1. The length of the longest valid (well-formed) parentheses substring

```
// stack<int> st;

 //     st.push(-1);
 //     int maxLen=0;
 //     int n = s.size();
 //     for(int i=0;i<n;i++){
 //         if(s[i]==')'){
 //             st.pop();
 //             if(st.empty()){
 //                 st.push(i);
 //             }
 //             else{
 //                 int len = i - st.top();
 //                 cout << len << " ";
 //                 maxLen=max(len,maxLen);
 //             }
 //         }
 //         else{
 //             st.push(i);
 //         }
 //     }
 int longestValidParentheses(string s) {
     int open=0,close=0;
     int maxLen=0;
     int n = s.size();
     for(int i=0;i<n;i++){
         if(s[i]=='(') open++;
         else close++;
         if(open==close){
             int len = open+close;
             maxLen = max(maxLen,len);
         }
         else if(close>open){
             open=0;
             close=0;
         }
     }
     open=0,close=0;
     for(int i=n-1;i>=0;i--){
         if(s[i]=='(') open++;
         else close++;
         if(open==close){
             int len = open+close;
             maxLen = max(maxLen,len);
         }
         else if(close<open){
             open=0;
             close=0;
         }
     }
     return maxLen;
 }
```

2. Maximum no of 1's row

```
int findFirstOne(int i,int j,vector<int> a){

            int idx=-1;
            while(i<=j){
                int mid = (i+j)/2;
                if(a[mid]==1){
                    idx=mid;
                    j=mid-1;
                }
                else i=mid+1;
            }
            return idx;
        }
        int maxOnes (vector <vector <int>> &Mat, int N, int M)
        {
            // your code here
            // find the transition point :)
            int idx=0;
            int ans=0;
            for(int i=0;i<N;i++){
                int index = findFirstOne(0,M-1,Mat[i]);
                if(index>=0 and M - index > ans){
                    idx=i;
                    ans = M-index;
                }
            }
            return idx;
        }
```

3.Sequential Digits

```
queue<int> q;

        for(int i=1;i<=9;i++){
            q.push(i);
        }
        vector<int> ans;
        while(!q.empty()){
            int num = q.front(); // 1
            q.pop();
            if(num>=low and num<=high)
                ans.push_back(num);
            if(num%10 < 9) // last digit < 9
            {
                int rem = num%10; // 1
                q.push(num*10+rem+1); // 12
            }
        }
        return ans;
```

4.Nearly Sorted Array (K Sorted Array)

```
void sortK(int arr[], int n, int k)

{
    int size;
    size=(n==k)?k:k+1;
    priority_queue<int, vector<int>, greater<int> > pq(arr, arr +size);
     int index = 0;
    for (int i = k + 1; i < n; i++) {
        arr[index++] = pq.top();
        pq.pop();
        pq.push(arr[i]);
    }

    while (pq.empty() == false) {
        arr[index++] = pq.top();
        pq.pop();
    }
}
```

5.Find in a sorted matrix

```
int i=0, j=M-1;

        while(i<N and j>=0){
            if(mat[i][j]==X) return 1;

            else if(mat[i][j]>X){
                j--;
            }
            else i++;
        }
      return 0;
```

6.Number of ways to tile a floor

```
long long solve(long long w,long long curr,vector<long long> &dp){

      if(curr>w) return 0;
      if(dp[curr]!=-1) return dp[curr];
      if(curr==w){
          return 1;
      }
      return dp[curr]= (solve(w,curr+1,dp)%mod + solve(w,curr+2,dp)%mod)%mod;
  }
    long long numberOfWays(long long w) {
        vector<long long> dp(w+1,-1);
        return solve(w,0,dp)%mod;
    }
```

7. Given a String of the form ab2c3 where the string preceding the integer is repeated that many times, you are supposed to find the Kth character of the string.

```
int main(){

    string str = "ab4c2ed3";
    int k=9;
    int i,j;
    int n = str.size();
    int len,num,freq;
    i=0;
    while(i<n){
        j = i;
        len = 0;
        freq = 0;

        while (j < n && isalpha(str[j])) {
            j++;
            len++;
        }
        while (j < n && isdigit(str[j])) {
            freq = freq * 10 + (str[j] - '0');
            j++;
        }
        num = freq * len;

        if (k > num) {
            k -= num;
            i = j;
        }
         else {
            k--;
            k %= len;
            cout <<  str[i + k];
            return 0;
        }
    }
    cout <<  str[k-1];
```

8. Find the smallest and second smallest number in an array

```
vector<int> minAnd2ndMin(int a[], int n) {

    int mini1 = INT_MAX; int mini2 = mini1;
    for(int i=0;i<n;i++){
        if(a[i]<mini1){
            mini1 = a[i];
        }
    }
    for(int i=0;i<n;i++){
        if(a[i]<mini2 and a[i]!=mini1){
            mini2 = a[i];
        }
    }
    if(mini2==INT_MAX) return {-1};
    return {mini1,mini2};
```

9.Max Distance between same elements

```
int maxDistance(int arr[], int n)

    {
        int maxi=0;
        unordered_map<int,int> mp;
        for(int i=0;i<n;i++){
            if(mp.find(arr[i])!=mp.end()){
                maxi = max(maxi,i-mp[arr[i]]);
            }
            else mp[arr[i]]=i;
        }
        return maxi;
    }
```

10. Check whether a tree is BST or not

```
bool f(Node* root,int maxi,int mini){

        if(root==NULL) return true;

        if(root->data >= maxi or root->data <= mini) return false;

        return f(root->left,root->data,mini) and f(root->right,maxi,root->data);
    }
```

11. Possible path between two vertices

```
vector<int>vis(V,0);

       queue<int>q;
       q.push(source);
       vis[source]=1;
       int count=0;
       while(!q.empty()){
           int node=q.front();
           q.pop();
           if(node==destination){
               count++;
           }
           for(auto it:adj[node]){
               if(vis[it]==0){ q.push(it);
                   vis[it]==1;       return count;
```