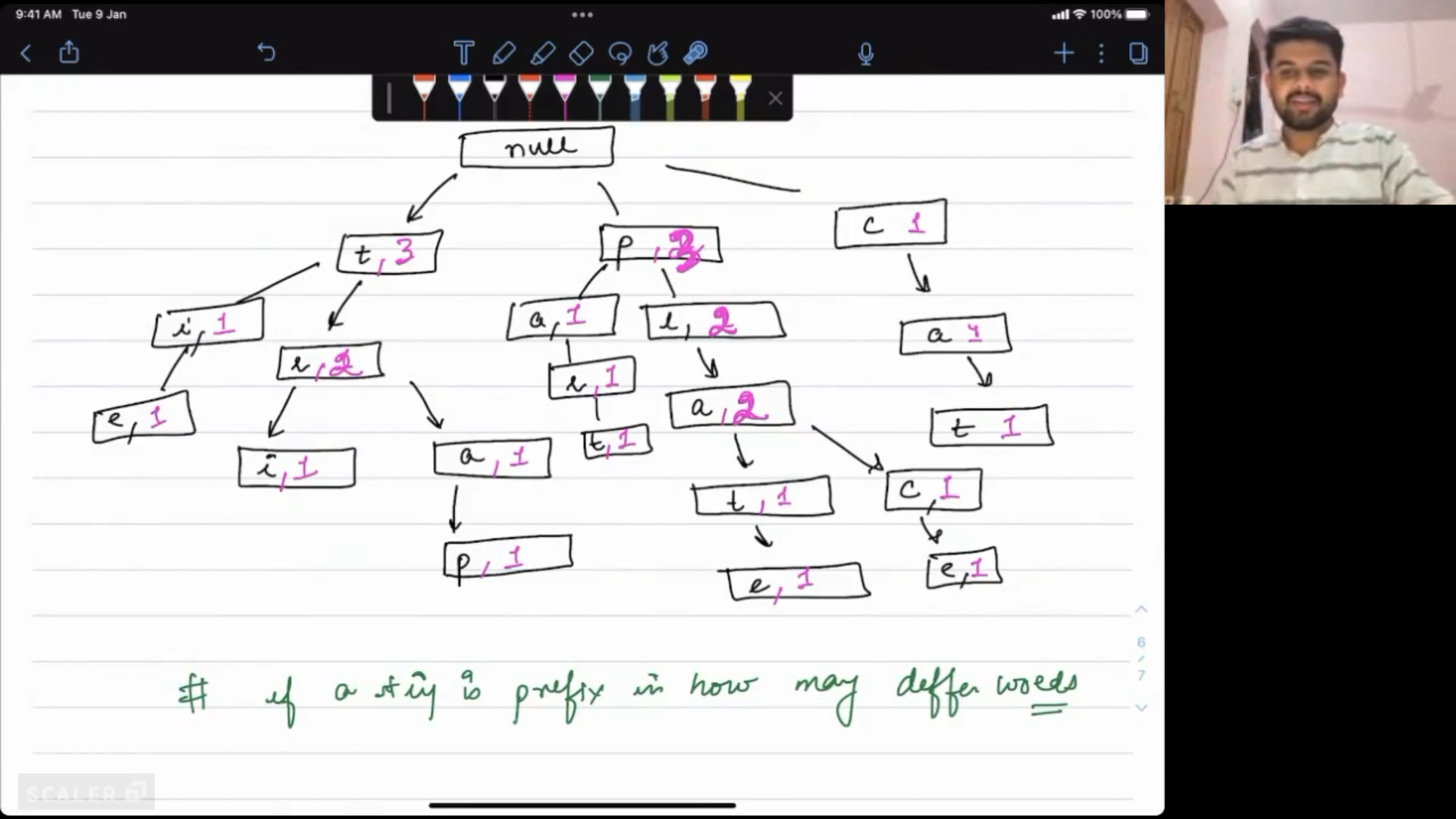
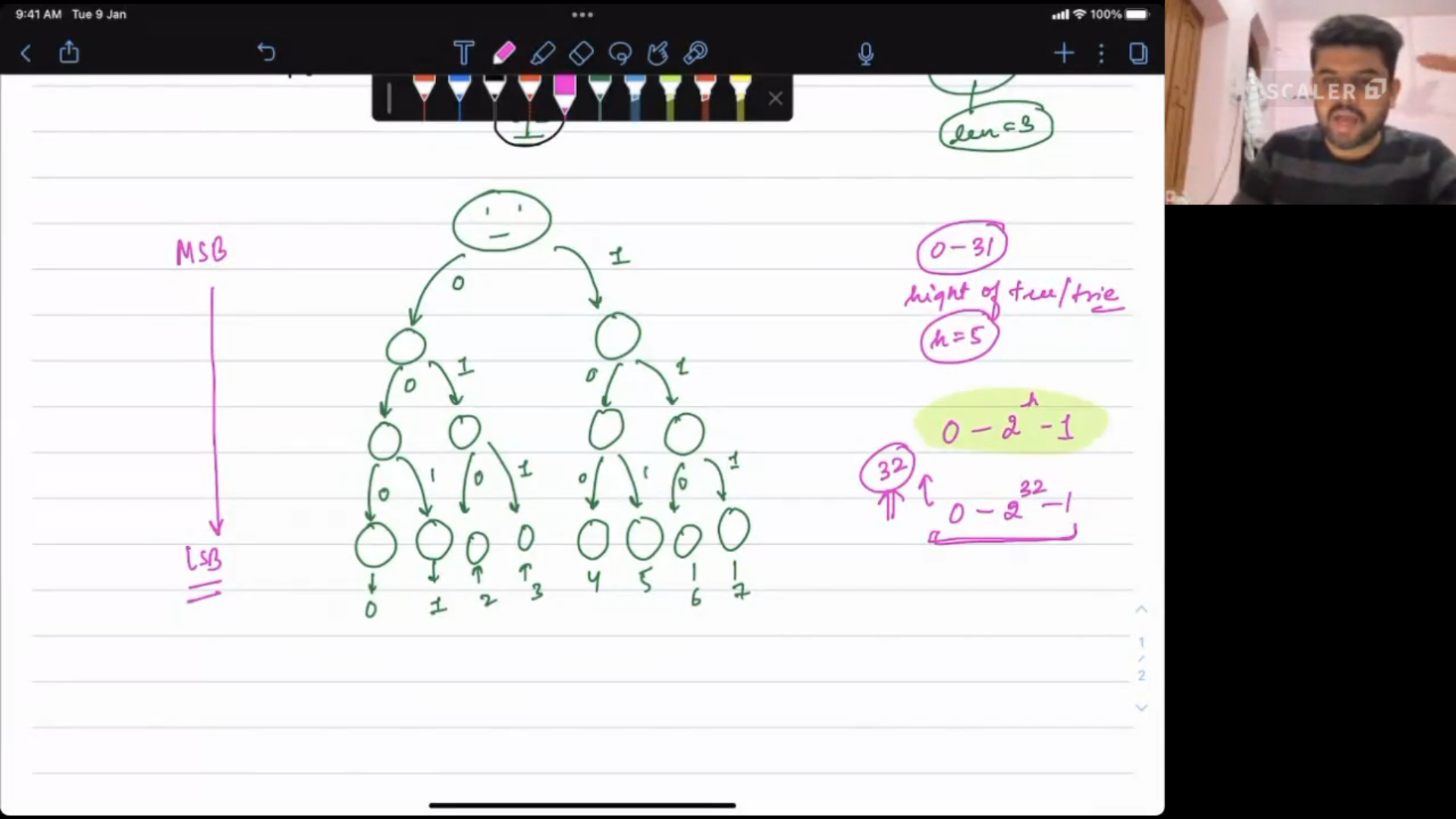
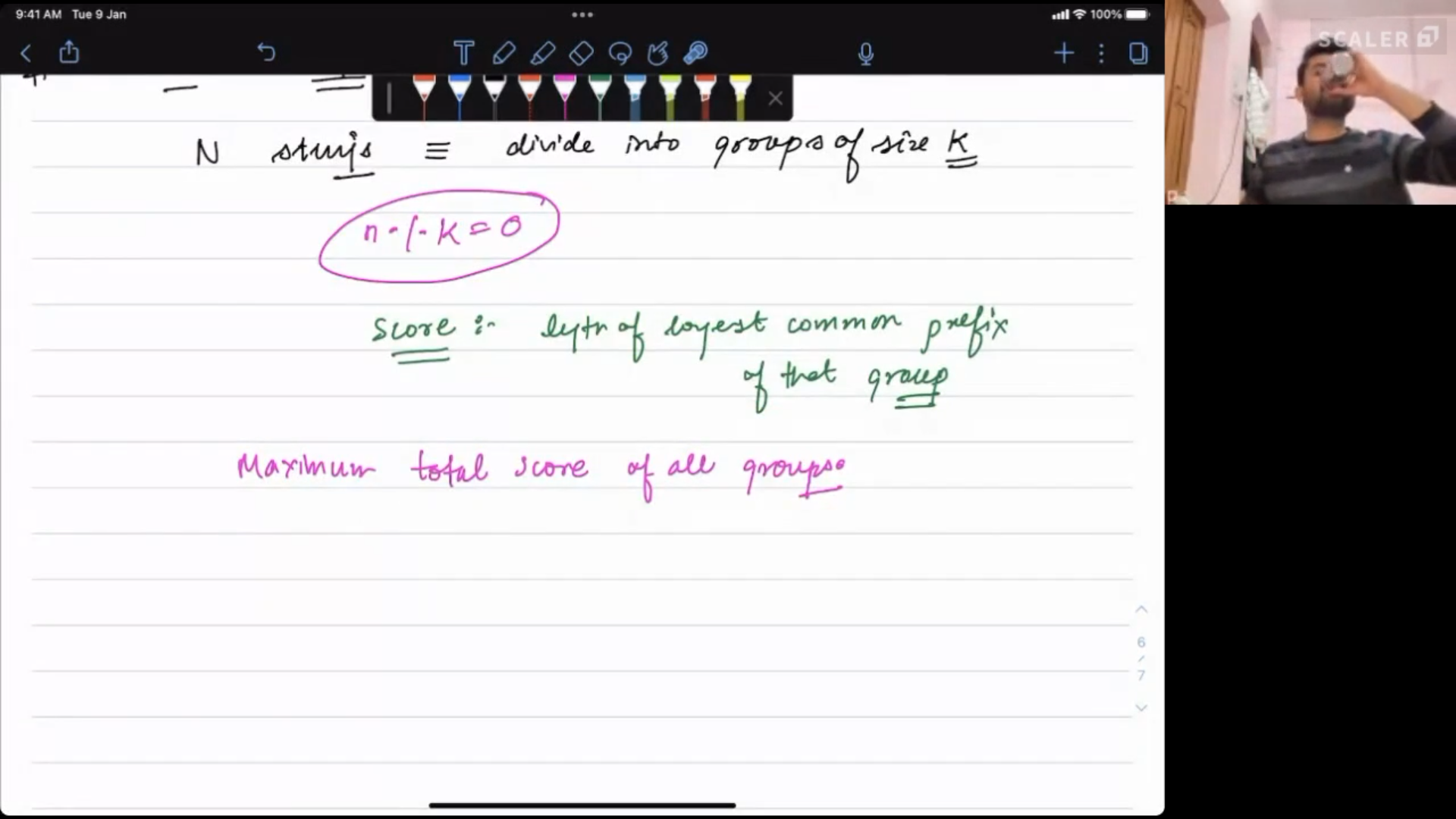
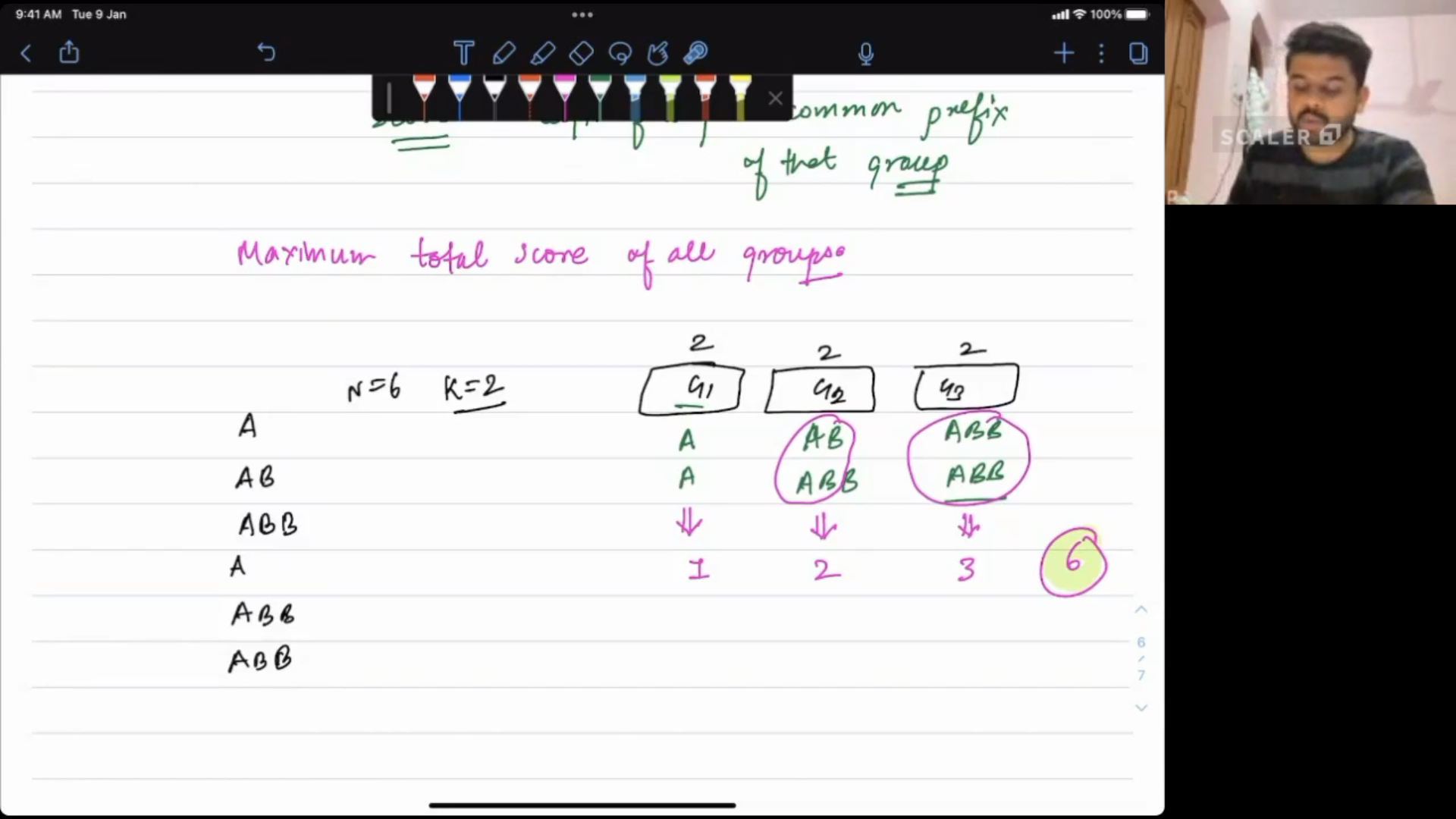
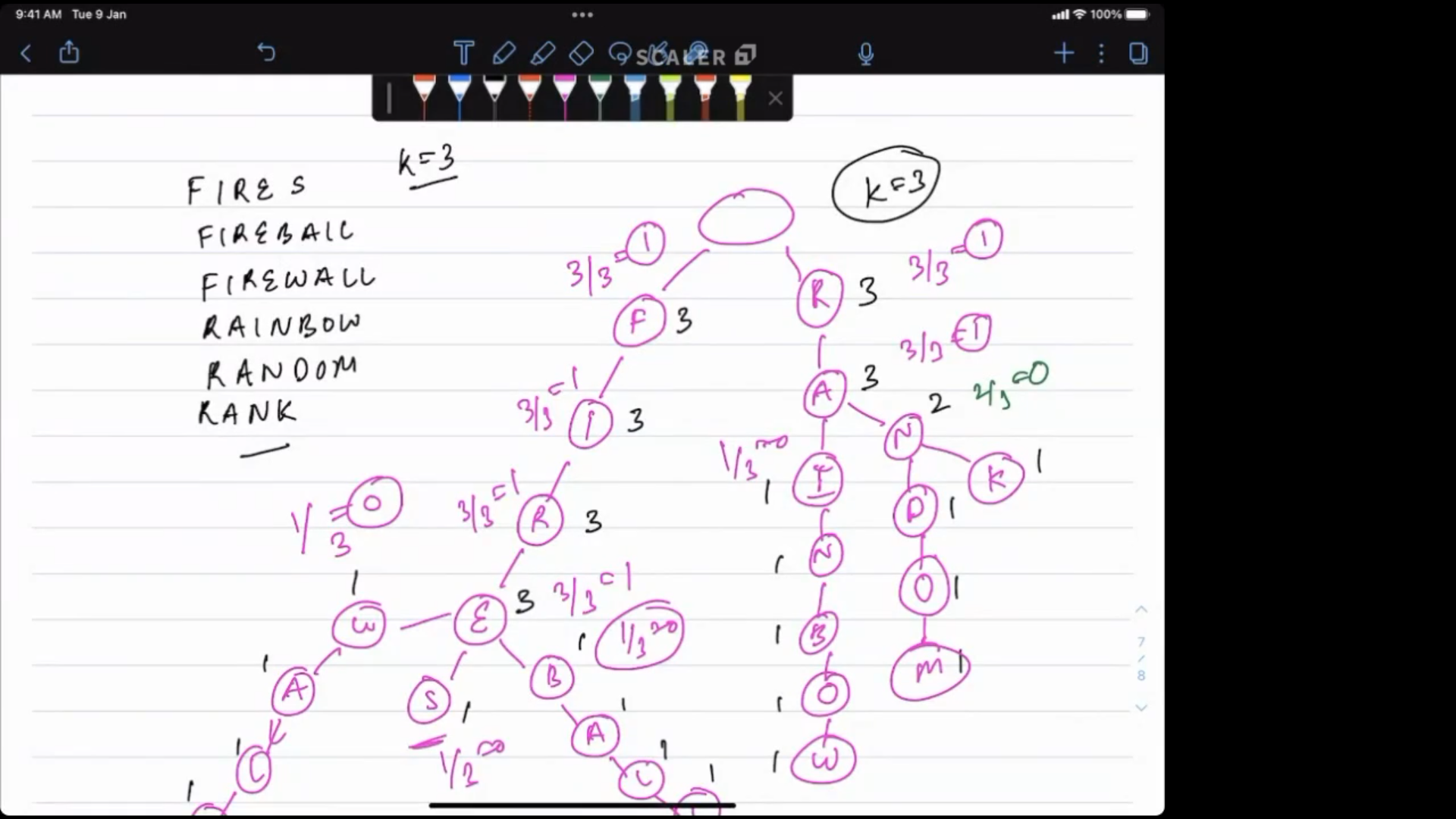
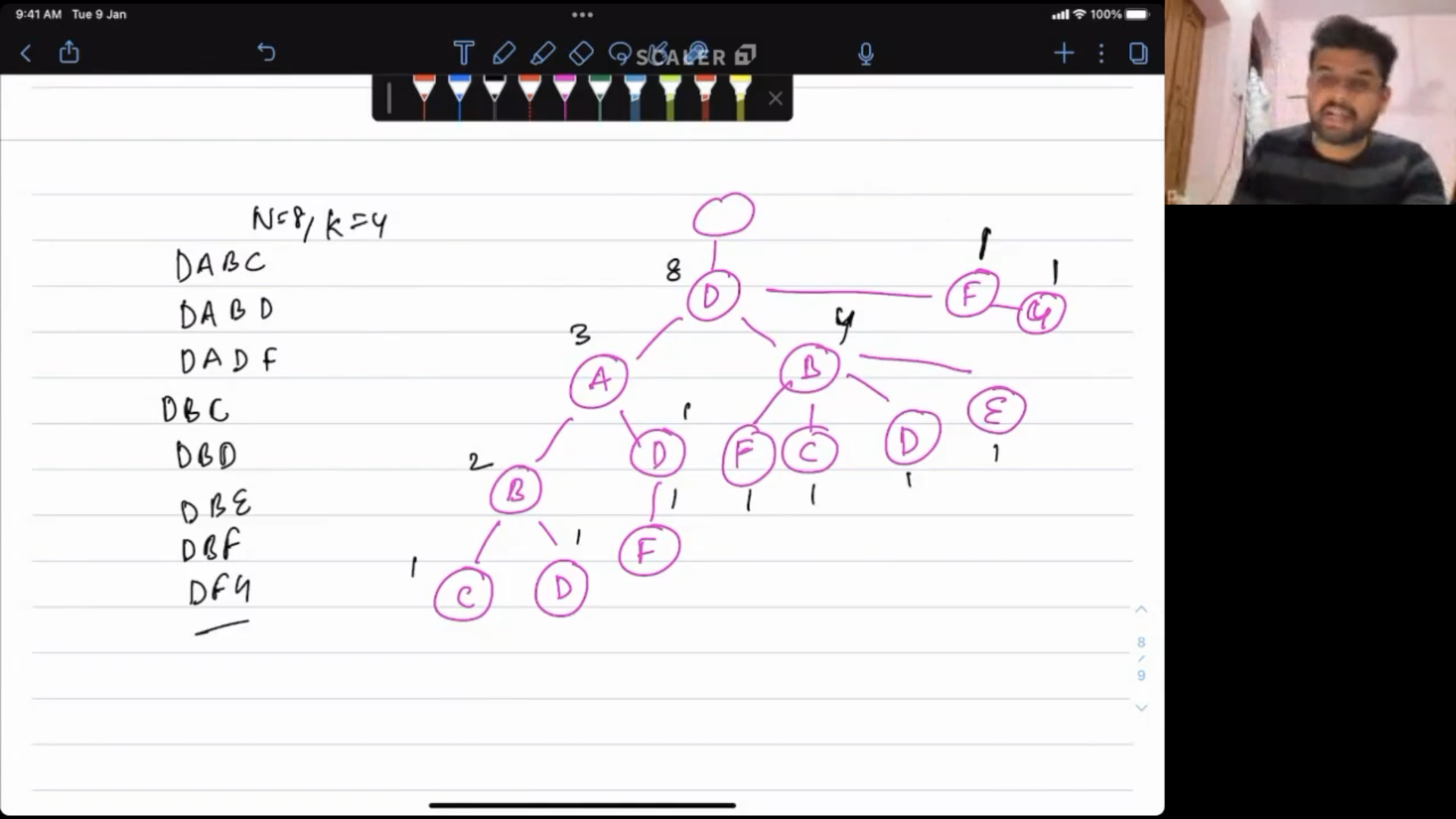
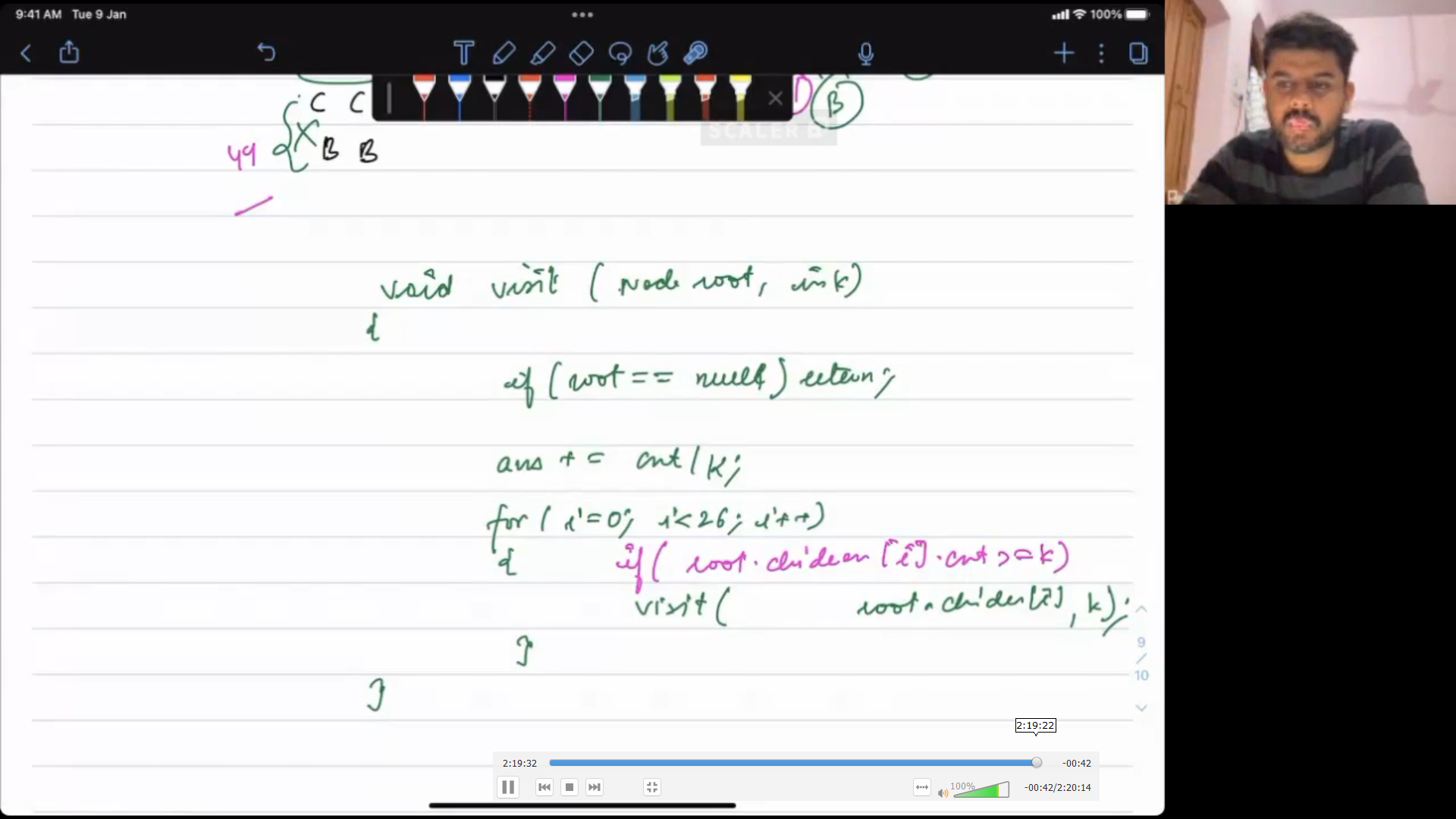
//TRIES

  
  
TRIE ON BITS🡪  
  
for(int i:a){

If(xorVal^i is present in trie) found a pair {i, xorVal^i }

Else Insert I into trie

}

**Maximum subarray xor**🡪Subarray i to j with max. value of prefixXor[j]^prefixXor[i-1]. Ans=max(ans, prefixXor[j]^prefixXor[i-1])  
  
  
given an array of size=n, divide it into n/k groups, each of size=k such that their score is max.  
Score=Sum of (Longest common prefix size of each group)  
  
Build this trie with cnt attribute  
  


// SEGMENT TREES

#include <bits/stdc++.h>

using namespace std;

int seg[100005],seg[4\*100005];

void build(int ind, int low, int high)

{

if(low==high){

seg[ind]=a[low];

return;

}

int mid=(low+high)>>1;

build(2\*ind+1,low,mid);

build(2\*ind+2,mid+1,high);

seg[ind]=max(seg[2\*ind+1],seg[2\*ind+2]); //seg[ind]=seg[2\*ind+1]+seg[2\*ind+2];

}

int query(int ind, int low, int high, int l, int r) //Range l-r

{

if(low>=l && high<=r){

return seg[ind];

}

if(low<r> || high<l){

return INT\_MIN;

}

int mid=(low+high)>>1;

int left = query(2\*ind+1,low,mid,l,r);

int right = query(2\*ind+2,mid+1,high,l,r);

return max(left,right);

}

//single point update-->update(0,0,n-1,node,val)

void pointUpdate(int ind, int low, int high, int node, int val){

if(low==high){

seg[low]+=val;

return;

}

int mid=(low+high)>>1;

if(node<=mid && node>=low) pointUpdate(2\*ind+1,low,mid,node,val);

else pointUpdate(2\*ind+2,mid+1,high,node,val);

seg[ind]=seg[2\*ind+1]+seg[2\*ind+2];

}

void rangeUpdate(int ind, int low, int high, int l, int r, int val) //Range l-r

{

if(lazy[ind]!=0){

seg[ind]+=(high-low+1)\*lazy[ind];

if(low!=high){

lazy[2\*ind+1]+=lazy[ind];

lazy[2\*ind+2]+=lazy[ind];

}

lazy[ind]=0;

}

if(r<low || l>high || low>high) return 0;

if(low>=l && high<=r){

seg[ind]+=(high-low+1)\*val;

if(low!=high){

lazy[2\*ind+1]+=lazy[ind];

lazy[2\*ind+2]+=lazy[ind];

}

return;

}

int mid=(low+high)>>1;

rangeUpdate(2\*ind+1,low,mid,l,r,val);

rangeUpdate(2\*ind+2,mid+1,high,l,r,val);

seg[ind]=seg[2\*ind+1]+seg[2\*ind+2];

}

int querySumLazy(int ind, int low, int high, int l, int r, int val)

{

if(lazy[ind]!=0){

seg[ind]+=(high-low+1)\*lazy[ind];

if(low!=high){

lazy[2\*ind+1]+=lazy[ind];

lazy[2\*ind+2]+=lazy[ind];

}

lazy[ind]=0;

}

if(r<low || l>high || low>high) return 0;

if(low>=l && high<=r){

return seg[ind];

}

int mid=(low+high)>>1;

return querySumLazy(2\*ind+1,low,mid,l,r,val) + querySumLazy(2\*ind+2,mid+1,high,l,r,val);

}

int main(){

int n;

cin>>n;

for(int i=0;i<n;i++){

cin>>a[i];

}

build(0,0,n-1);

int q;

cin>>q;

while(q--){

int l,r;

cin>>l>>r;

cout<<query(0,0,n-1,l,r)<<endl;

}

}