BACKTRACKING COMBINATIONAL SUM

```
vector<vector<int>> ans;
  void solve(vector <int> &arr, int index, int sum , vector<int> curr)
  {
     if(sum < 0)
        return;
     if(sum == 0)
        ans.push_back(curr);
     for(int i = index ; i< arr.size(); i++)
     {
        curr.push_back(arr[i]);
        solve(arr, i, sum-arr[i] , curr);
        curr.pop_back();
     }
     return;
  }
   vector<vector<int> > combinationSum(vector<int> &A, int B) {
      // Your code here
       vector <int> curr;
     sort(A.begin(), A.end());
     auto itr = unique(A.begin(), A.end());
     A.resize(distance(A.begin(), itr));
     solve(A,0, B, curr);
     return ans;
   }
```

BINARY SEARCHABLE ELEMENT

an element in an array is binary searchable when
we subtract previous element of array from current array and when we subtract
the current element from next coming element of array ,if in both cases
result is positive then element is binary searchable array.

```
int binarySearchable(int Arr[], int n) {
       // code here
       int count=0;
     for(int i=0;i<n;i++){
```

```
        if(i!=0 && i!=n-1){
        if(Arr[i]-Arr[i-1]>0 && Arr[i+1]-Arr[i]>0)
        {
           count++;
        }
        }
        else if(i==0){
           if(Arr[i+1]-Arr[i]>0){
              count++;
           }
        }
        else if(i==n-1){
           if(Arr[i]-Arr[i-1]>0){
              count++;
           }
        }
     }
     return count;
  }
```

BACKTRACKING FIND ALL PALINDROMIC PARTITION OF A STRING

```
bool isPalin(string s){
    for(int i=0;i<s.size()/2;i++){
       if(s[i]!=s[s.size()-i-1])
       return false;
    }
    return true;
 }
 void solve(int ind,string s,vector<vector<string>>&ans,vector<string>&temp){
    if(ind==s.size()){
       ans.push_back(temp);
       return;
    }
    for(int i=ind;i<s.size();i++){
       string sub=s.substr(ind,i-ind+1);
       if(isPalin(sub)){
```

```
            temp.push_back(sub);
            solve(i+1,s,ans,temp);
            temp.pop_back();
        }
     }
  }
    vector<vector<string>> allPalindromicPerms(string s) {
      vector<vector<string>>ans;
       vector<string>temp;
       solve(0,s,ans,temp);
       return ans;
    }
```

BINARY SEARCH MEDIAN OF ROW WISE SORTED MATRIX

```
bool isPossible(vector<vector<int>> matrix, int r, int c, int n, int mid){
       int cnt = 0;

       for(int i=0;i<r;i++){
          int j = c - 1;
          if(mid >= matrix[i][j]){
             cnt += j + 1;
          } else {
             while(mid < matrix[i][j]) j--;
             cnt += j + 1;
          }
       }

       return cnt >= ((n + 1) / 2);
    }

    int median(vector<vector<int>> &matrix, int r, int c){
       // code here
       int s = INT_MAX, e = INT_MIN;

       int n = r * c;
       int ans = 0;
```

```
        for(int i=0;i<r;i++){
            s = min(s, matrix[i][0]);
            e = max(e, matrix[i][c-1]);
        }



        while(s <= e){
            int mid = s + (( e - s ) / 2);
            if(isPossible(matrix, r, c, n, mid)){
                ans = mid;
                e = mid - 1;
            } else {
                s = mid + 1;
            }
        }
        return ans;
    }
```

BACKTRACKING GRAPH M COLORING PROBLEM

```
bool isSafe(int node, int color[], bool graph[101][101],int n, int col)
{
    for(int k=0;k<n;k++)
    {
        if(k!=node  && graph[k][node]==1  &&  color[k]==col)
        return false;
    }
    return true;
}
bool solve(int node, int color[],int m, int n, bool graph[101][101])
{
    if(node==n)
    {
        return true;
    }
    else
    for(int i=1;i<=m;i++)
```

```
    {
        if(isSafe(node,color,graph,n,i))
        {
            color[node]=i;
            if(solve(node+1,color,m,n,graph))
            return true;
            color[node]=0;
        }
    }
    return false;
}
bool graphColoring(bool graph[101][101], int m, int n) {
    int color[n]={0};
    if(solve(0,color,m,n,graph))return true;
    return false;
}
```

BACKTRACKING N QUEEN

```
bool issafe(int row, int col,vector<vector<int>>&ans,vector<vector<int>>&board,int n)
    {
        int x=row,y=col;
        while(y>=0)
        {
            if(board[x][y]==1)
            {
                return false;
            }
            y--;
        }
        x=row, y=col;
        while(x>=0 && y>=0) {
            if(board[x][y]==1) {
                return false;
            }
            x--,y--;
        }
        x=row, y=col;
```

```
    while(x<n && y>=0) {
        if(board[x][y]==1) {
            return false;
        }
        x++,y--;
    }
    return true;
}
void addans(vector<vector<int>>&ans,vector<vector<int>>&board)
{
    vector<int>tans;
    int n=board.size();
    for(int i=0;i<n;i++)
    {
        for(int j=0;j<n;j++)
        {
            if(board[i][j]!=0)
            tans.push_back(j+1);
        }
    }
    ans.push_back(tans);
}
void solve(int col,vector<vector<int>>&ans,vector<vector<int>>&board,int n)
{
    if(col==n)
    {
        addans(ans,board);
        return;
    }
    for(int row=0;row<n;row++)
    {
        if(issafe(row,col,ans,board,n))
        {
            board[row][col]=1;
            solve(col+1,ans,board,n);
            board[row][col]=0;
        }
```

```
    }
  }
  vector<vector<int>> nQueen(int n) {
    vector<vector<int>>ans;
    vector<vector<int>>board(n,vector<int>(n,0));
    int src=0;
    solve(src,ans,board,n);
    sort(ans.begin(),ans.end());
    return ans;
  }
```

BACKTRACKING REMOVE INVAID PARENTHESIS

```
vector<string> res;
  unordered_map<string,int> mp;

  int getMinInValid(string s)
  {
    stack<char> stck;
    int i = 0;
    while(i < s.size())
    {
      if(s[i] == '(')
        stck.push('(');
      else if(s[i] == ')')
      {
        if(stck.size() > 0 && stck.top() == '(')
          stck.pop();
        else
          stck.push(')');
      }
      i++;
    }
    return stck.size();
  }

  void solve(string s,int minInv){
```

```
        if(mp[s] != 0)
            return;
        else
            mp[s]++; //mp[s] = 1
        //base case
        if(minInv < 0){
            return;
        }
        if(minInv == 0)
        {
            if(!getMinInValid(s))
                res.push_back(s);
            return;
        }

        for(int i=0; i<s.size(); i++)
        {
            string left = s.substr(0,i);
            string right = s.substr(i+1);
            solve(left+right, minInv-1);
        }
        return;
    }

    vector<string> removeInvalidParentheses(string s) {
        // code here
        solve(s, getMinInValid(s));
        sort(res.begin(),res.end());
        return res;
    }
```

BACKTRACKING SUDOKU SOLVER

```
class Solution
{
    public:
    #define UNASSIGNED 0
```

```
#define N 9
bool UsedInRow(int grid[N][N], int row, int num)
{
   for (int col = 0; col < N; col++)
      if (grid[row][col] == num)
         return true;
   return false;
}
bool UsedInCol(int grid[N][N], int col, int num)
{
   for (int row = 0; row < N; row++)
      if (grid[row][col] == num)
         return true;
   return false;
}
bool UsedInBox(int grid[N][N], int boxStartRow,
        int boxStartCol, int num)
{
   for (int row = 0; row < 3; row++)
      for (int col = 0; col < 3; col++)
         if (grid[row + boxStartRow]
               [col + boxStartCol] ==
                           num)
            return true;
   return false;
}
bool isSafe(int grid[N][N], int row,
      int col, int num)
{
   return !UsedInRow(grid, row, num)
       && !UsedInCol(grid, col, num)
      && !UsedInBox(grid, row - row % 3,
                col - col % 3, num)
      && grid[row][col] == UNASSIGNED;
}
bool FindUnassignedLocation(int grid[N][N],
                  int& row, int& col)
```

```
{
    for (row = 0; row < N; row++)
        for (col = 0; col < N; col++)
            if (grid[row][col] == UNASSIGNED)
                return true;
    return false;
}
bool SolveSudoku(int grid[N][N])
{
    // Your code here
    int row, col;
if (!FindUnassignedLocation(grid, row, col))
    return true;
for (int num = 1; num <= 9; num++)
{
    if (isSafe(grid, row, col, num))
    {
        grid[row][col] = num;
        if (SolveSudoku(grid))
            return true;
        grid[row][col] = UNASSIGNED;
    }
}
return false;
}

void printGrid (int grid[N][N])
{
    for (int row = 0; row < N; row++)
{
    for (int col = 0; col < N; col++)
        cout << grid[row][col] << " ";
}
}
```

CHECK IS SUDOKU VALID

```
int isValid(vector<vector<int>> mat){
```

```
    // code here
    map<string,int>mp;


   for(int i=0;i<9;i++)
   {
      for(int j=0;j<9;j++)
      {
         int box = (i/3)*3+(j/3);
         if(mat[i][j]!=0)
         {
            string s1 ="row" + to_string(i) +to_string(mat[i][j]);
            string s2 ="col" + to_string(j) + to_string(mat[i][j]);
            string s3 ="box" + to_string(box) + to_string(mat[i][j]);
            mp[s1]++;
            mp[s2]++;
            mp[s3]++;
            if(mp[s1]>1 || mp[s2]>1 || mp[s3]>1)
            {
               return 0;
            }
         }
      }
   }

   return 1;
 }
```

PARTITION ARRAY TO K SUBSETS

```
bool solve(int sum,int n,vector<int>& vis,int csum,int a[],int k)
 {
   if(k==1)
   {
      return true;
   }
   if(csum>sum)return false;
   if(csum==sum)
   {
```

```
        return solve(sum,n,vis,0,a,k-1);
      }
      for(int i=0;i<n;i++)
      {
        if(!vis[i])
        {
          vis[i]=1;
          if(solve(sum,n,vis,csum+a[i],a,k))return true;
          vis[i]=0;
        }
      }
      return false;
    }
    bool isKPartitionPossible(int a[], int n, int k)
    {
       //Your code here
        int sum=accumulate(a,a+n,0);
      if(k>n || sum%k!=0)return false;
      vector<int> vis(n,0);
      sum/=k;
      return solve(sum,n,vis,0,a,k);

    }

    BST FIX TWO NODES OF A BST
Node* prev =NULL;
 Node* first = NULL;
 Node* second = NULL;
 void inorder(Node* root){
    if(root==NULL){
       return;

    }
    inorder(root->left);
    if(prev != NULL && root->data < prev->data){
       if(first == NULL){
          first = prev;
```

```
        }
        second = root;
    }
    prev = root;
    inorder(root->right);
}

  void correctBST( struct Node* root )
  {
      // add code here.
      if(root==NULL){
          return;
      }
      inorder(root);
      int temp = first->data;
      first->data = second->data;
      second->data =temp;
  }
```

BST TO GREATER SUM TREE

```
void fun(Node * root,int &sum)
  {
      if(!root) return;
      fun(root->right,sum);
      int t=root->data;
      root->data=sum;
      sum=sum+t;
      fun(root->left,sum);
      return;
  }
  void transformTree(struct Node *root)
  {
      //code here
      int sum = 0;
      fun(root,sum);
      return;
  }
```

CELEBRITY PROBLEM

```
int celebrity(vector<vector<int> >& M, int n)
   {
       // code here
       int i=0;
      int j = n-1;

      while(i<j)
      {
         if(M[j][i]==1)
            j--;
         else
            i++;
      }
      int candi =i;

      for(i=0;i<n;i++)
      {
         if(i!=candi)
         {
         if(M[i][candi]==0 || M[candi][i]==1)
            return -1;
         }
      }
      return candi;
   }
```

CHECK N-ARRAY TREE

```
int checkMirrorTree(int n, int e, int A[], int B[]) {
       unordered_map<int,stack<int>> m;
      for(int i=0;i<e*2;i=i+2)
      {
         m[A[i]].push(A[i+1]);
      }
      for(int i=0;i<2*e;i+=2)
      {
```

```
        if(m[B[i]].top()!=B[i+1])
        return 0;

        m[B[i]].pop();
      }
      return 1;
  }
```

CONTAINER WITH MOST WATER

```
long long maxArea(long long A[], int len)
{
   // Your code goes here
   long long res = 0;
  int i=0,j =len-1;
  while(i<=j)
  {
    res = max(res,(j-i)*min(A[i],A[j]));
    if(A[i] < A[j])i++;
    else j--;
  }

  return res;
}
```

COUNT FACTORIAL OF A LARGE NO

```
vector<int> ans;
int fac(int num,int count){
   int carry=0;
   for(int i=0;i<count;i++){
      int val=ans[i]*num+carry;
      ans[i]=val%10;
      carry=val/10;
   }

   while(carry){
      ans.push_back(carry%10);
      carry/=10;
```

```
        count++;
    }
    return count;
}


    vector<int> factorial(int N){
        // code here
        ans.push_back(1);
        int count=1;
        for(int i=2;i<=N;i++)
            count=fac(i,count);
        reverse(ans.begin(),ans.end());
        return ans;
    }
int findNumberOfTriangles(int arr[], int n)
    {
        // code here
        sort(arr, arr + n);
        int ans = 0;
        for(int i = 0; i < n; i++){
            int k = i + 2;
            for(int j = i + 1; j < n - 1; j++){
                while(k < n && arr[k] < arr[i] + arr[j]){
                    k++;
                }
                ans += k - j - 1;
            }
        }
        return ans;
    }
```

COUNT OCCURANCE OF ANAGRAMS

```
int search(string pat, string txt) {
                // code here
                 int arr[26];
    int check[26];
    for(int i = 0; i<26; i++){
```

```
    arr[i] = 0;
    check[i] = 0;
}

for(int i = 0; i<pat.size(); i++){
    arr[pat[i]-'a']++;
}
int n = txt.size();
int i = 0;
int j = 0;
int flag;
int count = 0;
while(j<n){
    flag = 0;
    check[txt[j]-'a']++;
    if(j-i+1<pat.size()){
      //   check[txt[j]-'a']++;
        j++;
    }

   else if(j-i+1 == pat.size()){

        for(int k=0; k<26; k++){
            if(check[k]!=arr[k]){
                flag = 1;
            }
        }
        if(flag!=1){
            count++;
        }
        check[txt[i]-'a']--;
        i++;
        j++;
    }
}
return count;
        }
```

Count subscquence of type a^ib^jc^k

```
int fun(string &s) {
    //code here
    long long ans = 0 , mod = 1e9+7 , a = 0, ab = 0, abc = 0 ;
  for(auto c: s){
    if(c == 'a')
      a = (2*a + 1)%mod ;
    else if(c == 'b')
      ab = (2*ab + a)%mod ;
    else
      abc = (2*abc + ab)%mod ;
  }
  return abc ;
  }
```

DP EQUAL SUM PARTITION

```
int equalPartition(int N, int arr[])
  {
    // code here
    int sum=0;
    for(int i=0; i<N; i++) sum += arr[i];
    if(sum%2 != 0) return 0;
    sum/=2;
    vector<vector<bool>> dp(N+1, vector<bool>(sum+1));
    for(int i=0; i<N+1; i++){
      for(int j=0; j<sum+1; j++){
        if(j==0) dp[i][j] = true;
        else if(i==0) dp[i][j] = false;
        else{
          if(arr[i-1]>j) dp[i][j] = dp[i-1][j];
          else{
            dp[i][j] = dp[i-1][j] || dp[i-1][j-arr[i-1]];
          }
        }
      }
    }
```

```
    return dp[N][sum]? 1 : 0;
  }
```

DP HANDSHAKES

```
int count(int N){
    // code here
    vector<int>dp(N+1,0);
   dp[0]=1;
   for(int n=2;n<=N;n+=2){
      for(int i=0;i<=n-2;i+=2){
         dp[n]+=dp[i]*dp[n-2-i];
      }
   }
   return dp[N];
  }
```

LARGEST NO IN K SWAPS

```
public:
  void solve(string &max,string str, int k, int idx)
  {
     if(k==0)
     return;
     int n=str.length();
     char maxc=str[idx];
     for(int i=idx+1;i<n;i++)
     {
        if(maxc<str[i])
        maxc=str[i];
     }
     if(maxc!=str[idx])
     k-=1;
     for(int i=idx;i<n;i++)
     {
        if(str[i]==maxc){
        swap(str[i],str[idx]);
        if(str.compare(max)>0)
        max=str;
```

```
            solve(max,str,k,idx+1);
            swap(str[i],str[idx]);
            }
        }
    }
    string findMaximumNum(string str, int k)
    {
      // code here.
      string max=str;
      solve(max,str,k,0);
      return max;
    }
```

PAINTING THE FENCE

```
long long countWays(int n, int k){
        // code here
        long long int  mod = 1e9+7;
        long long int dp[n+1];
        dp[0] = 0;
        dp[1] = k;
        dp[2] = k*k;
        for(long int i=3;i<=n;i++)
        {
            dp[i] = ((k-1)*(dp[i-1] + dp[i-2]))%mod;
        }
        return dp[n];

    }
```

DP SHORTEST UNCOMMON SUBSCQUENCE

```
int shortestUnSub(string S, string T) {
        // code here
         int n=S.length(), m=T.length();

        int dp[n+1][m+1];

        int MAX = 600;
```

```
    for(int i=0;i<=n;i++) dp[i][0]=1;
    for(int i=0;i<=m;i++) dp[0][i]=MAX;

    for(int i=1;i<=n;i++) {

      for(int j=1;j<=m;j++) {

        int k;
        for(k=j-1;k>=0;k--) {
          if(T[k]==S[i-1]) break;
        }

        if(k<0) dp[i][j]=1;
        else {

          dp[i][j] = min(dp[i-1][j], dp[i-1][k]+1);

        }

      }

    }

    if(dp[n][m] >= MAX) return -1;
    return dp[n][m];
```

DP STACK NO OF PREVIOUS SMALLER ELEMENT INCLUDING ITSELF

```
vector <int> calculateSpan(int price[], int n)
  {
    // Your code here
    stack<int> st;
   vector<int> span(n,1);
   for(int i=0;i<n;i++){
      while(!st.empty() && price[st.top()]<=price[i]){
        span[i]+= span[st.top()];
```

```
            st.pop();
        }
        st.push(i);
    }
    return span;

}
```

DP TREE MAX SUM SUCH THAT NO TWO NODES ARE ADJACENT

```
unordered_map<Node* , int> dp;
int getMaxSum(Node *root)
{
    // Add your code here
    if(!root) return 0;

    if(dp[root]) return dp[root];

    int inc = root->data;
    if(root->left){
        inc += getMaxSum(root->left->left);
        inc += getMaxSum(root->left->right);

    }
    if(root->right){
        inc += getMaxSum(root->right->left);
        inc += getMaxSum(root->right->right);

    }

    int exc = getMaxSum(root->left) + getMaxSum(root->right);

    dp[root] = max(inc , exc);

    return dp[root];
```

FIND A STRING IN A GRID

```
bool search(vector<vector<char>>grid, string word,int index, int r, int c, int m, int n,int dir_r,int dir_c)
```

```
{
   if(index==word.length())
   {
      return true;
   }
   else if(r<0 ||r>m-1 || c<0 ||c>n-1)
    {
       return false;
    }
   if(grid[r][c]==word[index])
   {
      return search(grid,word,index+1,r+dir_r,c+dir_c,m,n,dir_r,dir_c);
   }
   return false;
}


        vector<vector<int>>searchWord(vector<vector<char>>grid, string word){
           // Code here
           int m=grid.size();
   int n=grid[0].size();
   vector<vector<int>>res;
   //right, left,down,up,right-up,left-up,left-down,right-down}
   int x[]={1,-1,0, 0, 1,-1,-1,1 } ;            // change in column
   int y[]={0, 0,1,-1,-1,-1, 1,1 } ;            //change in row
   vector<int> v;
   for(int i=0;i<m;i++)
   {
      for(int j=0;j<n;j++)
      {
         if(grid[i][j]==word[0])
         {
         for(int dir=0;dir<8;dir++)
         {

            if(search(grid,word,1,i+y[dir],j+x[dir],m,n,y[dir],x[dir]))
             {
              v.push_back(i);
```

```
                v.push_back(j);
                res.push_back(v);
                break;
               }
            }
            }
            v.clear();
          }
       }
       return res;
```

FIND LARGEST WORD IN A DICTIONARY

```
bool issubsequence(string& s1, string& s2)
{
   int n = s1.length(), m = s2.length();
   int i = 0, j = 0;
   while (i < n && j < m) {
      if (s1[i] == s2[j])
         i++;
      j++;
   }
   return i == n;
}
   string findLongestWord(string S, vector<string> d) {
      // code here
      int longest=0;
      string ans="";
      for(int i=0;i<d.size();i++)
      {
         if(issubsequence(d[i],S))
         {
            if(d[i].length()>longest)
            {
               longest=d[i].length();
               ans=d[i];
            }
```

```
            else if(d[i].length()==longest)
            {
                int res = ans.compare(d[i]);
                if(res>0)
                {
                   ans=d[i];
                }

            }
            else
            {
               continue;
            }
         }
      }
      return ans;
   }
```

FLATTEN A LINKEDLIST

```
Node* merge(Node* first , Node* second)
{
  Node* temp = new Node(-1);
  Node* ans = temp;

  while(first!=NULL && second!=NULL)
  {
     if(first->data < second->data)
     {
        temp->bottom = first;
        temp = first;
        first = first->bottom;
     }

     else
     {
        temp->bottom = second;
        temp = second;
```

```
        second = second->bottom;
      }
    }

    if(first!=NULL)
    {
        temp->bottom = first;
          temp = first;
          first = first->bottom;
    }

    if(second!=NULL)
    {
        temp->bottom = second;
          temp = second;
          second = second->bottom;
    }

    ans = ans->bottom;
    return ans;
}
Node *flatten(Node *root)
{
    // Your code here
     if(root == NULL || root->next == NULL)
    {
       return root;
    }

    root->next = flatten(root->next);
    root = merge(root,root->next);
    return root;
}
```

GAME STRING

```
int minValue(string s, int k){
      // code here
```

```
    int sum=0;
    map<char,int>m;
    vector<int>v;
    for(int i=0;i<s.size();i++){
        m[s[i]]++;
    }
    for(auto it:m){
        v.push_back(it.second);
    }
    sort(v.begin(),v.end());
    while(k>0){
        v[v.size()-1]--;
        sort(v.begin(),v.end());
        k--;
    }
    for(int i=0;i<v.size();i++){
        sum+=v[i]*v[i];
    }
    return sum;
}
```

GREEDY FRACTIONAL KNAPSACK

```
double fractionalKnapsack(int W, Item arr[], int n)
{
    // Your code here
    vector<pair<double,int>>v;
    for(int i=0;i<n;i++)
    {
        double x= (arr[i].value*1.0)/(arr[i].weight*1.0);
        v.push_back({x,i});
    }
    sort(v.begin(),v.end(),greater<pair<double,int>>());
    int s=0;
    double ans=0;
    for(int i=0;i<v.size();i++)
    {
        if(s+arr[v[i].second].weight<W)
```

```
        {
            ans+=arr[v[i].second].value;
            s+=arr[v[i].second].weight;
        }
        else
        {
            double x=(W-s)*1.0;
            ans+=(x*v[i].first);
            break;
        }
    }
    return ans;
}
```

JOB SCQUENCING PROBLEM

```
static bool comp (Job a, Job b)
{
    return (a.profit>b.profit);
}
vector<int> JobScheduling(Job arr[], int n)
{
    // your code here
    sort(arr,arr+n,comp);
    vector<int>vis(n,0);
    int day=0,Profit=0;
    for(int i=0;i<n;i++)
    {
        for(int j=min(n,arr[i].dead-1);j>=0;j--)
        {
            if(vis[j]==0)
            {
                day+=1;
                Profit+=arr[i].profit;
                vis[j]=1;
                break;
            }
        }
```

```
        }
        return {day,Profit};
    }
```

MINIMIZE THE HEIGHT 2

```
int getMinDiff(int arr[], int n, int k) {
        // code here
        sort(arr,arr+n);
        int ans=arr[n-1]-arr[0];
        int smallest=arr[0]+k;
        int largest=arr[n-1]-k;
        int mn,mx;
        for(int i=0;i<n-1;i++)
        {
            mn=min(smallest,arr[i+1]-k);
            mx=max(largest,arr[i]+k);
            if(mn<0)
            {
                continue;
            }
            ans=min(ans,(mx-mn));
        }
        return ans;
    }
```

GREEDY MINIMUM PLATFORM

```
int findPlatform(int arr[], int dep[], int n)
    {
            sort(arr, arr+n);
            sort(dep, dep+n);
            int maxPlatform=1;
            int ans=maxPlatform;
            int i=1;
            int j=0;
            while(i<n && j<n){
                if(arr[i]<=dep[j]){
                    maxPlatform++;
```

```
            i++;
        }
        else{
            maxPlatform--;
            j++;
        }
        ans=max(ans,maxPlatform);
      }
      return ans;
  }
```

GREEDY POLICE AND THIEF

```
int catchThieves(char arr[], int n, int k)
  {
    // Code here
    queue<int> dq;
    vector<int> v;
    int count=0;
    for(int i=0;i<n;i++){
      if(arr[i]=='P')
        v.push_back(i);
      else dq.push(i);
    }
    for(int i:v){
      if(dq.size() and dq.front()<i){
        while(dq.size() and i-dq.front()>k)
          dq.pop();
        if(dq.size()){
          count++;
          dq.pop();
        }
      }
      else if(dq.size() and dq.front()-i<=k){
        dq.pop();
        count++;
      }
    }
```

```
    return count;
  }
```

GREEDY MAJORITY ELEMENT

```
int majorityElement(int a[], int size)
{

  // your code here
  int majority=a[0], count=1 ;
    for(int i=1 ; i<size ; i++)
    {
      if(a[i]!=majority)
      {
        count-- ;
        if(count==0) majority=a[i],count=1 ;
      }
      else count++ ;
    }

    count=0 ;
    for(int i=0 ; i<size ; i++)
      if(a[i]==majority) count++ ;

    return count>size/2 ? majority : -1 ;

}
```

GREEDY WATER THE PLANTS

```
vector<pair<int,int>>sprinklers;
    for(int i=0;i<n;i++)
    {
      if(gallery[i]!=-1)
      sprinklers.push_back({i-gallery[i],i+gallery[i]});
    }
    sort(sprinklers.begin(),sprinklers.end());
    int target =0;
    int ans=0,i=0;
```

```
    while(target<n)
    {
       if(i==sprinklers.size()||sprinklers[i].first>target)
       return -1;
       int max_range = sprinklers[i].second;
    while(i+1<sprinklers.size()&&sprinklers[i+1].first<=target)
    {
       i++;
       max_range=max(max_range,sprinklers[i].second);
    }
    if(max_range<target)
    return -1;
    ans++;
    target=max_range+1;
    i++;
    }
    return ans;
```

IMP ADVENTURE IN A MAZE

```
long long MOD = 1e9+7;
vector<int> FindWays(vector<vector<int>>&matrix){
   // Code here
    int n=matrix.size();
  long long int dp[n][n];
  long long int sum[n][n];

  for(int i=0;i<n;i++)
  {
     for(int j=0;j<n;j++)
     {
        dp[i][j]=0;
        sum[i][j]=0;
     }
  }

  dp[0][0]=1;
  sum[0][0]=matrix[0][0];
```

```
int i=1;
while(i<n && (matrix[0][i-1]==1 || matrix[0][i-1]==3))
{
   dp[0][i]=1;
   sum[0][i]=sum[0][i-1]+matrix[0][i]+MOD;
   sum[0][i]=(sum[0][i])%MOD;
   i++;
}

i=1;
while(i<n && (matrix[i-1][0]==2 || matrix[i-1][0]==3))
{
   dp[i][0]=1;
   sum[i][0]=sum[i-1][0]+matrix[i][0]+MOD;
   sum[i][0]=(sum[i][0])%MOD;
   i++;
}

for(int i=1;i<n;i++)
{
   for(int j=1;j<n;j++)
   {
      int tempsum1=0;
      int tempsum2=0;

      if(dp[i][j-1] && (matrix[i][j-1]==1 || matrix[i][j-1]==3))
      {
         dp[i][j]+=dp[i][j-1]+MOD;
         dp[i][j]=(dp[i][j])%MOD;
         tempsum1=matrix[i][j-1];
      }

      if(dp[i-1][j] && (matrix[i-1][j]==2 || matrix[i-1][j]==3))
      {
         dp[i][j]+=dp[i-1][j]+MOD;
         dp[i][j]=(dp[i][j])%MOD;
```

```
            tempsum2=matrix[i-1][j];
        }

        if(tempsum1 && tempsum2)
        {
            sum[i][j]=max(sum[i][j-1],sum[i-1][j])+matrix[i][j]+MOD;
            sum[i][j]=sum[i][j]%MOD;
        }
        else if(tempsum1)
        {
            sum[i][j]=sum[i][j-1]+matrix[i][j]+MOD;
            sum[i][j]=sum[i][j]%MOD;
        }
        else if(tempsum2)
        {
            sum[i][j]=sum[i-1][j]+matrix[i][j]+MOD;
            sum[i][j]=sum[i][j]%MOD;
        }
      }
    }
    return {dp[n-1][n-1],sum[n-1][n-1]};
```

BINARY SEARCH IN FOREST

```
int solve(int a,int tree[],int n){
      int p=0;
      for(int i=0;i<n;i++){
          if(tree[i]>a){
              p+=tree[i]-a;
          }
      }
      return p;

    }
     int find_height(int tree[], int n, int k)
     {
        // code here
```

```
    if(n==1) return tree[0]-k;
    int m=INT_MAX;
    int ma=INT_MIN;
    for(int i=0;i<n;i++){
        if(tree[i]>ma){
            ma=tree[i];
        }
        if(tree[i]<m){
            m=tree[i];
        }
    }
    int l=m,h=ma;
    while(l<=h){
        int m=(l+h)/2;
        int p=solve(m,tree,n);
        if(p==k){
            return m;
        }
        else if(p>k){
            l=m+1;
        }
        else{
            h=m-1;
        }
    }
    return -1;
}
```

IMP CATALAN NO

```
cpp_int findCatalan(int n)
{
    cpp_int ans = 1;
    for(cpp_int i=1; i<=n; i++){
        ans = (ans*(4*i - 2))/(i+1);
    }
    return ans;
}
```

IMP MAXIMIZE NO OF 1'S

```
int findZeroes(int arr[], int n, int m) {
        // code here
        int left = 0 , right = 0 ;
        int zeros = 0, ans =0;

        while(right < n){
            if(zeros <= m) {
                if(arr[right] == 0) zeros++ ;
                right++;
            }
            if(zeros > m){
                if(arr[left] == 0) zeros-- ;
                left++ ;
            }
            ans = max(ans , right-left);
        }
        return ans ;
    }
```

IMP DUPLICATE SUBTREE IN A BINARY TREE

```
unordered_map<string,int>mp;
    string solve(Node* root)
    {
        if(!root)
        return "$";
        string s="";
        if(!root->right && !root->left)
        {
            s=to_string(root->data);
            return s;
        }
        s=s+to_string(root->data);
        s=s+solve(root->left);
        s=s+solve(root->right);
        mp[s]++;
```

```
        return s;
    }
    int dupSub(Node *root) {
        // code here
        mp.clear();
        solve(root);
        for(auto it:mp)
        if(it.second>=2)
        return 1;
        return 0;
    }
```

IMP MAXIMUM SUM IN THE CONFIGURATION

```
int max_sum(int A[],int N)
{
//Your code here
int tsum = 0,sum=0;
    for(int i=0;i<N;i++){
        tsum += A[i]*i;
        sum += A[i];
    }
    int ans = tsum;
    int len = N;
    while(N){
        tsum = tsum + sum - len*A[N-1];
        ans = max(ans,tsum);
        N--;
    }
    return ans;
```

IMP RETURN THE NO WHICH OCCURE ONLY ONC WHILE OTHER OCCURE THRICE

```
int singleElement(int arr[] ,int n) {
        int ones = 0, twos = 0;

        int common_bit_mask;

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

```
        twos = twos | (ones & arr[i]);

        ones = ones ^ arr[i];

        common_bit_mask = ~(ones & twos);

        ones &= common_bit_mask;

        twos &= common_bit_mask;


    }
    return ones;
    }
```

IMP MAXIMUM DIFF BETWEEN NODE AND ANCESTOR

```
int solve(Node* root,int &ans)
{
    if(!root)
    return INT_MAX;
    int left=solve(root->left,ans);
    int right=solve(root->right,ans);
    ans=max(ans,max(root->data-left,root->data-right));
    return min(left,min(root->data,right));
}
int maxDiff(Node* root)
{
    // Your code here
    int ans=INT_MIN;
    solve(root,ans);
    return ans;
}
```

INTERSECTION POINT OF TWO LINKEDLIST

```
Node* findIntersection(Node* head1, Node* head2)
```

```
{
    Node *head = new Node(0), *ptr = head;
    while(head1 && head2){
        if(head1->data < head2->data){
            head1 = head1->next;
        }
        else if(head1->data > head2->data){
            head2 = head2->next;
        }
        else{
            Node *tmp = new Node(head1->data);
            ptr->next = tmp;
            ptr = tmp;
            head1 = head1->next;
            head2 = head2->next;
        }
    }
    return head->next;
}
```

LARGEST NO FORMED FROM AN ARRAY

```
static bool cmp(string& a,string& b){
  return a+b > b+a;
}


            string printLargest(vector<string> &arr) {
                vector<string>s;
  int n=arr.size();
  for(int i=0;i<n;i++){
     string a=arr[i] ;
     s.push_back(a);
  }
  sort(s.begin(),s.end(),cmp);
  string ans="";
  for( auto i : s){
     ans+=i;
  }
```

```
  return ans;
        }
```

K SUM PATH

```
void solve(Node* root, vector<int> arr, int k, int &count){
    if(root == NULL){
       return;
    }

    arr.push_back(root->data);
    int sum = 0;



    solve(root->left, arr, k, count);
    solve(root->right, arr, k, count);

    int n = arr.size();
    for(int i=n-1; i>=0; i--){
       sum += arr[i];
       if(sum == k){
          count++;
       }
    }

    arr.pop_back();
  }

  int sumK(Node *root,int k)
  {
     // code here
     vector<int> arr;
    int count = 0;
    solve(root, arr, k, count);
    return count;
  }
```

JUMP GAME

```
int canReach(int A[], int N) {
      // code here
       int farthest=0;
   for(int i=0;i<N;i++){
      if(farthest<i) return false;
      farthest=max(farthest,i+A[i]);
   }
   return true;


   }
```

Kth SMALLEST ELEMENT IN A BST

```
void help(Node *head, int &k,int &r)
  {
     if(head==NULL)
        return;
     help(head->left,k,r);
     --k;
     if(k==0)
     {
        r=head->data;
        return;
     }
     help(head->right,k,r);
  }


  int KthSmallestElement(Node *root, int K)
  {
     int r=-1;
     help(root,K,r);
     return r;
  }
```

LINKEDLIST POLYNOMIAL ADDITION

```
Node* addPolynomial(Node *p1, Node *p2)
   {
      //Your code here
```

```
    Node *sum = new Node(0, 0);
  Node *sumRef = sum;
  while((p1 != NULL) && (p2 != NULL))
  {
    if(p1->pow == p2->pow)
    {
      sum->next = new Node(p1->coeff + p2->coeff, p1->pow);
      p1 = p1->next;
      p2 = p2->next;
    }
    else if(p1->pow > p2->pow)
    {
      sum->next = new Node(p1->coeff, p1->pow);
      p1 = p1->next;
    }
    else{
      sum->next = new Node(p2->coeff, p2->pow);
      p2 = p2->next;
    }
    sum = sum->next;
  }

  if(p1)
    sum->next = p1;
  else if(p2)
    sum->next = p2;
  return sumRef->next;
  }
```

LINKEDLIST QUICK SORT

```
node *partition(node *root,node *end)
{
  node *pivot=root, *curr=root->next;
  while(curr!=end)
  {
    if(curr->data<=pivot->data)
    {
```

```
        swap(curr->data,pivot->data);
      }
      curr=curr->next;
   }
   return pivot;

}
void quick_sort(node *root , node* end=NULL)
{
   if(!root || !root->next || root==end)
   {
      return;
   }
   node* mid=partition(root,end);
   quick_sort(root,mid);
   quick_sort(mid->next,end);
}
void quickSort(struct node **headRef)
{
   quick_sort(*headRef);
}
```

MAXIMUM DIFF BETWEEN PAIR IN A MATRIX

```
int findMaxValue(vector<vector<int>>&mat, int N)
   {
      int maxValue = INT_MIN;
    int maxArr[N][N];
    maxArr[N-1][N-1] = mat[N-1][N-1];

    int maxv = mat[N-1][N-1];
    for (int j = N - 2; j >= 0; j--)
    {
       if (mat[N-1][j] > maxv)
          maxv = mat[N - 1][j];
       maxArr[N-1][j] = maxv;
    }
```

```
    maxv = mat[N - 1][N - 1];
    for (int i = N - 2; i >= 0; i--)
    {
        if (mat[i][N - 1] > maxv)
            maxv = mat[i][N - 1];
        maxArr[i][N - 1] = maxv;
    }

    for (int i = N-2; i >= 0; i--)
    {
        for (int j = N-2; j >= 0; j--)
        {

            if (maxArr[i+1][j+1] - mat[i][j] >
                                    maxValue)
                maxValue = maxArr[i + 1][j + 1] - mat[i][j];

            maxArr[i][j] = max(mat[i][j],
                        max(maxArr[i][j + 1],
                            maxArr[i + 1][j]) );
        }
    }

    return maxValue;
    }
```

LUCKY NUMBER

```
 bool isLucky(int n) {
       for(int i=2;i<=n;i++){
      if(n%i==0) return false;
      n = n - n/i;
       }
    }
```

PRINT LONGEST PALINDROME IN A STRING

```
string longestPalin (string S) {
       // code here
```

```
int len=0;
int start=0;
int l;
int r;
//even case
for(int i=0; i<S.length()-1; i++)
{
   l=i;
   r=i+1;
   while(l>=0 && r<S.length() && S[l]==S[r])
   {
      if(r-l+1 > len){
         len=r-l+1;
         start=l;
      }
      l--;
      r++;
   }
}

//odd case
for(int i=1; i<S.length()-1; i++)
{
   l=i-1;
   r=i+1;
   while(l>=0 && r<S.length() && S[l]==S[r])
   {
      if(r-l+1 > len){
         len=r-l+1;
         start=l;
      }
      l--;
      r++;
   }
}
if(len==0){
   len++;
```

```
        }
        return S.substr(start, len);
    }
```

MCM RECURSIVE SOLUTION

```
int mcm(int arr[], int l, int r)
    {
        if(l==r)
        return 0;
        int res=INT_MAX;
        int temp;
        for(int i=l;i<r;i++)
        {
            temp=mcm(arr,l,i)+mcm(arr,i+1,r)+(arr[l-1]*arr[i]*arr[r]);
            res=min(res,temp);
        }
        return res;
    }
    int matrixMultiplication(int N, int arr[])
    {
        // code here
        return mcm(arr, 1, N-1);
    }
```

MCM DP SOLUTION

```
int matrixMultiplication(int N, int arr[])
    {
        vector<vector<int>>v(N,vector<int>(N,-1));
       for(int i=1;i<N;i++)
       v[i][i]=0;
       for(int i=N-1;i>=1;i--)
       {
          for(int j=i+1;j<=N-1;j++)
          {
             int m=1e9;
             for(int k=i;k<j;k++)
             {
```

```
                int x=v[i][k]+v[k+1][j]+arr[i-1]*arr[k]*arr[j];
                m=min(m,x);
            }
            v[i][j]=m;
          }
      }
      return v[1][N-1];
   }
```

MATRIX ROTATE BY 90 DEGREE

```
void rotate(vector<vector<int> >& matrix)
{
   // Your code goes here
   int n=matrix.size();
  for(int i=0;i<n;i++)
  {
     for(int j=0;j<n/2;j++)
     {
        swap(matrix[i][j],matrix[i][n-j-1]);
     }
  }

  for(int i=0;i<n;i++)
  {
     for(int j=i+1;j<n;j++)
     {
        swap(matrix[i][j],matrix[j][i]);
     }
  }
}
int maxAreaHist(vector<int> arr, int n) {
      stack<int> s;

      long long maxArea = 0;

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

```
        while(!s.empty() and(i == n or arr[s.top()] >= arr[i])) {
            long long height = arr[s.top()];
            s.pop();
            long long width;
            if(s.empty() == true) {
                width = i;
            } else {
                width =  i - s.top() - 1;
            }

            maxArea = max(maxArea, height*width);
        }
        s.push(i);
    }

    return maxArea;
}
int maxArea(int M[MAX][MAX], int n, int m) {
    // Your code here
    vector<int> height(m);

    for(int j = 0; j < m; j++) {
        height[j] = (M[0][j]);
    }

    int area = maxAreaHist(height, m);




    for(int i = 1; i < n; i++) {
        int local = 0;
        for(int j = 0; j < m; j++) {
            if(M[i][j] == 0) {
                height[j] = 0;
            } else {
                height[j] +=  1;
            }
```

```
        }


        local = maxAreaHist(height, m);

        area = max(area, local);
      }

     return area;

  }
```

MERGE SORT

```
void merge(int arr[], int l, int m, int r)
  {
    int i=l; int j=m+1;
    int k=0;
    int arr1[r-l+2];
    while(i<=m and j<=r)
    {
       if(arr[i]<arr[j])
       arr1[k++]=arr[i++];
       else arr1[k++]=arr[j++];
    }
    while(i<=m)
    {
       arr1[k++]=arr[i++];
    }
    while(j<=r)
    {
       arr1[k++]=arr[j++];
    }
    k=0;
    for(int i=l;i<=r;i++)
     arr[i]=arr1[k++];
  }
  public:
```

```
  void mergeSort(int arr[], int l, int r)
  {
      //code here
      if(l>=r) return ;

    int mid=l+(r-l)/2;

    mergeSort(arr,l,mid);

    mergeSort(arr,mid+1,r);

    merge(arr,l,mid,r);
  }
```

MERGE SORT LINKED LIST

```
Node *mergekro(Node* l1, Node* l2){
    if(l1==NULL){
        return l2;
    }
    if(l2==NULL){
        return l1;
    }
    if(l1->data > l2->data){
        l2->next = mergekro(l1,l2->next);
        return l2;
    }else{
        l1->next = mergekro(l1->next,l2);
        return l1;
    }
 }
  Node* mergeSort(Node* head) {
      // your code here
      if(head==NULL || head->next==NULL) return head;
      Node *temp = NULL;
      Node *slow = head;
      Node *fast = head;
```

```
    while(fast && fast -> next)
    {
        temp = slow;
        slow = slow->next;
        fast = fast ->next ->next;

    }
    temp -> next = NULL;
    Node *l1 = mergeSort(head);
    Node *l2 = mergeSort(slow);
    return mergekro(l1,l2);
  }
bool search(vector<vector<int> > matrix, int n, int m, int x)
  {
    // code here
    int row = 0;
    int col = matrix[0].size()-1;
    while(row<matrix.size() && col>=0){
    if(matrix[row][col]==x){
        return true;
    }
    else if(x >matrix[row][col]){
        row++;
    }
    else{
        col--;
    }
  }
  return false;
  }
```

MAXIMUM AREA OF A HISTOGRAM

```
long long getMaxArea(long long arr[], int n)
  {
    // Your code here
   vector<long long> left(n,-1);
    vector<long long> right(n,n);
```

```
    stack<long long> st;

    for(int i=0 ; i<n; i++)
    {
       while(!st.empty() && arr[st.top()] >= arr[i])
          st.pop();
       if(!st.empty())
          left[i] = st.top();
       st.push(i);
    }

    while(!st.empty())
       st.pop();

    for(int i=n-1 ; i>=0 ; i--)
    {
       while(!st.empty() && arr[st.top()] >= arr[i])
          st.pop();

       if(!st.empty())
          right[i] = st.top();
       st.push(i);
    }

    long long ans = 0;
    for(int i=0 ; i<n ; i++)
       ans = max(ans, arr[i]*(right[i]-left[i]-1));
    return ans;
  }
```

MINIMUM CHARACTER ADDED IN FRONT OF A STRING TO MAKE IT PALINDROME

```
int minChar(string s){
    //Write your code here
    int n=s.size();
      string str=s;
     s.push_back('#');
      reverse(str.begin(),str.end());
```

```
        s=s+str;


         vector<int>lps(s.size());
         lps[0]=0;
         int i=1,j=0;
        while(i<s.size()){
           if(s[i]==s[j]){
              lps[i]=j+1;
              i++;j++;
           }else{
              if(j!=0){
                 j=lps[j-1];
              }else{
                 lps[i]=0;
                 i++;
              }
           }
        }


   return n-lps.back();
```

MINIMUM COST OF ROPES

```
long long minCost(long long arr[], long long n) {
       // Your code here
      priority_queue<long long,vector<long long>,greater<long long>> pq;
      for(int i=0;i<n;i++)
      pq.push(arr[i]);

      long long ans =0;
      long long size = n;
      while(size>1)
      {
         long long a = pq.top();
         pq.pop();
         long long b = pq.top();
         pq.pop();
         long long aa = a+b;
```

```
        pq.push(aa);
        ans =ans + aa;
        size--;
      }

      return ans;
    }
```

MULTIPLY TWO STRINGS

```
string multiplyStrings(string num1, string num2) {
      //Your code here
      if (num1 == "0" || num2 == "0") return "0";
      string res = "";
      if(num1[0]=='-' && num2[0]=='-')
      {
        num1=num1.substr(1);
        num2=num2.substr(1);
      }
      else if(num1[0]=='-' && num2[0]!='-')
      {
        num1=num1.substr(1);
        res.push_back('-');
      }
      else if(num1[0]!='-' && num2[0]=='-')
      {
        num2=num2.substr(1);
        res.push_back('-');
      }
       vector<int> num(num1.size() + num2.size(), 0);
       for (int i = num1.size() - 1; i >= 0; --i) {
          for (int j = num2.size() - 1; j >= 0; --j) {
            num[i + j + 1] += (num1[i] - '0') * (num2[j] - '0');
            num[i + j] += num[i + j + 1] / 10;
            num[i + j + 1] %= 10;
          }
       }
       int i = 0;
```

```
    while (i < num.size() && num[i] == 0) ++i;
    while (i < num.size()) res.push_back(num[i++] + '0');
    return res;
  }
```

RETURN ALL SUBSETS

```
void fun(vector<int>temp,vector<vector<int>>&ans,vector<int>A,int i)
  {
    if(i>=A.size())
    {
      ans.push_back(temp);
      return;
    }
    fun(temp,ans,A,i+1);
    temp.push_back(A[i]);
    fun(temp,ans,A,i+1);

  }
  vector<vector<int> > subsets(vector<int>& A)
  {
    //code here
    vector<vector<int>>ans;
    vector<int>temp;
    fun(temp,ans,A,0);
    sort(ans.begin(),ans.end());
    return ans;
  }
```

RECURSSION POSSIBLE NO FROM PHONE DIGITS

```
vector<string> s{"","","abc","def","ghi","jkl","mno","pqrs","tuv","wxyz"};
  vector<string>ans;
  void solve(int ip[],string op,int n,int idx)
  {
    if(idx==n)
    {
```

```
            ans.push_back(op);
            return;
        }
        int x=ip[idx];
        for(int i=0;i<s[x].size();i++)
        {
            solve(ip,op+s[x][i],n,idx+1);
        }


    }
    vector<string> possibleWords(int a[], int N)
    {
        //Your code here
        string op;
        solve(a,op,N,0);
        return ans;
    }
```

ROMAN NO TO INTEGER

```
int romanToDecimal(string &s) {
        // code here
        int sr, n=s.length(),sum=0,temp=INT_MAX;
       for(int i=0; i<s.length(); i++)
       {
           if(i==n-2 && s[i]=='I' && s[i+1]=='V')
           {
               sr=4;
               sum+=sr;
               break;
           }
           else if(i==n-2 && s[i]=='I' && s[i+1]=='X')
           {
               sr=9;
               sum+=sr;
               break;
           }
```

```
        else if(s[i]=='I')  sr=1;
        else if(s[i]=='V') sr=5;
        else if(s[i]=='X') sr=10;
        else if(s[i]=='L') sr=50;
        else if(s[i]=='C') sr=100;
        else if(s[i]=='D') sr=500;
        else if(s[i]=='M') sr=1000;
        sum+=sr;
        if(temp<sr)
        sum -= temp*2;
        temp = sr;
    }
    return sum;
  }
```

SLIDING WINDOW MIN SWAPS TO MAKE ALL ELEMENTS LESS THAN K TOGETHER

```
int minSwap(int arr[], int n, int k) {
     // Complet the function
    int count = 0;
    for(int i=0; i<n; i++)
        if(arr[i] <= k)count++;

        if(count == 0) return 0;
     int i=0, j=0;
     int bad = 0;
     int ans = INT_MAX;

     while(j < n){
         if(arr[j] > k) bad++;
         if(j-i+1 < count) j++;
         else if(j-i+1 == count){
             ans = min(ans, bad);

             if(arr[i] > k) bad--;
             i++,j++;
         }
     }
```

```
        return ans;
    }
string removeKdigits(string S, int k)
    {
        int i = 0;
        stack <char> s;
        while(i<S.length()){
            while(!s.empty() && s.top()>S[i] && k){
                s.pop();
                k--;
            }
            if(!s.empty() || S[i] != '0')
                s.push(S[i]);
            i++;
        }
        while(!s.empty() && k--)
            s.pop();
        string str = "";
        while(!s.empty()){
            str += s.top();
            s.pop();
        }
        reverse(str.begin(), str.end());
        if(str.empty())
            str += "0";
        return str;
    }
```

SLIDING WINDOW LONGEST K UNIQUE CHARACTER SUBSTRING

```
int longestKSubstr(string s, int k) {
    // your code here
    unordered_map<char,int>mp;
        int i=0,j=0;
        int ans =-1;
        while(j<s.length()){
            mp[s[j]]++;
            if(mp.size() == k){
```

```
            ans = max(ans,j-i+1);
        }
        else if(mp.size() > k){
            while(mp.size() > k){
                mp[s[i]]--;
                if(mp[s[i]] == 0) mp.erase(s[i]);
                i++;
            }
        }
        j++;
    }
    return ans;
}
```

STACK NEXT GREATER ELEMENT

```
vector<long long> nextLargerElement(vector<long long>arr,int n)
  {
     vector<long long>ans;
    stack<long long>st;
    st.push(arr[n-1]);
    ans.push_back(-1);
    for(int i=n-2;i>=0;i--)
    {
       while(!st.empty() &&  st.top()<=arr[i])
         st.pop();
       long long x=st.empty()?-1:st.top();
       ans.push_back(x);
       st.push(arr[i]);
    }
    reverse(ans.begin(),ans.end());
    return ans;
  }
```

RESTRICTIVE CANDY CRUSH

```
string Reduced_String(int k,string s){
    // Your code goes here
    if(k==1)
```

```
    return "";
    stack<pair<char,int>> st;
    for(int i=0;i<s.size();i++)
    {
      if(st.empty() || st.top().first != s[i])
      st.push({s[i],1});
      else if(st.top().first == s[i] && st.top().second == k-1)
      st.pop();
      else{
        st.top().second++;
      }

    }
    string ans;
    while(!st.empty())
    {
      int n = st.top().second;
      char c = st.top().first;
      while(n--)
      {
        ans+=c;
      }
      st.pop();
    }
    reverse(ans.begin(),ans.end());
    return ans;
  }
```

STRING FORMATION FROM A SUBSTRING

```
int isRepeat(string s)
        {
          // Your code goes here
          int i=1, j=0;
 int n= s.length();
 while(i<s.length())
```

```
 {
    if(s[j]==s[i]){
       i++;
       j++;
    }
    else{
       i=i-j+1;
       j=0;
    }
 }
if(j==0)return 0;
return n%(n-j)==0;
          }
vector<vector<string> > Anagrams(vector<string>& str) {
      //code here
      vector<vector<string>>ans;
     unordered_map<string, vector<string>>m;
     for(int i=0; i<str.size(); i++) {
        string s = str[i];
        sort(str[i].begin(), str[i].end());
        m[str[i]].push_back(s);
     }
     for(auto &i : m) {
        ans.push_back(i.second);
     }
     return ans;
  }
```

SUBSET SUM BETWEEN THE GIVEN RANGE

```
long long int ans = 0;
  void helper(vector<int>&arr, int n, int l, int r,int count,int index){
      if(index == n){
         if(count >= l && count <= r)ans++;
         return;
      }
      helper(arr,n,l,r,count+arr[index],index+1);
      helper(arr,n,l,r,count,index+1);
```

```
}
long long int countSubsets(vector<int>&arr, int n, int l, int r){
    //Write your code here
    helper(arr,n,l,r,0,0);
    return ans;
}
```

TREE ANCESTORS IN A BINARY TREE

```
void solve(Node* root,vector<int>&res,int &check,int target){
    if(!root)
    return;
    if(root!=NULL){
        if(root->data==target){
            check=1;
            return;
        }
        solve(root->left,res,check,target);
        if(check){
            res.push_back(root->data);
            return;
        }
        else solve(root->right,res,check,target);
        if(check)
            res.push_back(root->data);
    }
}
vector<int> Ancestors(struct Node *root, int target)
{
    // Code here
    vector<int>res;
    int check=0;
    if(root!=NULL && root->data==target)
        return {};
    solve(root,res,check,target);
    return res;
}
```

TREE CHECK IF A TREE IS SUBTREE OF T OR NOT

```
bool solve(Node* n1, Node* n2)
  {
     if(!n1 && !n2)
     return true;
     if(!n1 || !n2)
     return false;
     if(n1->data!=n2->data)
     return false;
     return (solve(n1->left,n2->left)&&solve(n1->right,n2->right));
  }
  bool isSubTree(Node* T, Node* S)
  {
     // Your code here
     if(!T)
     return false;
     if(solve(T,S))
     return true;
     return (isSubTree(T->left,S) || isSubTree(T->right,S));

  }
```

TREE CHECK IF TWO NODES ARE COUSINS OR NOT

```
bool isCousins(Node *root, int a, int b)
{
  //add code here.
  int f1=0,f2=0;
 queue<Node*>q;
 q.push(root);
 while(!q.empty()){
    int n=q.size();
    for(int i=1;i<=n;i++){
       Node*temp=q.front();
       q.pop();
       if(temp->left and temp->right){
          if(temp->left->data==a and temp->right->data==b)return false;
```

```
            if(temp->left->data==b and temp->right->data==a)return false;
        }
        if(temp->data==a)f1=1;
        if(temp->data==b)f2=1;
        if(temp->left)q.push(temp->left);
        if(temp->right)q.push(temp->right);
    }
    if(f1 and f2){
        return true;
    }
    if(f1 or f2){
        return false;
    }
 }
 return false;

}
```

TREE CHECK IF TWO TREE ARE MIRROR OR NOT

```
int areMirror(Node* a, Node* b) {
   // Your code here
      if(a==NULL  &&  b==NULL)
      return 1;
      if(a!=NULL && b!=NULL && a->data==b->data)
      {
         return areMirror(a->left,b->right) && areMirror(a->right,b->left);
      }
      else
      return 0;
   }
```

TREE CONNECT NODE AT SAME LEVEL

```
void connect(Node *root)
   {
      // Your Code Here
      queue<Node*> q;
```

```
    q.push(root);
    while(!q.empty()){
        int size = q.size();
        for(int i=0;i<size;i++){
            Node* temp = q.front();
            q.pop();
            if(i<size-1){
                temp->nextRight = q.front();

            }
            if(temp->left){
                q.push(temp->left);
            }
            if(temp->right){
                q.push(temp->right);
            }
        }
    }
  }
```

TREE DIAGONAL TRAVERSAL SUM

```
void solve(Node* root,map<int,int>&m,int level){
  if(root==NULL) return ;
  m[level]+=root->data;
  solve(root->left,m,level-1);
  solve(root->right,m,level);
}
vector <int> diagonalSum(Node* root) {
   // Add your code here
   map<int,int> m;
  vector<int> v;
  solve(root,m,0);
  for(auto it:m){
     v.push_back(it.second);
  }
  reverse(v.begin(),v.end());
  return v;
```

```
}
```

TREE DIAGONAL TRAVERSAL

```
vector<int> diagonal(Node *root)
{
  // your code here
  vector<int>ans;
  if(!root)
  return ans;
  queue<Node*>q;
  q.push(root);
  while(!q.empty())
  {
     Node* temp=q.front();
     q.pop();
     while(temp)
     {
        if(temp->left) q.push(temp->left);
        ans.push_back(temp->data);
        temp=temp->right;
     }
  }
  return ans;
}
```

TREE EXPRESSION TREE

```
 int evalTree(node* root) {
       // Your code here
       if(root==NULL)return 0;
      if(root->left==NULL and root->right==NULL)return stoi(root->data);
      int lh=evalTree(root->left);
      int rh=evalTree(root->right);
      if(root->data=="+")return lh+rh;
      if(root->data=="-")return lh-rh;
      if(root->data=="*")return lh*rh;
      if(root->data=="/")return lh/rh;
    }
```

TREE ISOMORPHIC

```
bool isIsomorphic(Node *root1,Node *root2)
   {
    //add code here.
   if(root1==NULL and root2==NULL)return true;
   if(root1==NULL or root2==NULL)return false;
   if(root1->data!=root2->data){
      return false;
   }
   bool a=isIsomorphic(root1->left,root2->left) and isIsomorphic(root1->right,root2->right);
   bool b=isIsomorphic(root1->left,root2->right) and isIsomorphic(root1->right,root2->left);
   return a or b;


    }
```

TREE MIN DIS BETWEEN TWO NODES

```
int distance(Node* root,int x,int level)
   {
      if(!root)
      return -1;
      if(root->data==x)
      return level;
      int left=distance(root->left,x,level+1);
      if(left==-1)
      {
         int right=distance(root->right,x,level+1);
         return right;
      }
      return left;
   }
   Node* lca(Node* root ,int n1 ,int n2 )
   {
     //Your code here
     if(!root)
     return NULL;
     else
```

```
    {
        Node* left=lca(root->left,n1,n2);
        Node* right=lca(root->right,n1,n2);
        if(root->data==n1 || root->data==n2)
        {
            return root;
        }
        if(left==NULL)
        return right;
        else if(right==NULL)
        return left;
        else
        return root;
    }
  }
  int findDist(Node* root, int a, int b) {
      // Your code here
      int ca=lca(root,a,b)->data;
      int dis=0,dis1=0,dis2=0;
      dis=distance(root,ca,0);
      dis1=distance(root,a,0);
      dis2=distance(root,b,0);
      return (dis1+dis2-(2*dis));
  }
```

TREE PATH FROM ROOTH WITH A SPECIFIED SUM

```
void solve(Node* root, int sum, vector<vector<int>>&ans, vector<int>temp)
  {
      if(!root)
      {
          return;
      }
      if((root->key-sum)==0)
      {
          temp.push_back(root->key);
          ans.push_back(temp);
      }
```

```
        else if(root->key<sum)
        {
            temp.push_back(root->key);
            solve(root->left,sum-root->key,ans,temp);
            solve(root->right,sum-root->key,ans,temp);
        }
        else
        {
            return;
        }
    }
    vector<vector<int>> printPaths(Node *root, int sum)
    {
        //code here
        vector<vector<int>>ans;
        vector<int>temp;
        solve(root,sum,ans,temp);
        return ans;
    }
```

TREE PRINT ALL NODES WHICH DON'T HAVE SIBLINGS

```
void solve(Node*root,vector<int>&v){
   if(root==NULL){
       return;
   }
   if(root->left and root->right==NULL){
       v.push_back(root->left->data);
   }
   if(root->left==NULL and root->right){
       v.push_back(root->right->data);
   }
   solve(root->left,v);
   solve(root->right,v);
}
vector<int> noSibling(Node* node)
{
    // code here
```

```
    vector<int>v;
   solve(node,v);
   if(v.size()==0){
      v.push_back(-1);
      return v;
   }
   sort(v.begin(),v.end());
   return v;
}
```

TREE REVERSE NODE OF ALTERNATE LEVEL

```
void helper(Node *&r1 , Node *&r2 , int level){

   if(!r1 || !r2) return;
   if(!(1&level)){
      int t=r1->data;
      r1->data=r2->data , r2->data=t;
   }
   helper(r1->left , r2->right , level+1);
   helper(r1->right , r2->left , level+1);
}
void reverseAlternate(Node *root)
{
    //Your code here
     helper(root->left , root->right , 2);
}
```

TREE ROOT TO LEAF PATH SUM

```
long long pathsum(Node *root,int i)
{
  if(!root) return 0;
  i=(i*10)+root->data;
  if(!(root->left) and !(root->right))return i;
  return pathsum(root->left,i)+pathsum(root->right,i);
}
long long treePathsSum(Node *root)
{
```

```
    //Your code here
    return pathsum(root,0);
}
```

MAXIMUM IN EACH LEVEL

```
vector<int> largestValues(Node* root)
    {
        //code here
        if(root==NULL)
        return {};
        vector<int>ans;
        queue<Node*>q;
        q.push(root);
        q.push(NULL);
        int temp_max=INT_MIN;
        while(!q.empty())
        {
            if(q.front()!=NULL)
            {
                temp_max=max(temp_max,q.front()->data);
                if(q.front()->left!=NULL)
                q.push(q.front()->left);
                if(q.front()->right!=NULL)
                q.push(q.front()->right);
                q.pop();
            }
            else if(q.front()==NULL && q.size()>1)
            {
                ans.push_back(temp_max);
                temp_max=INT_MIN;
                q.pop();
                q.push(NULL);
            }
            else
            {
                ans.push_back(temp_max);
                q.pop();
```

```
        }
      }
      return ans;
    }
```

TREE VERTICAL ORDER TRAVERSAL

```
vector<int> verticalOrder(Node *root)
  {
     //Your code here
     vector<int>ans;
    int index = 0;
    queue<pair<Node*, int>>q;
    map<int, vector<int>>m;
    q.push(make_pair(root, index));
    while(q.size() > 0) {
       int count = q.size();
       for(int i=0; i<count; i++) {
          Node *temp = q.front().first;
          index = q.front().second;
          q.pop();
          m[index].push_back(temp->data);
          if(temp->left != NULL) {
             q.push(make_pair(temp->left, index-1));
          }
          if(temp->right != NULL) {
             q.push(make_pair(temp->right, index+1));
          }
       }
    }
    for(auto &i : m) {
       for(auto &j : i.second)
          ans.push_back(j);
    }
    return ans;
  }
```

TRIE COUNT DISTINCT SUBSTRING

```
struct TrieNode
{
   TrieNode* children[26];
   bool isend;
   TrieNode()
   {
      for(int i=0;i<26;i++)
       children[i]=NULL;
       isend=false;
   }
};
int insert(TrieNode* root,string s,int j)
{
   TrieNode* curr=root;
   int ans=0;
   for(int i=j;i<s.size();i++)
   {
      int index=s[i]-'a';
      if(curr->children[index]==NULL)
      {
         ans++;
         TrieNode* temp=new TrieNode();
         curr->children[index]=temp;
      }
      curr=curr->children[index];
   }
   curr->isend=true;
   return ans;
}
int countDistinctSubstring(string s)
{
   //Your code here
   int ans=0;
   TrieNode* root=new TrieNode();
   for(int i=0;i<s.size();i++)
   {
      ans+=insert(root,s,i);
```

```
    }
    return ans+1;
}
```

TRIE INSERT AND SEARCH

```
void insert(struct TrieNode *root, string key)
{
    // code here
     if(key.length()==0)
   {
      root->isLeaf = true;
      return;
   }

   int index = key[0]-'a';
   TrieNode *child;

   if(root->children[index]!=NULL)
   {
      child = root->children[index];
   }
   else
   {
      child = new TrieNode();
      root->children[index] = child;
   }

   insert(child,key.substr(1));
}

//Function to use TRIE data structure and search the given string.
bool search(struct TrieNode *root, string key)
{
    // code here
    if(key.length()==0)
   {
      return root->isLeaf;
```

```
    }

    int index = key[0]-'a';
    TrieNode *child;

    if(root->children[index]!=NULL)
    {
        child = root->children[index];
    }
    else
    {
        return false;
    }

    return search(child,key.substr(1));
}
```