Ex no:1(a)	Rectangle Overlap
Date:	

AIM:

To write a program to check the rectangles are overlap or not

PSEUDOCODE:

```
isRectangleOverlap(vector<int>& rec1, vector<int>& rec2)
BEGIN
  x1=rec1[0],y1=rec1[1],x2=rec1[2],y2=rec1[3]
  x11=rec2[0],y11=rec2[1],x22=rec2[2],y22=rec2[3]
  if( x1<x22 && x11<x2 && y1<y22 && y11<y2)
     BEGIN
      return true
     END
 return false
END
```

SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;
bool isRectangleOverlap(vector<int>& rec1, vector<int>& rec2)
{
int x1=rec1[0],y1=rec1[1],x2=rec1[2],y2=rec1[3];
int x11=rec2[0],y11=rec2[1],x22=rec2[2],y22=rec2[3];
if( x1<x22 && x11<x2 && y1<y22 && y11<y2){
    return true;
        }
 return false;
}
int main(){
 int val;
 vector<int>rec1;
 vector<int>rec2;
 cout<<"Enter The Values For rectangle1 : ";
 for(int i=0;i<4;i++){
       cin>>val;
       rec1.push_back(val);
    }
 cout<<"Enter The Values For rectangle1 : ";
 for(int i=0;i<4;i++){
      cin>>val;
      rec2.push_back(val);
    }
 if(isRectangleOverlap(rec1,rec2))
cout<<"Two Rectangle are OverLapping";
else
```

```
  cout<<"Two Rectangles are Not OverLapping";
 return 0;
  }
```

OUTPUT:

```
Enter The Values For rectangle1 : 2 7 4 3
Enter The Values For rectangle1 : 9 8 3 2
Two Rectangles are Not OverLapping

...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

Thus, the above program to check the rectangles are overlap or not is executed successfully and the output is also verified.

Ex no:1(b)	Matrix Cells in Distance Order
Date:	

AIM:

To find the solution for the given cells in the matrix which is sorted to their distance from rcenter, ccenter from smallest distance to the target distance.

PSEUDOCODE:

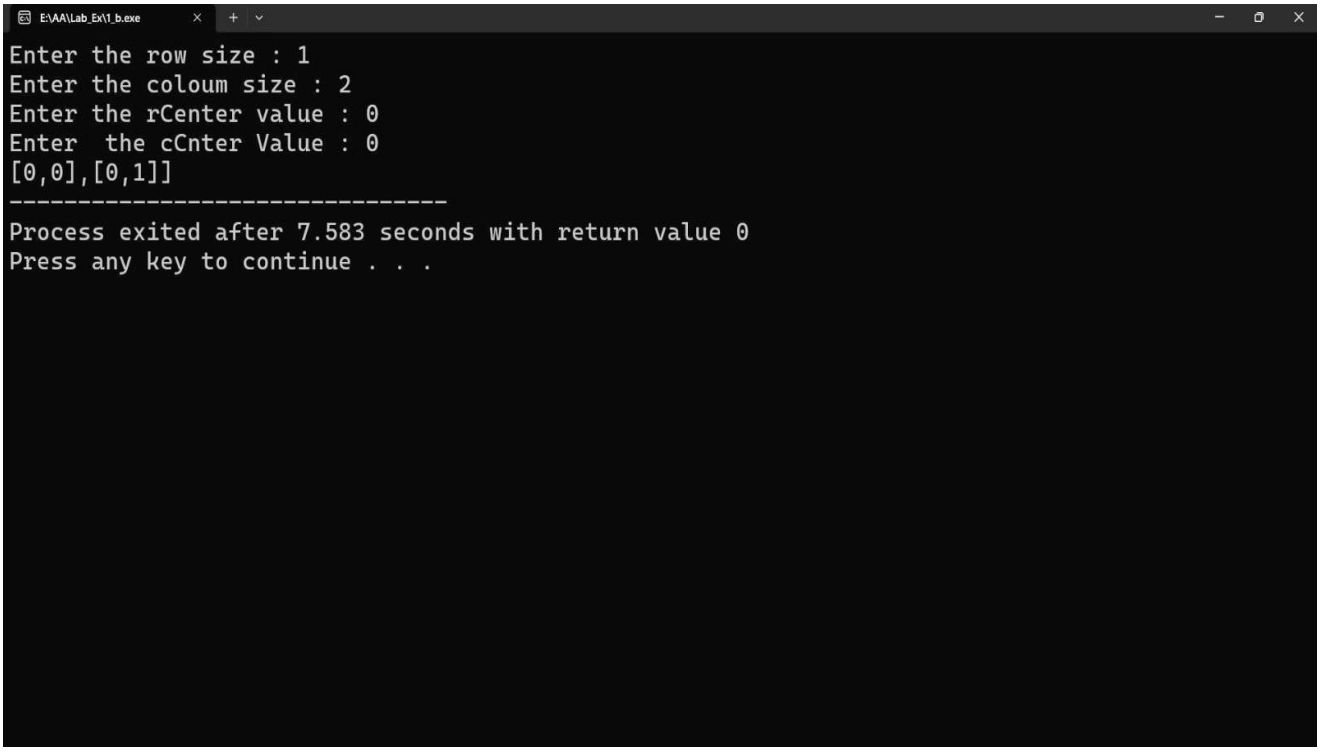
```
allCellsDistOrder(int rows, int cols, int rCenter, int cCenter)
BEGIN
   vector<int>ans
   for(int i=0;i<rows;i++)
      BEGIN
        for(int j=0;j<cols;j++)
         BEGIN
           vector<int>now
         now.push_back(i)
         now.push_back(j)
         ans.push_back(now)
         END
      END
      sort(ans.begin(),ans.end(),[&](auto &a,auto &b){
      return abs(rCenter-a[0])+abs(cCenter-a[1]) < abs(rCenter-b[0])+abs(cCenter-b[1]);
   });
   return ans
END
```

SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;
vector<vector<int>>
allCellsDistOrder(int rows, int cols, int rCenter, int cCenter)
{
   vector<vector<int>> ans;
   for(int i=0;i<rows;i++){
      for(int j=0;j<cols;j++){
         vector<int>now;
         now.push_back(i);
         now.push_back(j);
         ans.push_back(now);
      }
   }
   sort(ans.begin(),ans.end(),[&](auto &a,auto &b){
   return abs(rCenter-a[0])+abs(cCenter-a[1]) < abs(rCenter-b[0])+abs(cCenter-b[1]);
   });
   return ans;
}
```

```
int main(){
    int rows,cols,rCenter,cCenter;
    cout<<"Enter the row size : ";
    cin>>rows;
    cout<<"Enter the coloum size : ";
    cin>>cols;
    cout<<"Enter the rCenter value : ";
    cin>>rCenter;
    cout<<"Enter the cCnter Value : ";
    cin>>cCenter;
    vector<vector<int>>res=allCellsDistOrder(rows,cols,rCenter,center);
    int row=res.size(),col=res[0].size();
    for(int i=0;i<row;i++){
    cout<<"["<<res[i][0]<<","<<res[i][1];
        if(i+1==row){
            cout<<"]";
            break;
        }
        cout<<"],";
    }
    cout<<"]";
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the above to find the solution for the given cells in the matrix which is sorted to their distance from rcenter, ccenter from smallest distance to the target distance was executed successfully and the output was also verified.

Ex no:1(c)	Find the Largest Area of Square Inside Two Rectangles
Date:	

AIM:

To write a program to find the largest area of square inside two rectangles.

PSEUDOCODE:

```
largestSquareArea(vector<vector<int>>& f, vector<vector<int>>& s)
BEGIN
   long long area=-1e9 int a
   for(int  i=0;i<f.size();i++)
   BEGIN
      for(int j=i+1;j<f.size();j++)
       BEGIN
         x1=f[i][0],y1=f[i][1],x2=s[i][0],y2=s[i][1]
         x11=f[j][0],y11=f[j][1],x22=s[j][0],y22=s[j][1]
         overlapx1=max(x1,x11)
         overlapy1=max(y1,y11)
         overlapx2=min(x2,x22)
         overlapy2=min(y2,y22)
         if( overlapx1 < overlapx2 && overlapy1 < overlapy2 )
          BEGIN
            len = overlapx2-overlapx1
            wid = overlapy2-overlapy1
            a = len < wid ? len : wid  area= area >(long long)( (long long)a*(long long)a )? area : (long
long) ((long long)a*(long long)a)
          END
       END
    END
   return area == -1e9 ? 0 : area
END
```

SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;
long long largestSquareArea(vector<vector<int>>& f, vector<vector<int>>& s)
{
   long long area=-1e9;
   int a;
   for(int          i=0;i<f.size();i++){
         for(int j=i+1;j<f.size();j++){
            int x1=f[i][0],y1=f[i][1],x2=s[i][0],y2=s[i][1];
            int x11=f[j][0],y11=f[j][1],x22=s[j][0],y22=s[j][1];
            int  overlapx1=max(x1,x11);
            int  overlapy1=max(y1,y11);
            int  overlapx2=min(x2,x22);
            int overlapy2=min(y2,y22);
```

```
                if( overlapx1 < overlapx2 && overlapy1 < overlapy2 ){
                    int len = overlapx2-overlapx1 ;
                    int wid = overlapy2-overlapy1;
                    a = len < wid ? len : wid ;
                    area= area >(long long)( (long long)a*(long long)a )? area : (long long)( (long long)a*(long
long)a) ;
                }
            }
        }
      return area == -1e9 ? 0 : area;
    }
    int  main(){
        cout<<"Enter The Number Of Rectangle :";
        int n,x,y;
        cin>>n;
        vector<vector<int>>bottomLeft;
        vector<vector<int>>topRight;
        cout<<"Enter The BottomLeft Points : ";
        for(int i=0;i<n;i++){
            cin>>x>>y;
            bottomLeft.push_back({x,y});
        }
        cout<<"Enter The TopRight Points : ";
        for(int i=0;i<n;i++){
            cin>>x>>y;
            topRight.push_back({x,y});
        }
        int a=largestSquareArea(bottomLeft,topRight);
        if(a)  {
            cout<<"Area of Square is : "<<a;
        }else
             cout<<"The Square Doesn't Exist";  return 0;
}
```

OUTPUT:

```
Enter The Number Of Rectangle :2
Enter The BottomLeft Points : 5 4 2 7 8 9
Enter The TopRight Points : 3  2 4 5 6 7
The Square Doesn't Exist

...Program finished with exit code 0
Press ENTER to exit console.
```

RESULT:

Thus, the above program to find the largest area of square inside two rectangles was executed successfully and the output was also verified.

Ex no:1(d)	K Closest Points to Origin
Date:	

AIM:

To write a program to return the k closest points to the origin.

PSEUCODE:

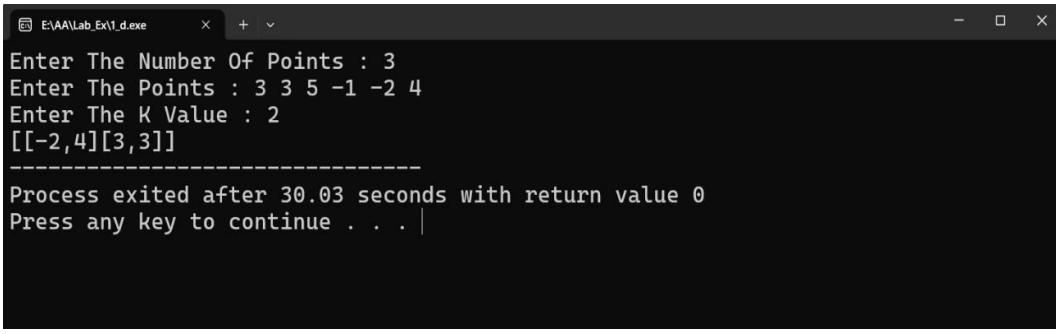
```
kClosest(vector<vector<int>>& points, int k)
BEGIN
   priority_queue<pair<double,vector<int>>,vector<pair<double,vector<int>>>,greater<>>pq;
   for(auto it : points)
     BEGIN
         int dis=sqrt(it[0]*it[0]+ it[1]*it[1])
         pq.push({dis,it})
      END
      vector<vector<int>> ans
      for(int i=0;i<k;i++)
        BEGIN
          ans.push_back(pq.top().second);
          pq.pop()
         END
   return ans
END
```

SOURCE CODE:

```
#include<bits/stdc++.h>
using namespace std;
vector<vector<int>> kClosest(vector<vector<int>>& points, int k) {
   priority_queue<pair<double,vector<int>>,vector<pair<double,vector<int>>>,greater<>>pq;
   for(auto it : points){
      int dis=sqrt(it[0]*it[0] + it[1]*it[1]);
      pq.push({dis,it});
   }
   vector<vector<int>>ans;
   for(int i=0;i<k;i++){
     ans.push_back(pq.top().second);
     pq.pop();
   }
   return ans;
}
int   main()   {
   int n,v1,v2;
   cout<<"Enter The Number Of Points : ";
   cin>>n;
   vector<vector<int>>points;
   cout<<"Enter The Points : ";
   for(int i=0;i<n;i++){
```

```
        cin>>v1>>v2;
        points.push_back({v1,v2});
    }
    int k;
    cout<<"Enter The K Value : ";
    cin>>k;
    vector<vector<int>> res=kClosest(points,k);
    cout<<"[";
    for(int i=0;i<res.size();i++){
        cout<<"["<<res[i][0]<<","<<res[i][1]<<"]";
    }
    cout<<"]";
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to return the k closest points to the origin was executed successfully and the output was also verified.

Ex.No: 2(A)	**Maximum Repeating Substring**
Date:	

AIM:

To write a program to return the maximum k-repeating value of word in sequence.

PSEUDOCODE:

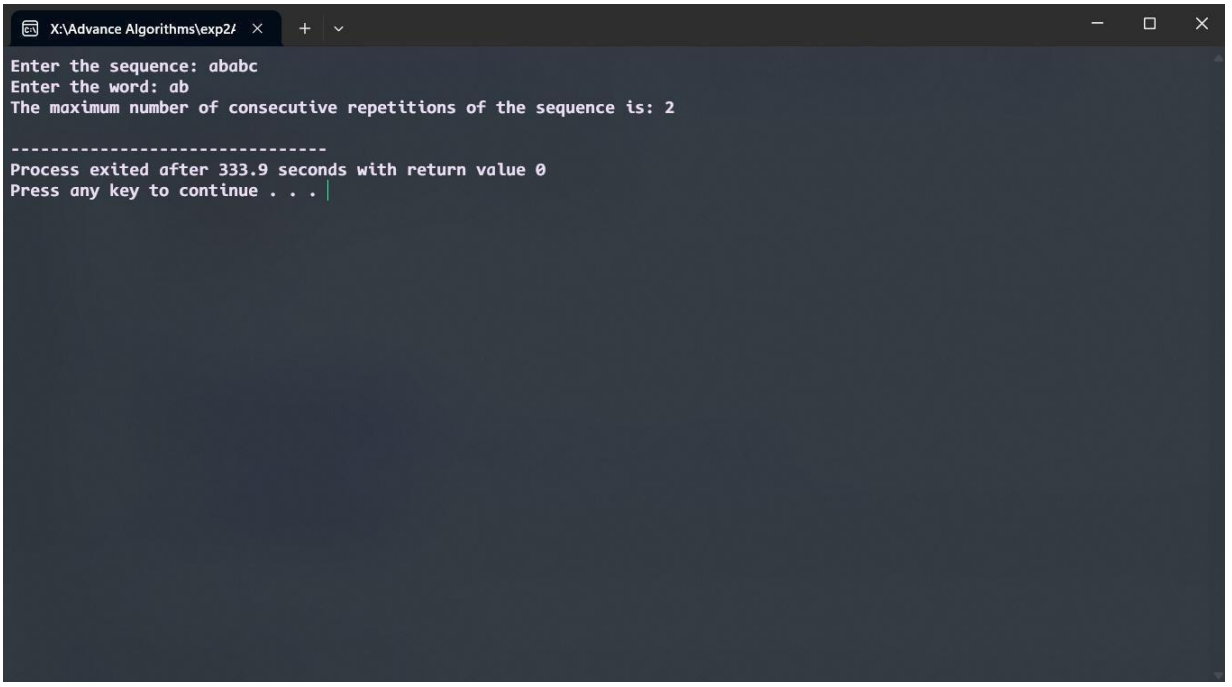
```
BEGIN:
   FUNCTION maxRepeating(sequence, word)
   BEGIN
      m=LENGTH of sequence
      n=LENGTH of word
      maxi=INT_MIN
      ans=0
      FOR i FROM 0 TO (m - n) DO BEGIN
         IF sequence substring starting from i of length n == word THEN BEGIN
            ans=ans + 1
            i=i + (n - 1)
         ENDIF
         ELSE BEGIN
            IF ans > 0 THEN BEGIN
               maxi=max(ans, maxi)
               i=i - (n - 1)
               ans=0
            ENDIF
         ENDIF
      END
      maxi=max(ans, maxi)
      RETURN maxi
   END
END
```

SOURCE CODE:

```
#include <iostream>
#include <string>
#include <climits>
using namespace std;
 int maxRepeating(string sequence, string word) {
      int m = sequence.size();
      int n = word.size();
      int maxi = INT_MIN;
      int ans = 0;
      for (int i = 0; i <= m - n; i++) {
         if (sequence.substr(i, n) == word) {
            ans++;
            i += (n - 1);
         }
         else {
            if (ans) {
               maxi = max(ans, maxi);
               i -= (n - 1);
               ans = 0;
            }
```

```
        }
    }
    maxi = max(ans, maxi);
    return maxi;
}
int main() {
  string sequence = "abababab";
  string word = "ab";
  int result = maxRepeating(sequence, word);
  cout << "Maximum number of consecutive repetitions: " << result << endl;
  return 0;
```

OUTPUT:



RESULT:

Thus, the above program to return the maximum k-repeating value of word in sequence is executed successfully and the output is also verified.

Ex.No: 2(B)	PASCAL'S TRIANGLE
Date:	

AIM:

To write a program to return the first numRows of Pascal's triangle.

PSEUDOCODE:

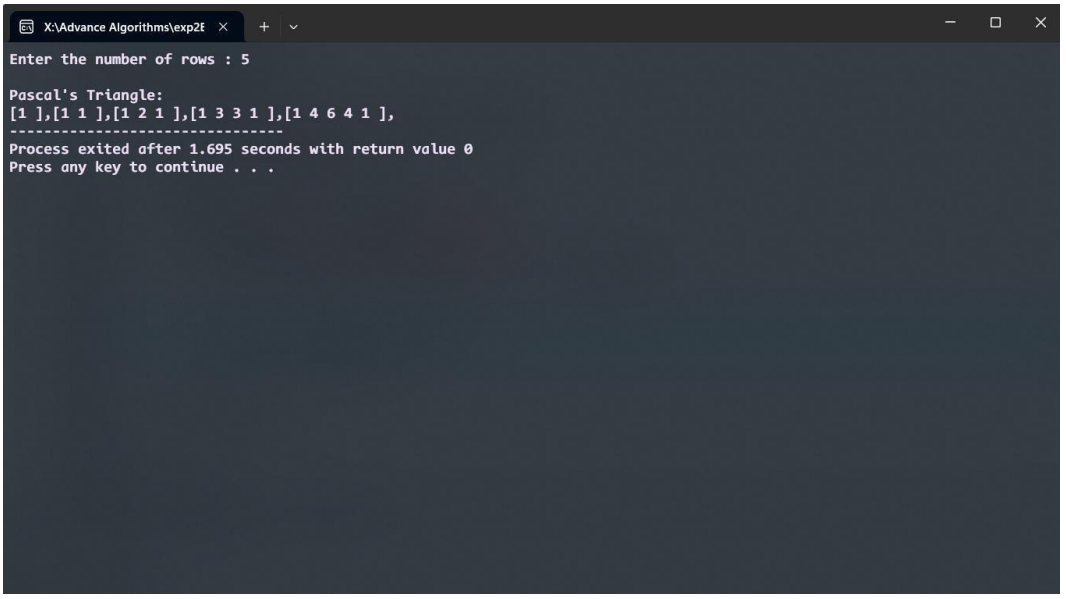
```
BEGIN:
  FUNCTION generate(numRows) BEGIN
    result = empty 2D list
    prevRow = empty list

    FOR i FROM 0 TO numRows - 1 DO BEGIN
      currentRow = list of (i + 1) elements, all initialized to 1

      FOR j FROM 1 TO i - 1 DO BEGIN
        currentRow[j] = prevRow[j - 1] + prevRow[j]
      END

      APPEND currentRow TO result
      prevRow = currentRow
    END

    RETURN result
  END
END
```

SOURCE CODE:

```
#include <iostream>
#include <vector>
using namespace std;
vector<vector<int> > generate(int numRows) {
    vector<vector<int> > result;
    vector<int> prevRow;

    for (int i = 0; i < numRows; i++) {
        vector<int> currentRow(i + 1, 1);
        for (int j = 1; j < i; j++) {
            currentRow[j] = prevRow[j - 1] + prevRow[j];
        }

        result.push_back(currentRow);
        prevRow = currentRow;
    }

    return result;
}

int main() {
    int numRows;
    cout << "Enter the number of rows : ";
    cin >> numRows;
    vector<vector<int> > triangle = generate(numRows);
    cout << "\nPascal's Triangle:\n";
    for (int i = 0; i < triangle.size(); i++) {
```

```
        cout<<"[";
        for (int j = 0; j < triangle[i].size(); j++) {
            cout << triangle[i][j] << " ";
        }
        cout << "],";
    }

    return 0;
}
```

OUTPUT:



RESULT:

Thus, the above to return the first numRows of Pascal's triangle. was executed successfully and the output was also verified.

Ex.No: 2(C)	IS SUBSEQUENCE
Date:	

AIM:

To write the C++ Program to return that the given subsequence is in the string or not .

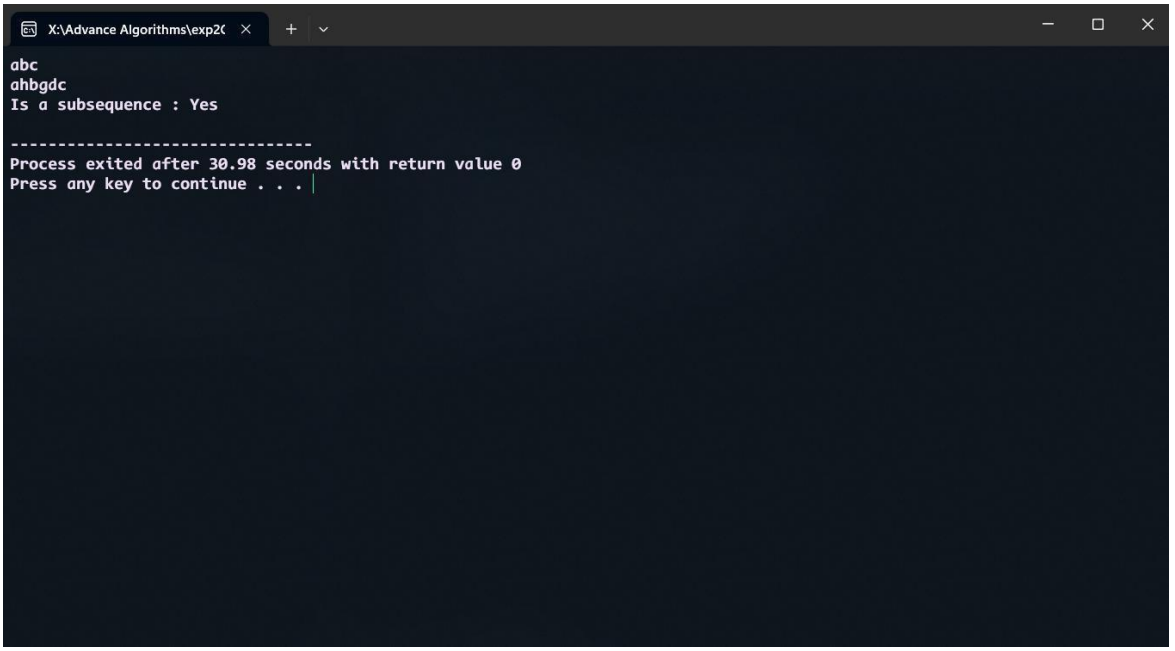
PSEUDOCODE:

```
BEGIN:
  FUNCTION isSubsequence(s, t)
  BEGIN
    n = LENGTH of s
    m = LENGTH of t
    j = 0

    FOR i FROM 0 TO m - 1 DO BEGIN
       IF s[j] == t[i] THEN
          j = j + 1
       ENDIF
    END

    RETURN (j == n)
  END
END
```

SOURCE CODE:

```
#include <iostream>
#include <string>
using namespace std;

bool isSubsequence(string s, string t) {
        int n = s.length(), m = t.length();
        int j = 0;
        for (int i = 0; i < m; i++) {
            if (s[j] == t[i]) j++;
        }
        return j == n;
    }
};

int main() {
    string s1 , t1;
    cin >> s1;
    cin >> t1;
    bool result1 = isSubsequence(s1, t1);
    cout << "Is a subsequence : “ << (result1 ? "Yes" : "No") << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to return that the given subsequence is in the string or not was successfully executed and output was verified.

Ex.No: 2(D)	COUNTING BITS
Date:	

AIM:

To write a C++ program to count number of 1 bits for the given number+1 size array.

PSEUDOCODE:

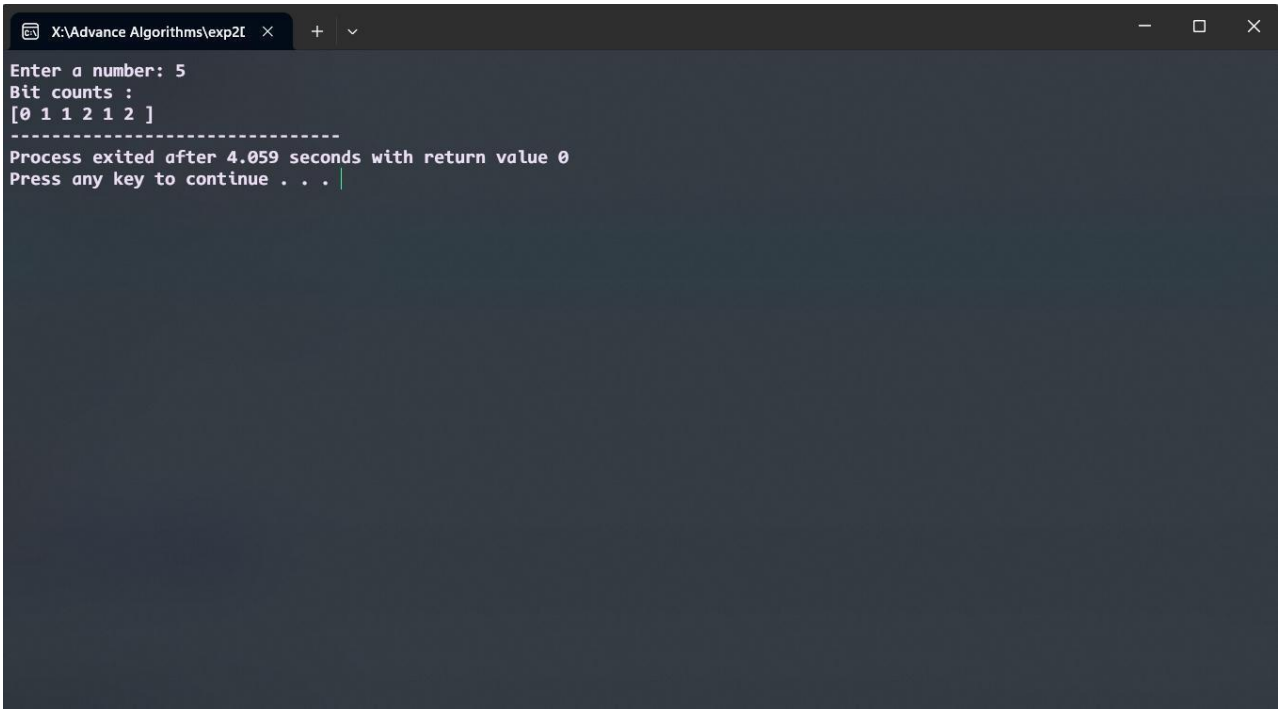
```
BEGIN:
   FUNCTION countBits(n)
       BEGIN
      Initialize dp as a vector of size (n + 1) with 0
      sub = 1

      FOR i FROM 1 TO n
       BEGIN
         IF sub * 2 == i
             sub = i
        END
         dp[i] = dp[i - sub] + 1
      END
      RETURN dp
   END
END
```

SOURCE CODE:

```
#include <iostream>
#include <vector>
using namespace std;
vector<int> countBits(int n) {
        vector<int> dp(n + 1, 0);
        int sub = 1;

        for (int i = 1; i <= n; i++) {
            if (sub * 2 == i) {
                sub = i;
            }

            dp[i] = dp[i - sub] + 1;
        }

        return dp;
}

int main() {

    int n;
    cout << "Enter a number: ";
    cin >> n;
    vector<int> result = countBits(n);

    cout << "Bit counts :" << endl;
    cout <<"[";
```

```
    for (int i = 0; i <= n; i++) {
        cout <<result[i]<<' ';
    }
         cout <<"]";
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to count number of 1 bits for the given number + 1 size array was executed successfully and output was verified.

Ex.No: 3A	SUM OF ALL SUBSET XOR TOTALS
Date:	

AIM:

To write a program to return the sum of all XOR totals for every subset of given array of nums.

PSEUDOCODE:

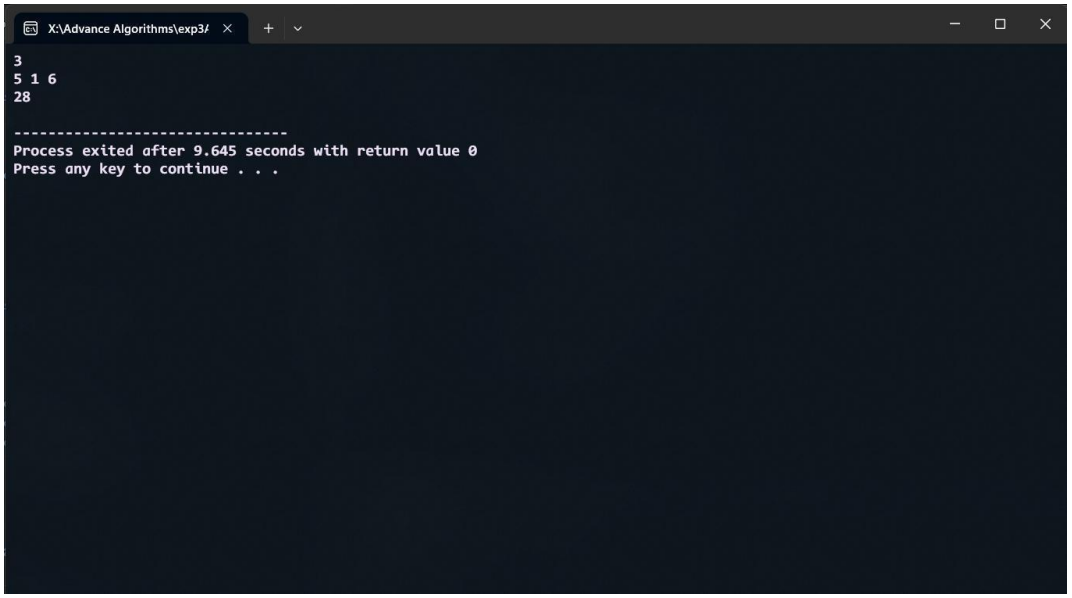
```
BEGIN
int subsetXORSum(vector<int>& nums)
BEGIN
   n = nums.size()
   totalSum = 0
   FOR i = 0 TO (1 << n) – 1
   BEGIN
      subsetXor = 0
      FOR j = 0 TO n – 1
      BEGIN
         IF (i & (1 << j))
         BEGIN
            subsetXor = subsetXor ^ nums[j]
         END
      END
      totalSum = totalSum + subsetXor
   END
   return totalSum
END
END
```

SOURCE CODE:

```
#include <iostream>
#include <vector>
using namespace std;

int subsetXORSum(vector<int>& nums) {
    int n = nums.size();
    int totalSum = 0;
    for (int i = 0; i < (1 << n); i++) {
        int subsetXor = 0;
        for (int j = 0; j < n; j++) {
            if (i & (1 << j)) {
                subsetXor ^= nums[j];
            }
        }
        totalSum += subsetXor;
    }
    return totalSum;
}

int main() {
int n;
cin >> n;
```

```
    vector<int> nums(n);
    for (int i = 0; i < n; i++) {
        cin >> nums[i];
    }
    cout << subsetXORSum(nums) << endl;
    return 0;
   }
```

OUTPUT:



RESULT:

Thus, the program to return the sum of all XOR totals for every subset of given array of nums was executed successfully and output was verified.

Ex.No: 3B	**BEAUTIFUL ARRANGEMENT**
Date:	

AIM:

To write a program to return the number of the beautiful arrangement that can be construct.

PSEUDOCODE:

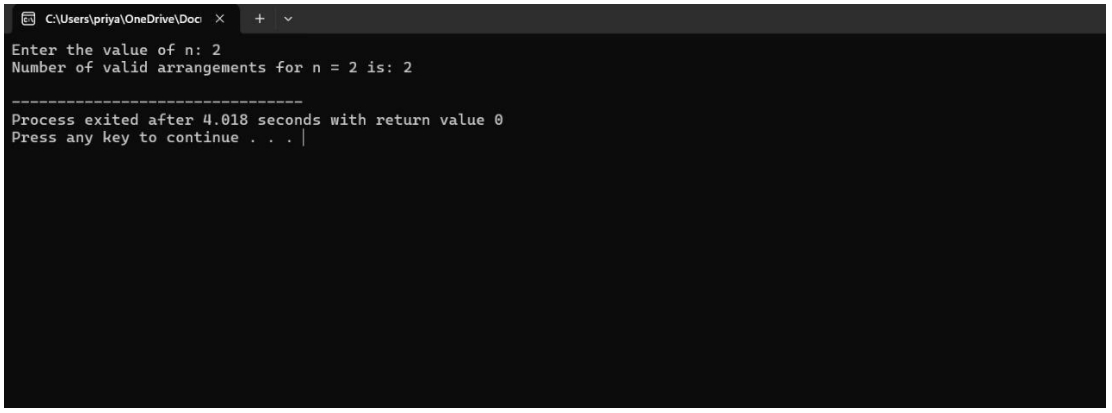
```
BEGIN
   FUNCTION countArrangement(n, pos = 1):

  IF pos > n THEN
      Increment res by 1
      RETURN

   FOR i FROM 1 TO n DO:
      bit = 1 LEFT SHIFT i

   IF (seen AND bit) is 0 AND (i MOD pos = 0 OR pos MOD i = 0) THEN
         seen = seen XOR bit
         countArrangement(n, pos + 1)
         seen = seen XOR bit

   RETURN res;
END
```

SOURCE CODE:

```
#include <iostream>
using namespace std;

class Solution {
public:
    int seen = 0;
    int res = 0;

    int countArrangement(int n, int pos = 1) {
        if (pos > n) {
            res++;
            return 0;
        }

        for (int i = 1; i <= n; i++) {
            int bit = 1 << i;
            if (!(seen & bit) && (i % pos == 0 || pos % i == 0)) {
                seen ^= bit;
                countArrangement(n, pos + 1);
                seen ^= bit;
            }
        }

        return res;
    }
};

int main() {
```

```
    Solution sol;
    int n;

    cout << "Enter the value of n: ";
    cin >> n;

    int result = sol.countArrangement(n);
    cout << "Number of valid arrangements for n = " << n << " is: " << result << endl;

    return 0;
}
```

OUTPUT:



RESULT:

Thus, the above to return the number of a beautiful arrangement and it was executed successfully and the output also verified.

Ex.No: 3C	COMBINATION SUM
Date:	

AIM:

To write the C++ Program to return that the unique combination that sum upto target .

PSEUDOCODE:

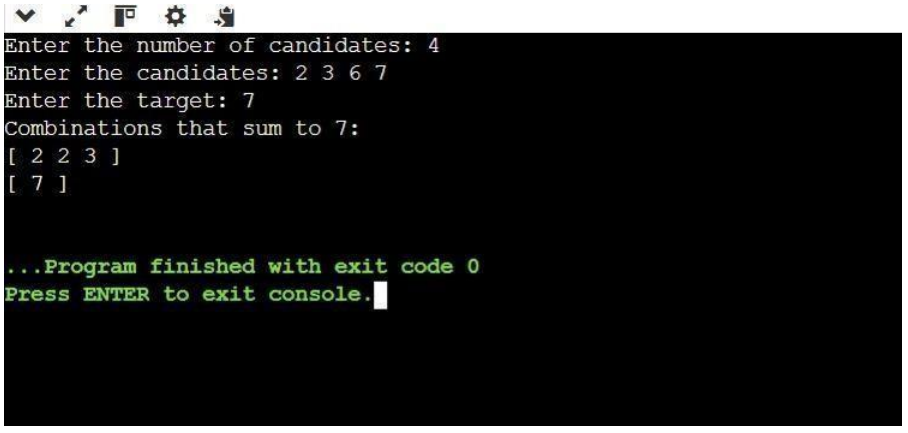
```
BEGIN
FUNCTION combinationSum(candidates, target)
   INITIALIZE result as an empty list
   INITIALIZE combination as an empty list
   CALL backtrack(candidates, target, 0, combination, 0, result)
   RETURN result

FUNCTION backtrack(candidates, target, i, combination, total, result)
   IF total == target THEN
      ADD combination to result
      RETURN

   IF total > target OR i >= length of candidates THEN
      RETURN
   ADD candidates[i] to combination
   CALL backtrack(candidates, target, i, combination, total + candidates[i], result)
   REMOVE last element from combination
   CALL backtrack(candidates, target, i + 1, combination, total, result)
END
```

SOURCE CODE:

```
#include <iostream>
#include <vector>
using namespace std;

class Solution {
public:
   vector<vector<int>> combinationSum(vector<int>& candidates, int target) {
      vector<vector<int>> res;
      vector<int> comb;
      makeCombination(candidates, target, 0, comb, 0, res);
      return res;
   }
private:
   void makeCombination(std::vector<int>& candidates, int target, int idx, vector<int>& comb, int total,
vector<vector<int>>& res) {
      if (total == target) {
         res.push_back(comb);
         return;
      }
     if (total > target || idx >= candidates.size()) {
         return;
      }
      comb.push_back(candidates[idx]);
      makeCombination(candidates, target, idx, comb, total + candidates[idx], res);
      comb.pop_back();
      makeCombination(candidates, target, idx + 1, comb, total, res);
```

```
    }
};

int main() {
    Solution solution;
    int n, target;
    cout << "Enter the number of candidates: ";
    cin >> n;
    vector<int> candidates(n);
    cout << "Enter the candidates: ";
    for (int i = 0; i < n; ++i) {
        cin >> candidates[i];
    }
    cout << "Enter the target: ";
    cin >> target;
    vector<vector<int>> result = solution.combinationSum(candidates, target);
    cout << "Combinations that sum to " << target << ":\n";
    for (const auto& combination : result) {
        cout << "[ ";
        for (int num : combination) {
            cout << num << " ";
        }
        cout << "]\n";
    }

    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to return that the combination was successfully executed and output was verified.

Ex.No: 3D	UNIQUE BINARY STRING
Date:	

AIM:

To write a C++ program to return that unique binary string of length.

PSEUDOCODE:

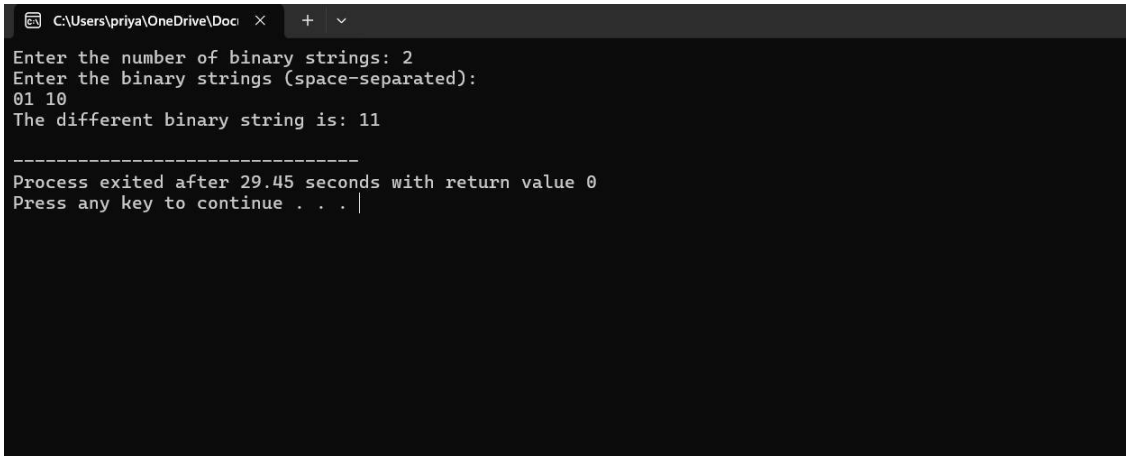
```
BEGIN
  FUNCTION findDifferentBinaryString(nums):
    SET numSet = CREATE a set from nums
    RETURN backtrack(numSet, "", LENGTH of nums)

    FUNCTION backtrack(numSet, current, n):
    IF LENGTH of current == n THEN:
       IF current NOT IN numSet THEN:
          RETURN current
       ENDIF
       RETURN
    result = backtrack(numSet, current + '0', n)
    IF result is not empty THEN:
       RETURN result
    ENDIF
  RETURN backtrack(numSet, current + '1', n)
END
```

SOURCE CODE:

```
#include <iostream>
#include <vector>
#include <unordered_set>
using namespace std;

class Solution {
public:
   string findDifferentBinaryString(vector<string>& nums) {
      unordered_set<string> numSet(nums.begin(), nums.end());
      return backtrack(numSet, "", nums.size());
   }

private:
   string backtrack(unordered_set<string>& numSet, string current, int n) {
      if (current.length() == n) {
         if (numSet.find(current) == numSet.end()) {
            return current;
         }
         return "";
      }
      string withZero = backtrack(numSet, current + '0', n);
      if (!withZero.empty()) return withZero;

      return backtrack(numSet, current + '1', n);
   }
};

int main() {
```

```
    Solution solution;

    int n;
    cout << "Enter the number of binary strings: ";
    cin >> n;

    vector<string> nums(n);
    cout << "Enter the binary strings (space-separated): ";
    for (int i = 0; i < n; ++i) {
        cin >> nums[i];
    }

    string result = solution.findDifferentBinaryString(nums);
    cout << "The different binary string is: " << result << endl;

    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to return that unique binary string of length was executed and the output was verified.

Ex No: 4A	**Detect Cycles in 2D Grid**
Date:	

AIM:

To write a program to detect the cycles in a 2D grid.

PSEUDOCODE:

```
bool containsCycle(vector<vector<char>>& grid)
BEGIN
    m = grid.size(), n = grid[0].size();
    vector<vector<int>> color(m, vector<int>(n, 0));
    cycle = 0;
    for (int i = 0; i < grid.size() && !cycle; i++)
        for (int j = 0; j < grid[0].size(); j++)
            if (color[i][j] == 0)
                dfs(i, j, -1, -1, grid, color);

    return cycle;
END
void dfs(int i, int j, int px, int py, vector<vector<char>>& grid, vector<vector<int>>& color)
    BEGIN
    color[i][j] = 1;
    int dir[4][2] = {{0, -1}, {-1, 0}, {0, 1}, {1, 0}};
    for (int idx = 0; idx < 4; idx++)
    BEGIN
        int r = i + dir[idx][0], c = j + dir[idx][1];
        if (r == px && c == py) continue;
        if (r >= 0 && c >= 0 && r < m && c < n && grid[i][j] == grid[r][c])
      BEGIN
            if (color[r][c] == 1)
          BEGIN
                cycle = 1;
                break;
            END
            if (color[r][c] == 0)
                dfs(r, c, i, j, grid, color);
        END
    END
    color[i][j] = 2;
END
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;
int m, n, cycle;
void dfs(int i, int j, int px, int py, vector<vector<char> >& grid, vector<vector<int> >& color) {
    color[i][j] = 1;
    int dir[4][2] = {{0, -1}, {-1, 0}, {0, 1}, {1, 0}};
    for (int idx = 0; idx < 4; idx++) {
        int r = i + dir[idx][0], c = j + dir[idx][1];
        if (r == px && c == py) continue;
        if (r >= 0 && c >= 0 && r < m && c < n && grid[i][j] == grid[r][c]) {
            if (color[r][c] == 1) {
                cycle = 1;
                return;
            }
            if (color[r][c] == 0)
                dfs(r, c, i, j, grid, color);
        }
    }
    color[i][j] = 2;
}
bool containsCycle(vector<vector<char> >& grid) {
    m = grid.size(), n = grid[0].size();
    vector<vector<int> > color(m, vector<int>(n, 0));
    cycle = 0;
    for (int i = 0; i < m && !cycle; i++)
        for (int j = 0; j < n; j++)
            if (color[i][j] == 0)
                dfs(i, j, -1, -1, grid, color);
    return cycle;
}

int main() {
    cout << "Enter the number of rows and columns: ";
    cin >> m >> n;

    vector<vector<char> > grid(m, vector<char>(n));
    cout << "Enter the grid characters row-wise:" << endl;
    for (int i = 0; i < m; i++) {
        for (int j = 0; j < n; j++) {
            cin >> grid[i][j];
        }
    }

    if (containsCycle(grid)) {
        cout << "Cycle detected in the grid." << endl;
    } else {
        cout << "No cycle found in the grid." << endl;
    }

    return 0;
}
```

OUTPUT:

```
C:\Users\praveen\Documents
Enter the number of rows and columns: 4 4
Enter the grid characters row-wise:
P P P P
P S S P
P S S P
P P P P
Cycle detected in the grid.

--------------------------------
Process exited after 23.31 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus, the program to detect the cycles in a 2D grid was executed successfully and output was verified.

Ex No: 4B	**Detect Capital**
Date:	

AIM:

To write a program to determine whether the capitalization of letters in a given word follows one of the three correct capitalization patterns.

PSEUDOCODE:

```
bool detectCapitalUse(string word)
BEGIN
    if(word.size()==1)
            return true;
        if(isupper(word[0])){
            if(isupper(word[1])==false){
                for(int i=1;i<word.size();i++)
                if(isupper(word[i]))
                    return false;
            else
                for(int i=1;i<word.size();i++)
                    if(islower(word[i]))
                        return false;
        else
            for(int i=0;i<word.size();i++)
                if(isupper(word[i]))
                    return false;
        return true;
END
```

CODE:

```
#include<bits/stdc++.h>
using namespace std;
bool detectCapitalUse(string word) {
    int fl=0;
    if(word.size()==1)
        return true;
    if(isupper(word[0])){
        int sl;
        if(isupper(word[1])==false){
            for(int i=1;i<word.size();i++){
            if(isupper(word[i])) return false;
             }
        }else{
            for(int i=1;i<word.size();i++){
            if(islower(word[i])) return false;
        }
        }
    }
    else{
        for(int i=0;i<word.size();i++)
            if(isupper(word[i]))
                return false;
```

```
        }
    return true;

    }
int main(){

        string str;
        int n;
        cin>>n;
        while(n--){
        cout<<"Enter the String : ";
        cin>>str;
        if(detectCapitalUse(str))
                cout<<"True"<<endl;
        else
                cout<<"False"<<endl;
        }


        return 0;
}
```

OUTPUT:

```
C:\Users\praveen\Documents
2
Enter the String : Praveen
True
Enter the String : praveenc
True

--------------------------------
Process exited after 9.039 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus, the above program to determine whether the capitalization of letters in a given word follows one of the three correct capitalization patterns was executed successfully and output was verified.

Ex No: 4C	**Determine Colour of a Chessboard Square**
Date:	

AIM:

To write a program to determine colour of a chessboard square.

PSEUDOCODE:

```
bool squareIsWhite(string coordinates)
BEGIN
        bool chess(string str) {
           int x=(str[0]-97);
           int y=str[1]+0;
           if(x%2!=0&&y%2!=0)
              return true;
           if(x%2==0 && y%2!=0)
                              return false;
           if(x%2!=0 && y%2 ==0)
              return false;
           return true;
        }

END
```

CODE:

```
#include<bits/stdc++.h>
using namespace std;
bool chess(string str) {
        int x=(str[0]-97);
        int y=str[1]+0;
        if(x%2!=0&&y%2!=0)
           return true;
        if(x%2==0 && y%2!=0)
                              return false;
        if(x%2!=0 && y%2 ==0)
           return false;
        return true;
     }
int main(){
           string str;
           cout<<"Enter the place on chess board : ";
           cin>>str;
           if(chess(str))
                    cout<<"Square is White.."<<endl;
           else
                    cout<<"Square is not White..."<<endl;
           return 0;
}
```

OUTPUT:

```
C:\Users\praveen\Documents
Enter the place on chess board : a1
Square is not White...

--------------------------------
Process exited after 2.123 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus, the program to determine colour of a chessboard square was executed successfully and output was verified.

Ex No: 4D	**Linked List Cycle**
Date:	

AIM:

To write a program to detect whether a given singly linked list contains a cycle by checking if any node is revisited during traversal.

PSEUDOCODE:

```
bool hasCycle(ListNode *head)
BEGIN
   ListNode *fast = head;
   ListNode *slow = head;
   while (fast != NULL && fast->next != NULL)
   BEGIN
      slow = slow->next;
      fast = fast->next->next;
      if (fast == slow)
         return true;
   END
   return false;
END
```

CODE:

```
#include  <iostream>
using namespace std;
struct ListNode {
   int val;
   ListNode *next;
   ListNode(int x) :
    val(x), next(NULL) {}
};
bool hasCycle(ListNode *head) {
   ListNode *fast = head;
   ListNode *slow = head;
   while (fast != NULL && fast->next != NULL) {
      slow = slow->next;
      fast = fast->next->next;
      if (fast == slow)
         return true;
   }
   return false;
}
int main() {
   int n, val, pos;
   cout << "Enter the number of nodes: ";
   cin >> n;
   if (n <= 0) {
      cout << "Invalid number of nodes!" << endl;
      return 1;
   }
   cout << "Enter the values of the nodes: ";
   ListNode *head = NULL, *tail = NULL;
   for (int i = 0; i < n; i++) {
      cin >> val;
      ListNode *newNode = new ListNode(val);
```

```
            if (head == NULL) {
                head = newNode;
                tail = newNode;
            } else {
                tail->next = newNode;
                tail = newNode;
            }
        }
        cout << "Enter the position to create a cycle (-1 for no cycle): ";
        cin >> pos;
        if (pos >= 0 && pos < n) {
            ListNode *temp = head;
            for (int i = 0; i < pos; i++)
                temp = temp->next;
            tail->next = temp;
        }
        if (hasCycle(head))
            cout << "The linked list has a cycle." << endl;
        else
            cout << "The linked list does not have a cycle." << endl;
        return 0;
    }
```

OUTPUT:

```
C:\Users\praveen\Documents
Enter the number of nodes: 10
Enter the values of the nodes: 9 3 4 4 2 7 2 0 9 0
Enter the position to create a cycle (-1 for no cycle): 2
The linked list has a cycle.

--------------------------------
Process exited after 24.37 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus, the program to detect whether a given singly linked list contains a cycle by checking if any node is revisited during traversal was executed successfully and output was verified.

Ex No: 5A	Find the Town Judge
Date:	

AIM:

To write a program to find the Judge of the Town.

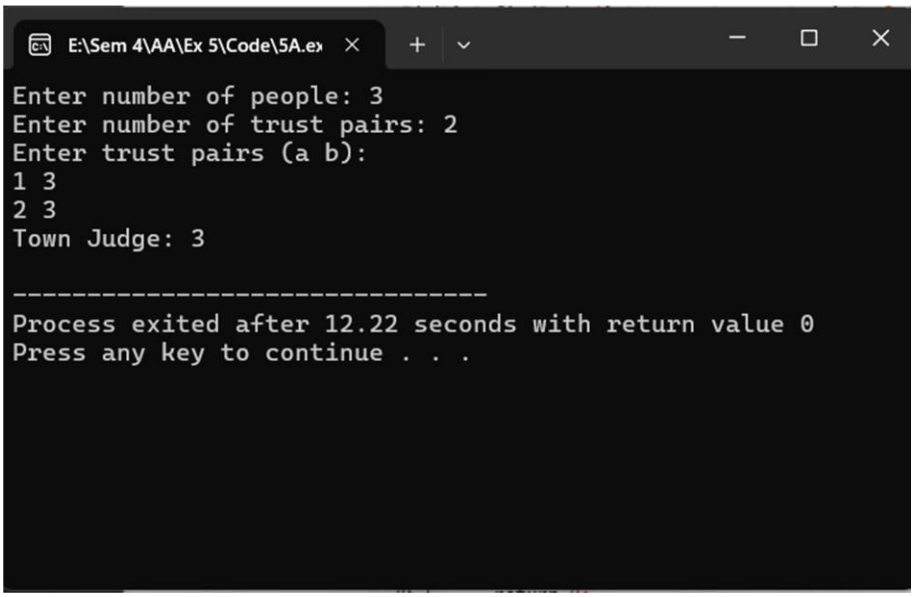
PSEUDOCODE:

```
int findJudge(int N, vector<vector<int>>& trust)
BEGIN
    vector<pair<int, int>> arr(N + 1, {0, 0});
    for (int i  =  0; i  < trust.size();  ++i)
    BEGIN
        arr[trust[i][0]].first += 1;
        arr[trust[i][1]].second += 1;
    END
    for (int i = 1; i <= N; ++i)
        if (arr[i].first == 0 && arr[i].second == N - 1)
            return i;
    return -1;
END
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;
int findJudge(int N, vector<vector<int>>& trust) {
    vector<pair<int, int>> arr(N + 1, {0, 0});
    for (int i = 0; i < trust.size(); ++i) {
        arr[trust[i][0]].first += 1;
        arr[trust[i][1]].second += 1;
    }
    for (int i = 1; i <= N; ++i)
        if (arr[i].first == 0 && arr[i].second == N - 1)
            return i;
    return -1;
}
int main() {
    int N, T;
    cout << "Enter number of people: ";
    cin >> N;
    cout << "Enter number of trust pairs: ";
    cin >> T;

    vector<vector<int>> trust(T, vector<int>(2));
    cout << "Enter trust pairs (a b):" << endl;
    for (int i = 0; i < T; i++)
        cin >> trust[i][0] >> trust[i][1];

    cout << "Town Judge: " << findJudge(N, trust) << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to find the Judge of the Town was executed successfully and output was verified.

Ex No: 5B	Find if Path Exists in Graph
Date:	

AIM:

To write a program to find the path exists from the source to destination in the graph.

PSEUDOCODE:

```
int find(int x)
BEGIN
   if (parent[x]  ==  x)  return  x;
   return parent[x] = find(parent[x]);
END

void unionSet(int x, int y)
BEGIN
   int rootX = find(x), rootY = find(y);
   if (rootX != rootY)
    BEGIN
      if  (rankSet[rootX]  >  rankSet[rootY])  parent[rootY]  =  rootX;
      else if (rankSet[rootX] < rankSet[rootY]) parent[rootX] = rootY;
      else
      BEGIN
         parent[rootY] = rootX;
         rankSet[rootX]++;
      END
    END
END

bool validPath(int n, vector<vector<int>>& edges, int source, int destination)
BEGIN
    parent.resize(n);
   rankSet.resize(n, 0);
   for (int i = 0; i < n; i++) parent[i] = i;
   for (auto& edge : edges) unionSet(edge[0], edge[1]);
   return find(source) == find(destination);
END
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;

vector<int> parent, rankSet;

int find(int x) {
   if (parent[x] == x) return x;
   return parent[x] = find(parent[x]);
}

void unionSet(int x, int y) {
   int rootX = find(x), rootY = find(y);
   if (rootX != rootY) {
```

```
        if (rankSet[rootX] > rankSet[rootY]) parent[rootY] = rootX;
        else if (rankSet[rootX] < rankSet[rootY]) parent[rootX] = rootY;
        else {
            parent[rootY] = rootX;
            rankSet[rootX]++;
        }
    }
}

bool validPath(int n, vector<vector<int>>& edges, int source, int destination) {
    parent.resize(n);
    rankSet.resize(n, 0);
    for (int i = 0; i < n; i++) parent[i] = i;
    for (auto& edge : edges) unionSet(edge[0], edge[1]);
    return find(source) == find(destination);
}

int main() {
    int n, e, src, dest;
    cout << "Enter number of nodes: ";
    cin >> n;
    cout << "Enter number of edges: ";
    cin >> e;
    parent.resize(n);
    rankSet.resize(n, 0);
    for (int i = 0; i < n; i++) parent[i] = i;
    vector<vector<int>> edges;
    cout << "Enter edges (u v):" << endl;
    for (int i = 0; i < e; i++) {
        int u, v;
        cin >> u >> v;
        edges.push_back({u, v});
    }
    cout << "Enter source and destination nodes: ";
    cin >> src >> dest;
    cout << (validPath(n, edges, src, dest) ? "Valid Path Exists" : "No Valid Path") << endl;
    return 0;
}
```

OUTPUT:

```
E:\Sem 4\AA\Ex 5\Code\5B.ex
Enter number of nodes: 3
Enter number of edges: 3
Enter edges (u v):
0 1
1 2
2 0
Enter source and destination nodes: 0 2
Valid Path Exists

--------------------------------
Process exited after 31.97 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus, the program to find the path exists from the source to destination in the graph was executed successfully and output was verified.

Ex No: 5C	Lowest Common Ancestor of a Binary Tree
Date:	

AIM:

To write a program to find the lowest common ancestor of two different nodes in the binary tree.

PSEUDOCODE:

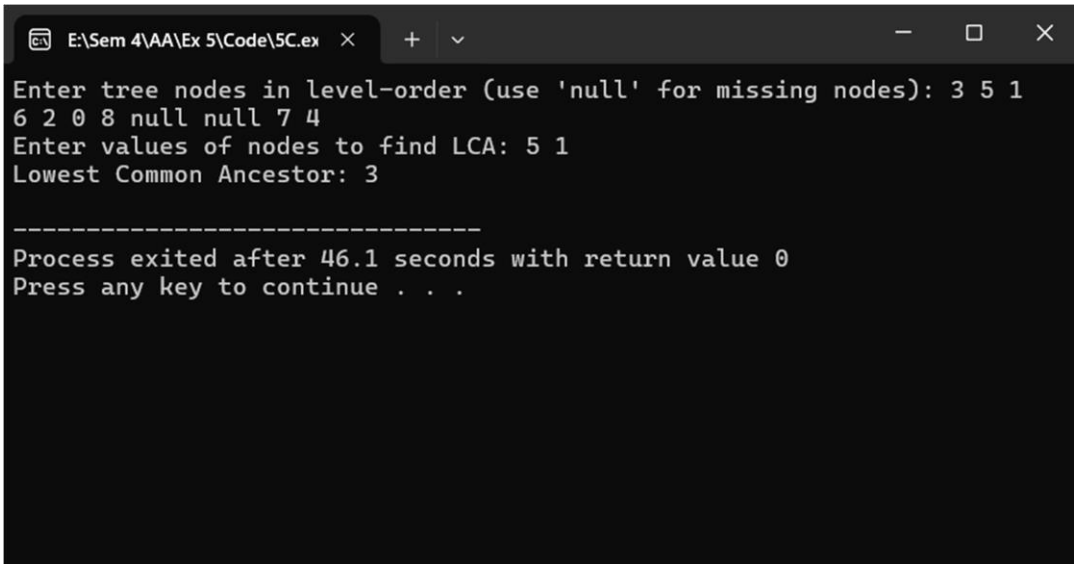
```
TreeNode* lowestCommonAncestor(TreeNode* r, TreeNode* p, TreeNode* q)
BEGIN
   if (r == NULL || r == p || r == q) return r;
   TreeNode* lt = lowestCommonAncestor(r->left, p, q);
   TreeNode* rt = lowestCommonAncestor(r->right, p, q);
   if (lt == NULL) return rt;
   if (rt == NULL) return lt;
   return r;
END
```

CODE:

```
#include <iostream>
#include <vector>
#include <queue>
#include <sstream>
using namespace std;

struct TreeNode {
    int val;
    TreeNode *left, *right;
    TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};

TreeNode* lowestCommonAncestor(TreeNode* r, TreeNode* p, TreeNode* q) {
    if (r == NULL || r == p || r == q) return r;
    TreeNode* lt = lowestCommonAncestor(r->left, p, q);
    TreeNode* rt = lowestCommonAncestor(r->right, p, q);
    if (lt == NULL) return rt;
    if (rt == NULL) return lt;
    return r;
}

TreeNode* buildTree(vector<string>& nodes) {
    if (nodes.empty() || nodes[0] == "null") return NULL;

    TreeNode* root = new TreeNode(stoi(nodes[0]));
    queue<TreeNode*> q;
    q.push(root);
    int i = 1;

    while (!q.empty() && i < nodes.size()) {
        TreeNode* curr = q.front();
        q.pop();

        if (nodes[i] != "null") {
            curr->left = new TreeNode(stoi(nodes[i]));
```

```
            q.push(curr->left);
        }
        i++;

        if (i < nodes.size() && nodes[i] != "null") {
            curr->right = new TreeNode(stoi(nodes[i]));
            q.push(curr->right);
        }
        i++;
    }
    return root;
}

TreeNode* findNode(TreeNode* root, int val) {
    if (!root) return NULL;
    if (root->val == val) return root;

    TreeNode* left = findNode(root->left, val);
    if (left) return left;

    return findNode(root->right, val);
}

int main() {
    string input;
    cout << "Enter tree nodes in level-order (use 'null' for missing nodes): ";
    getline(cin, input);

    vector<string>  nodes;
    stringstream ss(input);
    string temp;
    while (ss >> temp) nodes.push_back(temp);

    TreeNode* root = buildTree(nodes);

    int pVal, qVal;
    cout << "Enter values of nodes to find LCA: ";
    cin >> pVal >> qVal;

    TreeNode* p = findNode(root, pVal);
    TreeNode* q = findNode(root, qVal);

    if (!p || !q) {
        cout << "One or both nodes not found in tree!" << endl;
        return 0;
    }

    TreeNode* ancestor = lowestCommonAncestor(root, p, q);
    cout << "Lowest Common Ancestor: " << ancestor->val << endl;

    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to find the lowest common ancestor of two different nodes in the binary tree was executed successfully and output was verified.

Ex No: 5D	Find Common Characters
Date:	

AIM:

To write a program to find the common characters in the array of strings.

PSEUDOCODE:

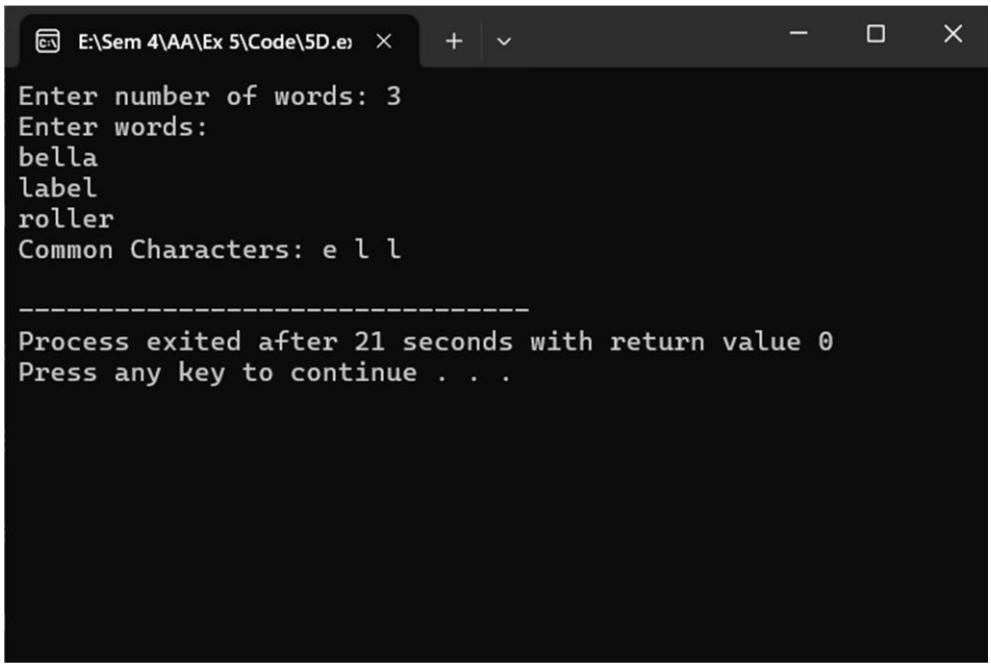
```
vector<string> commonChars(vector<string>& words)
BEGIN
   vector<int> finalFreq(26, INT_MAX);
   for (const string& word : words)
   BEGIN
      vector<int> curFreq(26, 0);
      for (char c : word) curFreq[c - 'a']++;
      for (int i = 0; i < 26; i++)
         finalFreq[i] = min(finalFreq[i], curFreq[i]);
   END
   vector<string> ans;
   for (int i = 0; i < 26; i++)
   BEGIN
      while (finalFreq[i]-- > 0)
         ans.push_back(string(1, 'a' + i));
   END
   return ans;
END
```

CODE:

```
#include <iostream>
#include <vector>
#include <string>
#include <climits>
using namespace std;

vector<string> commonChars(vector<string>& words) {
   vector<int> finalFreq(26, INT_MAX);
   for (const string& word : words) {
      vector<int> curFreq(26, 0);
      for (char c : word) curFreq[c - 'a']++;
      for (int i = 0; i < 26; i++)
         finalFreq[i] = min(finalFreq[i], curFreq[i]);
   }

   vector<string> ans;
   for (int i = 0; i < 26; i++) {
      while (finalFreq[i]-- > 0)
         ans.push_back(string(1, 'a' + i));
   }
   return ans;
}

int main() {
```

```
    int n;
    cout << "Enter number of words: ";
    cin >> n;
    vector<string> words(n);
    cout << "Enter words:" << endl;
    for (int i = 0; i < n; i++) cin >> words[i];
    vector<string> result = commonChars(words);
    cout << "Common Characters: ";
    for (const string& ch : result) cout << ch << " ";
    cout << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to find the common characters in the array of strings was executed successfully and output was verified.

Ex No: 6A	Make a Square with the Same Color
Date:	

AIM:

To write a program to make a 2 x 2 square with same color.

PSEUDOCODE:

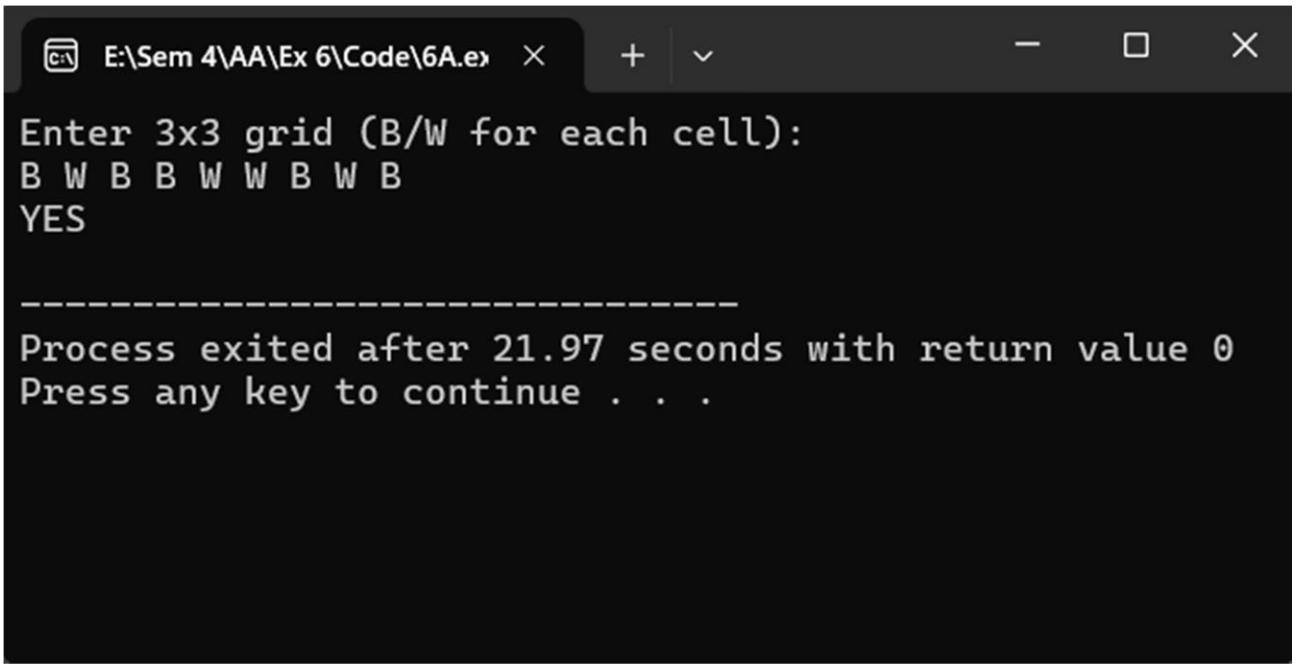
```
bool canMakeSquare(vector<vector<char>>& grid)
BEGIN
    for (int i = 0; i < 2; i++)
    BEGIN
        for (int j = 0; j < 2; j++)
       BEGIN
            int countB = 0, countW = 0;
            for (int x = i; x < i + 2; x++)
            BEGIN
                for (int y = j; y < j + 2; y++)
                BEGIN
                    if (grid[x][y] == 'B') countB++;
                    else countW++;
                END
            END
            if (countB == 4 || countW == 4 || countB == 3 || countW == 3)
                return true;
        END
    END
    return false;
END
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;

bool canMakeSquare(vector<vector<char>>& grid) {
    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < 2; j++) {
            int countB = 0, countW = 0;
            for (int x = i; x < i + 2; x++) {
                for (int y = j; y < j + 2; y++) {
                    if (grid[x][y] == 'B') countB++;
                    else countW++;
                }
            }
            if (countB == 4 || countW == 4 || countB == 3 || countW == 3)
                return true;
        }
    }
    return false;
}
```

```
int main() {
    vector<vector<char>>  grid(3,  vector<char>(3));
    cout << "Enter 3x3 grid (B/W for each cell):\n";
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            cin >> grid[i][j];

    cout << (canMakeSquare(grid) ? "YES" : "NO") << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to make a 2 x 2 square with same color was executed successfully and output was verified.

Ex No: 6B	Check if Two Chessboard Squares Have the Same Color
Date:	

AIM:

To write a program to check if two chessboard squares have the same color.

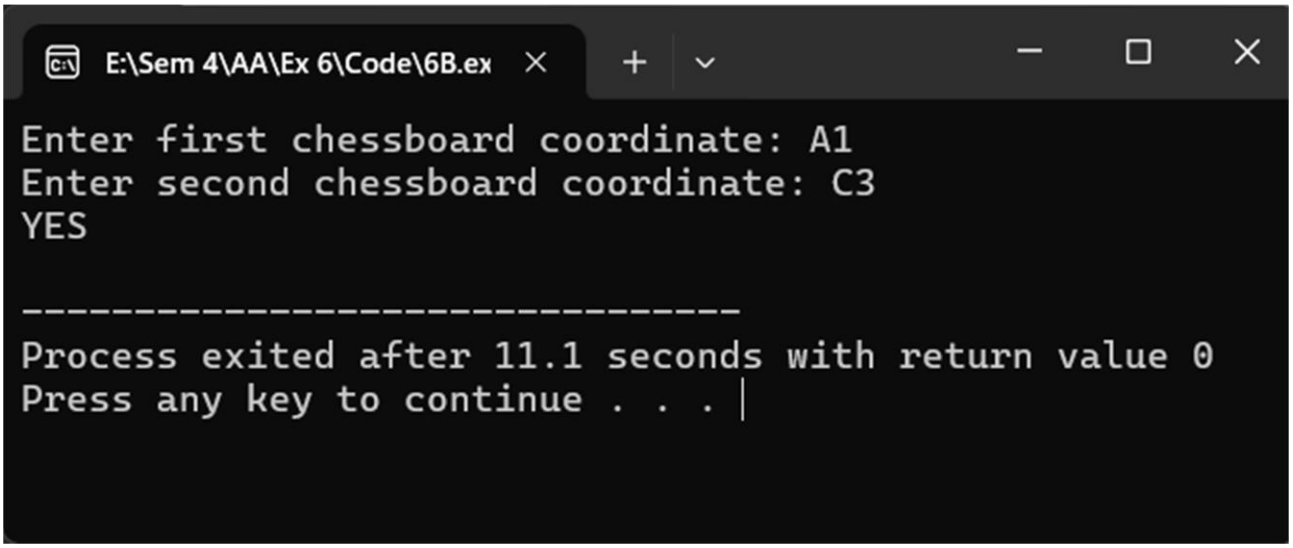
PSEUDOCODE:

```
bool checkTwoChessboards(string coordinate1, string coordinate2)
BEGIN
   map<char, int> ref = {
      {'a', 1}, {'b', 2}, {'c', 3}, {'d', 4},
      {'e', 5}, {'f', 6}, {'g', 7}, {'h', 8}
   };
   return (ref[coordinate1[0]] + (coordinate1[1] - '0')) % 2 ==
         (ref[coordinate2[0]] + (coordinate2[1] - '0')) % 2;
END
```

CODE:

```
#include <iostream>
#include <map>

using namespace std;

bool checkTwoChessboards(string coordinate1, string coordinate2) {
   map<char, int> ref = {
      {'a', 1}, {'b', 2}, {'c', 3}, {'d', 4},
      {'e', 5}, {'f', 6}, {'g', 7}, {'h', 8}
   };

   return (ref[coordinate1[0]] + (coordinate1[1] - '0')) % 2 ==
         (ref[coordinate2[0]] + (coordinate2[1] - '0')) % 2;
}

int main() {
   string coordinate1, coordinate2;
   cout << "Enter first chessboard coordinate: ";
   cin >> coordinate1;
   cout << "Enter second chessboard coordinate: ";
   cin >> coordinate2;

   cout << (checkTwoChessboards(coordinate1, coordinate2) ? "YES" : "NO") << endl;
   return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to check if two chessboard squares have the same color was executed successfully and output was verified.

Ex No: 6C	Minimum Operations to Make Columns Strictly Increasing
Date:	

AIM:

To write a program to find the minimum operations to make columns strictly increasing.

PSEUDOCODE:

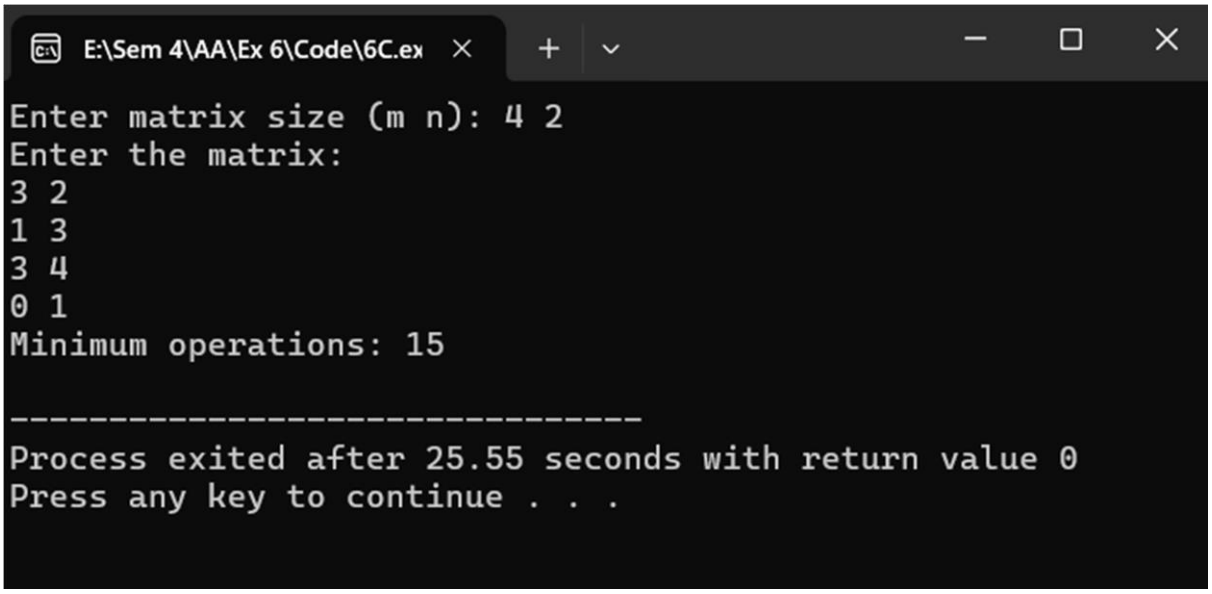
```
int minimumOperations(vector<vector<int>>& grid)
BEGIN
   int totOps = 0;
   int m = grid.size();
   int n = grid[0].size();
   for (int c = 0; c < n; c++)
   BEGIN
      int prev = -1;
      for (int r = 0; r < m; r++)
      BEGIN
         int curr = grid[r][c];
         if (!(curr > prev))
         BEGIN
            int numOps = prev - curr + 1;
            totOps += numOps;
            prev = curr + numOps;
         END
         else
            prev = curr;
      END
   END
   return totOps;
END
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;
int minimumOperations(vector<vector<int>>& grid) {
   int totOps = 0;
   int m = grid.size();
   int n = grid[0].size();
   for (int c = 0; c < n; c++) {
      int prev = -1;
      for (int r = 0; r < m; r++) {
         int curr = grid[r][c];
         if (!(curr > prev)) {
            int numOps = prev - curr + 1;
            totOps += numOps;
            prev = curr + numOps;
         } else {
            prev = curr;
         }
      }
```

```
        }
        return totOps;
    }

    int main() {
        int m, n;
        cout << "Enter matrix size (m n): ";
        cin >> m >> n;
        vector<vector<int>> grid(m, vector<int>(n));
        cout << "Enter the matrix:\n";
        for (int i = 0; i < m; i++)
            for (int j = 0; j < n; j++)
                cin >> grid[i][j];
        cout << "Minimum operations: " << minimumOperations(grid) << endl;
        return 0;
    }
```

OUTPUT:



RESULT:

Thus, the program to find the minimum operations to make columns strictly increasing was executed successfully and output was verified.

Ex No: 6D	Check if Every Row and Column Contains All Numbers
Date:	

AIM:

To write a program to check if every row and column contains all numbers from 1 to n.

PSEUDOCODE:

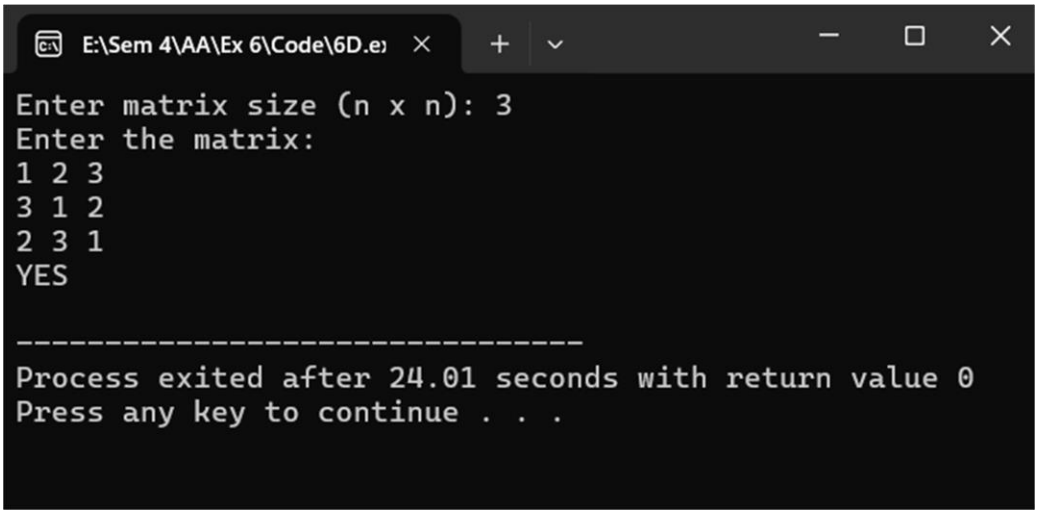
```
bool checkValid(vector<vector<int>>& matrix)
BEGIN
   int n = matrix.size();
   for (int i = 0; i < n; i++)
   BEGIN
      unordered_set<int> rowSet, colSet;
      for (int j = 0; j < n; j++)
      BEGIN
         rowSet.insert(matrix[i][j]);
         colSet.insert(matrix[j][i]);
      END
      if (rowSet.size() != n || colSet.size() != n)
         return false;
   END
   return true;
END
```

CODE:

```
#include <iostream>
#include <vector>
#include <unordered_set>

using namespace std;

bool checkValid(vector<vector<int>>& matrix) {
   int n = matrix.size();
   for (int i = 0; i < n; i++) {
      unordered_set<int> rowSet, colSet;
      for (int j = 0; j < n; j++) {
         rowSet.insert(matrix[i][j]);
         colSet.insert(matrix[j][i]);
      }
      if (rowSet.size() != n || colSet.size() != n) {
         return false;
      }
   }
   return true;
}

int main() {
   int n;
   cout << "Enter matrix size (n x n): ";
   cin >> n;
   vector<vector<int>> matrix(n, vector<int>(n));
```

```
    cout << "Enter the matrix:\n";
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            cin >> matrix[i][j];

    cout << (checkValid(matrix) ? "YES" : "NO") << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to check if every row and column contains all numbers from 1 to n was executed successfully and output was verified.

Ex No: 7A	Repeated Substring Pattern
Date:	

AIM:

To write a program to check whether the given string is constructed by using repeated substring of it.

PSEUDOCODE:

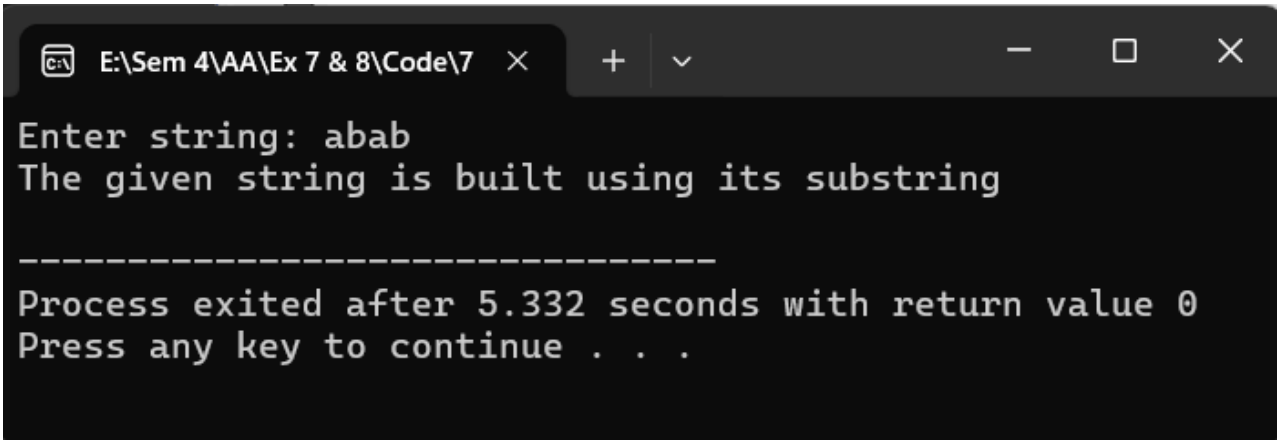
```
bool repeatedSubstringPattern(string s)
BEGIN
    int n = s.size();
    string substr1 = "";
    int subsize = 1 ;
    while(subsize <= n/2 )
    BEGIN
        if(n%subsize == 0 )
        BEGIN
            substr1 = s.substr(0,subsize);
            string summa = substr1;
            while(summa.size() <= n )
            BEGIN
                if(s == summa)
                    return true;
                else
                    summa += substr1 ;
            END
        END
        subsize++;
    END
    return false;
END
```

CODE:

```
#include <iostream>
#include <string>
using namespace std;

bool repeatedSubstringPattern(string s) {
    int n = s.size();
    string substr1 = "";
    int subsize = 1;
    while (subsize <= n / 2) {
        if (n % subsize == 0) {
            substr1 = s.substr(0, subsize);
            string summa = substr1;
            while (summa.size() <= n) {
                if (s == summa) {
                    return true;
                } else {
                    summa += substr1;
                }
            }
        }
```

```
        subsize++;
    }
    return false;
}

int main() {
    string s;
    cout << "Enter string: ";
    cin >> s;
    cout << (repeatedSubstringPattern(s) ? "The given string is built using its substring": "The given string is
not built using its substring") << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to check whether the given string is constructed by using repeated substring of it was executed successfully and output was verified.

Ex No: 7B	Substring Matching Pattern
Date:	

AIM:

To write a program to check if the given pattern is matches with the string.

PSEUDOCODE:

```
bool hasMatch(string s, string p)
BEGIN
   if (p.find('*') == string::npos)
      return s == p;
   size_t starPos = p.find('*');
   string lft = p.substr(0, starPos);
   string rht = p.substr(starPos + 1);
   size_t lftIdx = s.find(lft);
   if (lftIdx == string::npos) return false;
   size_t rhtIdx = s.find(rht, lftIdx + lft.length());
   if (rhtIdx == string::npos) return false;
   return true;
END
```

CODE:

```
#include <iostream>
#include <string>
using namespace std;

bool hasMatch(string s, string p) {
   if (p.find('*') == string::npos)
      return s == p;
   size_t starPos = p.find('*');
   string lft = p.substr(0, starPos);
   string rht = p.substr(starPos + 1);
   size_t lftIdx = s.find(lft);
   if (lftIdx == string::npos) return false;
   size_t rhtIdx = s.find(rht, lftIdx + lft.length());
   if (rhtIdx == string::npos) return false;
   return true;
}

int main() {
   string s, p;
   cout << "Enter string: ";
   cin >> s;
   cout << "Enter pattern: ";
   cin >> p;
   cout << (hasMatch(s, p) ? "Both the String and the Pattern mtached" :
    "The Pattern is not matched with the String") << endl;
   return 0;
}
```

OUTPUT:

```
E:\Sem 4\AA\Ex 7 & 8\Code\7

Enter string: leetcode
Enter pattern: ee*e
Both the String and the Pattern matched

--------------------------------
Process exited after 12.72 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus, the program check if the given pattern is matches with the string was executed successfully and output was verified.

Ex No: 7C	Special Positions in a Binary Matrix
Date:	

AIM:

To write a program to count the special positions of the binary matrix.

PSEUDOCODE:

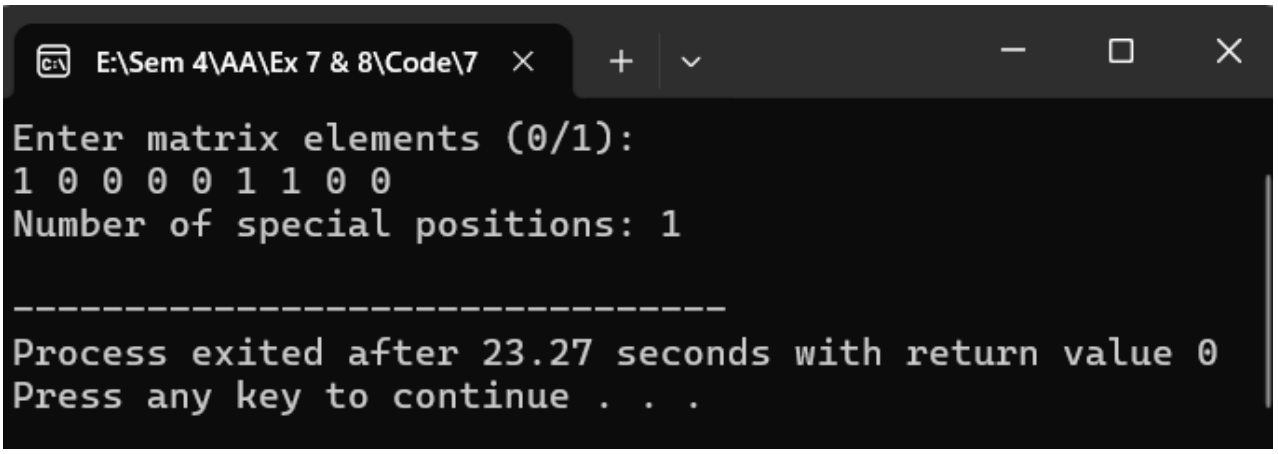
```
int numSpecial(vector<vector<int>>& mat)
BEGIN
   vector<int> rows(mat.size()), cols(mat[0].size());
   for (int i = 0; i < rows.size(); i++)
   BEGIN
      for (int j = 0; j < cols.size(); j++)
      BEGIN
         if (mat[i][j])
         BEGIN
            ++rows[i], ++cols[j];
         END
      END
   END
   int ans = 0;
   for (int i = 0; i < rows.size(); i++)
   BEGIN
      for (int j = 0; j < cols.size(); j++)
      BEGIN
         if (mat[i][j] && rows[i] == 1 && cols[j] == 1)
         BEGIN
            ++ans;
         END
      END
   END
   return ans;
END
```

CODE:

```
#include <iostream>
#include <vector>
using namespace std;

int numSpecial(vector<vector<int>>& mat) {
   vector<int> rows(mat.size()), cols(mat[0].size());
   for (int i = 0; i < rows.size(); i++) {
      for (int j = 0; j < cols.size(); j++) {
         if (mat[i][j]) {
            ++rows[i], ++cols[j];
         }
      }
   }
   int ans = 0;
   for (int i = 0; i < rows.size(); i++) {
      for (int j = 0; j < cols.size(); j++) {
```

```
            if (mat[i][j] && rows[i] == 1 && cols[j] == 1) {
                ++ans;
            }
        }
    }
    return ans;
}

int main() {
    int m, n;
    cout << "Enter number of rows and columns: ";
    cin >> m >> n;
    vector<vector<int>> matrix(m, vector<int>(n));
    cout << "Enter matrix elements (0/1):\n";
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> matrix[i][j];
    cout << "Number of special positions: " << numSpecial(matrix) << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to count the special positions of the binary matrix was executed successfully and output was verified.

Ex No: 7D	Lucky Numbers in a Matrix
Date:	

AIM:

To write a program to find the lucky number in a matrix.

PSEUDOCODE:

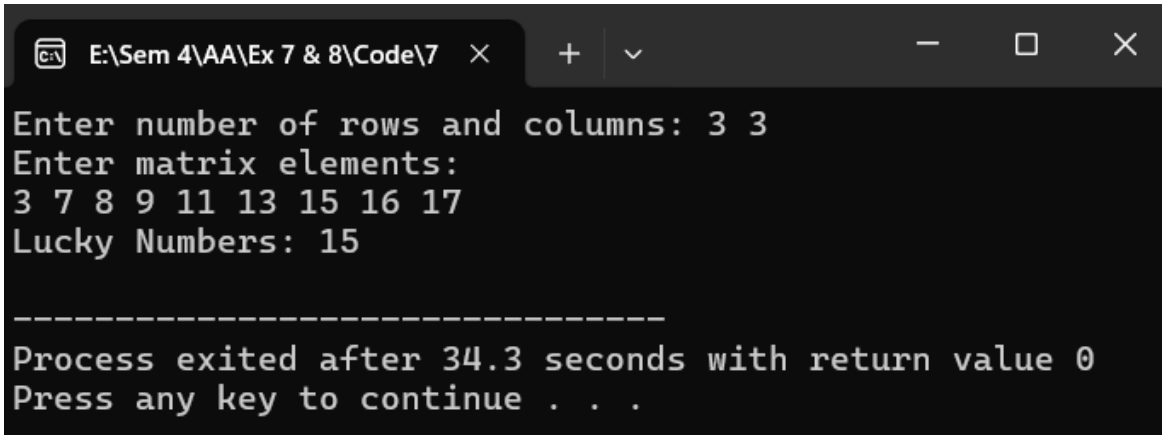
```
vector<int> luckyNumbers(vector<vector<int>>& matrix)
BEGIN
   int  m  =  matrix.size();
   int n = matrix[0].size();
   vector<int> ans, mi, ma;
   for (int i = 0; i < m; i++)
   BEGIN
      mi.push_back(*min_element(matrix[i].begin(), matrix[i].end()));
   END
   for (int i = 0; i < n; i++)
   BEGIN
      int maxi = matrix[0][i];
      for (int j = 1; j < m; j++)
      BEGIN
         if (matrix[j][i] > maxi)
            maxi = matrix[j][i];
      END
      ma.push_back(maxi);
   END
   for (int i = 0; i < ma.size(); i++)
   BEGIN
      for (int j = 0; j < mi.size(); j++)
      BEGIN
         if (mi[j] == ma[i])
         BEGIN
            ans.push_back(mi[j]);
         END
      END
   END
   return ans;
END
```

CODE:

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

vector<int> luckyNumbers(vector<vector<int>>& matrix) {
   int m = matrix.size();
   int n = matrix[0].size();
   vector<int> ans, mi, ma;
   for (int i = 0; i < m; i++) {
      mi.push_back(*min_element(matrix[i].begin(), matrix[i].end()));
```

```
    }
    for (int i = 0; i < n; i++) {
        int maxi = matrix[0][i];
        for (int j = 1; j < m; j++) {
            if (matrix[j][i] > maxi)
                maxi = matrix[j][i];
        }
        ma.push_back(maxi);
    }
    for (int i = 0; i < ma.size(); i++) {
        for (int j = 0; j < mi.size(); j++) {
            if (mi[j] == ma[i]) {
                ans.push_back(mi[j]);
            }
        }
    }
    return ans;
}

int main() {
    int m, n;
    cout << "Enter number of rows and columns: ";
    cin >> m >> n;
    vector<vector<int>> matrix(m, vector<int>(n));
    cout << "Enter matrix elements:\n";
    for (int i = 0; i < m; i++)
        for (int j = 0; j < n; j++)
            cin >> matrix[i][j];
    vector<int> result = luckyNumbers(matrix);
    cout << "Lucky Numbers: ";
    for (int num : result) {
        cout << num << " ";
    }
    cout << endl;
    return 0;
}
```

OUTPUT:



RESULT:

Thus, the program to find the lucky number in a matrix was executed successfully and output was verified.

Ex No: 8A	Minimum Positive Sum Subarray
Date:	

AIM:

To write a program to find the minimum positive sum of the subarray of the given array.

PSEUDOCODE:

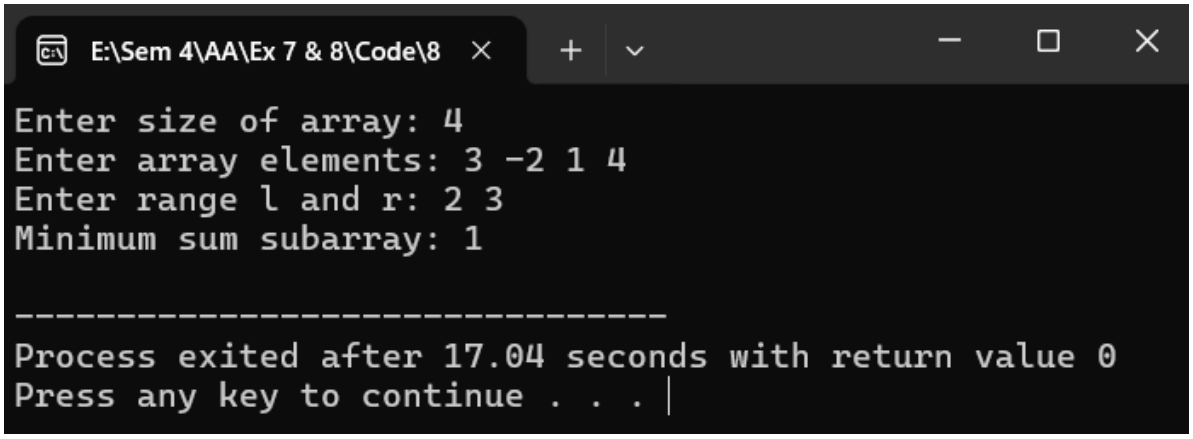
```
int minimumSumSubarray(vector<int>& nums, int l, int r)
BEGIN
   int mini = INT_MAX;
   bool flag = false;
   for (int i = l; i <= r; i++)
   BEGIN
      int sum = 0;
      for (int j = 0; j < i; j++)
      BEGIN
         sum += nums[j];
      END
      if (sum > 0)
      BEGIN
         mini = min(sum, mini);
         flag = true;
      END
      for (int k = i; k < nums.size(); k++)
      BEGIN
         sum += nums[k] - nums[k - i];
         if (sum > 0)
         BEGIN
            mini = min(sum, mini);
            flag = true;
         END
      END
   END
   return flag ? mini : -1;
END
```

CODE:

```
#include <iostream>
#include <vector>
#include <climits>
using namespace std;

int minimumSumSubarray(vector<int>& nums, int l, int r) {
   int mini = INT_MAX;
   bool flag = false;
   for (int i = l; i <= r; i++) {
      int sum = 0;
      for (int j = 0; j < i; j++) {
         sum += nums[j];
      }
      if (sum > 0) {
```

```
            mini = min(sum, mini);
            flag = true;
        }
        for (int k = i; k < nums.size(); k++) {
            sum += nums[k] - nums[k - i];
            if (sum > 0) {
                mini = min(sum, mini);
                flag = true;
            }
        }
    }
    return flag ? mini : -1;
}

int main() {
    int n, l, r;
    cout << "Enter size of array: ";
    cin >> n;
    vector<int> nums(n);
    cout << "Enter array elements: ";
    for (int &num : nums) cin >> num;
    cout << "Enter range l and r: ";
    cin >> l >> r;
    cout << "Minimum sum subarray: " << minimumSumSubarray(nums, l, r) << endl;
    return 0;
}
```

OUTPUT:



\

RESULT:

Thus, the program to find the minimum positive sum if the sub array of the given array was executed successfully and output was verified.

Ex No: 8B	Number of Distinct Averages
Date:	

AIM:

To write a program to count the distinct average.

PSEUDOCODE:

```
int distinctAverages(vector<int>& nums)
BEGIN
   set<double> s;
   sort(nums.begin(), nums.end());
   int n = nums.size();
   for (int i = 0; i < n / 2; i++)
   BEGIN
      double avg = (nums[i] + nums[n - 1 - i]) / 2.0;
      s.insert(avg);
   END
   return s.size();
END
```

CODE:

```
#include <iostream>
#include <vector>
#include <set>
#include <algorithm>
using namespace std;

int distinctAverages(vector<int>& nums) {
   set<double> s;
   sort(nums.begin(), nums.end());
   int n = nums.size();
   for (int i = 0; i < n / 2; i++) {
      double avg = (nums[i] + nums[n - 1 - i]) / 2.0;
      s.insert(avg);
   }
   return s.size();
}

int main() {
   int n;
   cout << "Enter size of array: ";
   cin >> n;
   vector<int> nums(n);
   cout << "Enter array elements: ";
   for (int &num : nums) cin >> num;
   cout << "Distinct Averages: " << distinctAverages(nums) << endl;
   return 0;
}
```

OUTPUT:

```
E:\Sem 4\AA\Ex 7 & 8\Code\8

Enter size of array: 6
Enter array elements: 4 1 4 0 3 5
Distinct Averages: 2

--------------------------------
Process exited after 17.57 seconds with return value 0
Press any key to continue . . .
```

RESULT:

Thus, the program to count the distinct average was executed successfully and output was verified.