Experiment No-3

Program:

```
#include<bits/stdc++.h>


using namespace std;


int main() {
  int N = 5, m = 6;
  vector < pair < int, int > > adj[N];
    adj[0].push_back({1,2});
            adj[0].push_back({3,6});
            adj[1].push_back({0,2});
            adj[1].push_back({2,3});
            adj[1].push_back({3,8});
            adj[1].push_back({4,5});
            adj[2].push_back({1,3});
            adj[2].push_back({4,7});
            adj[3].push_back({0,6});
            adj[3].push_back({1,8});
            adj[4].push_back({1,5});
            adj[4].push_back({2,7});
  int parent[N];
  int key[N];
  bool mstSet[N];


  for (int i = 0; i < N; i++)
    key[i] = INT_MAX, mstSet[i] = false;
  key[0] = 0;
  parent[0] = -1;
  int ansWeight = 0;
  for (int count = 0; count < N - 1; count++) {


    int mini = INT_MAX, u;
```

```
    for (int v = 0; v < N; v++) {
      if (mstSet[v] == false && key[v] < mini)
        mini = key[v], u = v;
    }
    mstSet[u] = true;

    for (auto it: adj[u]) {
      int v = it.first;
      int weight = it.second;
      if (mstSet[v] == false && weight < key[v])
        parent[v] = u, key[v] = weight;
    }

  }

  for (int i = 1; i < N; i++)
    cout << parent[i] << " - " << i << " \n";
  return 0;
}
```

Output:

```
0 - 1
1 - 2
0 - 3
1 - 4
```

Experiment No-5

Program:

```
#include <bits/stdc++.h>

using namespace std;

void lcsAlgo(string S1, string S2, int m, int n) {

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

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

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

      if (i == 0 || j == 0)

        LCS_table[i][j] = 0;

      else if (S1[i - 1] == S2[j - 1])

        LCS_table[i][j] = LCS_table[i - 1][j - 1] + 1;

      else

        LCS_table[i][j] = max(LCS_table[i - 1][j], LCS_table[i][j - 1]);

    }

  }


  int index = LCS_table[m][n];

  char lcsAlgo[index + 1];

  lcsAlgo[index] = '\0';


  int i = m, j = n;

  while (i > 0 && j > 0) {

    if (S1[i - 1] == S2[j - 1]) {

      lcsAlgo[index - 1] = S1[i - 1];

      i--;

      j--;

      index--;

    }


    else if (LCS_table[i - 1][j] > LCS_table[i][j - 1])

      i--;

    else
```

```
    j--;
  }
  for(int i=0; i<m; i++){
    for(int j=0; j<n; j++){
      cout << LCS_table[i][j] << " ";
    }
    cout << endl;
  }
  cout << "S1 : " << S1 << "\nS2 : " << S2 << "\nLCS: " << lcsAlgo << "\n";
}
int main() {
  string S1 = "ACADB";
  string S2 = "CBDA";
  int m = S1.length();
  int n = S2.length();
  lcsAlgo(S1, S2, m, n);
}
```

Output:

```
0 0 0 0 0
0 0 0 0 1
0 1 1 1 1
0 1 1 1 2
S1 : ACADB
S2 : CBDA
LCS: CB
```

Experiment No-6

Program:

```
#include<bits/stdc++.h>
using namespace std;
struct node {
   int u;
   int v;
   int wt;
   node(int first, int second, int weight) {
      u = first;
      v = second;
      wt = weight;
   }
};
int main(){
   int N=6,m=7;
   vector<node> edges;
            edges.push_back(node(0,1,5));
            edges.push_back(node(1,2,-2));
            edges.push_back(node(1,5,-3));
            edges.push_back(node(2,4,3));
            edges.push_back(node(3,2,6));
            edges.push_back(node(3,4,-2));
            edges.push_back(node(5,3,1));
   int src=0;
   int inf = 10000000;
   vector<int> dist(N, inf);
   dist[src] = 0;
   for(int i = 1;i<=N-1;i++) {
      for(auto it: edges) {
         if(dist[it.u] + it.wt < dist[it.v]) {
            dist[it.v] = dist[it.u] + it.wt;
         }
```

```
        }
    }
    int fl = 0;
    for(auto it: edges) {
        if(dist[it.u] + it.wt < dist[it.v]) {
            cout << -1;
            fl = 1;
            break;
        }
    }
    if(!fl) {
        for(int i = 0;i<N;i++) {
            cout << dist[i]<<" ";
        }
    }
    return 0;
}
```

Output:

```
0 5 3 3 1 2
```

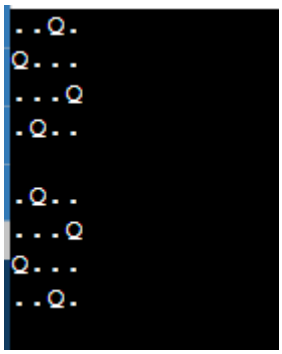
Experiment No-7

Program:

```
#include<bits/stdc++.h>
using namespace std;
vector<vector<string>> res;
   bool isValid(vector<string> arr, int r, int c, int n){
      for(int i=0;i<n;i++){
         if(arr[r][i]=='Q')return false;
      }
      for(int i=0;i<n;i++){
         if(arr[i][c]=='Q')return false;
      }
      for(int i=r,j=c;i<n && j<n; i++,j++){
         if(arr[i][j]=='Q')return false;
      }
      for(int i=r,j=c;i>=0 && j>=0; i--,j--){
         if(arr[i][j]=='Q')return false;
      }
      for(int i=r,j=c;i<n && j>=0; i++,j--){
         if(arr[i][j]=='Q')return false;
      }
      for(int i=r,j=c;i>=0 && j<n; i--,j++){
         if(arr[i][j]=='Q')return false;
      }
      return true;
   }
   void helper(vector<string> &s, int col,int n){
      if(col==n){
         res.push_back(s);
      }
      for(int i=0;i<n;i++){
         if( isValid(s,i,col,n) ){
            s[i][col]='Q';
```

```
            helper(s,col+1,n);
            s[i][col]='.';
        }
    }
}
vector<vector<string>> solveNQueens(int n) {
    vector<string> s(n);
    string a(n,'.');
    for(int i=0;i<n;i++){
        s[i]=a;
    }
    // res.push_back(s);
    helper(s,0,n);
    return res;
}
int main(){
    vector<vector<string>> v = solveNQueens(4);
    for(int i=0; i<v.size(); i++){
        for(int j=0; j<v[0].size(); j++){
            cout << v[i][j] << endl;
        }
        cout << endl;
    }
}
```

Output:



Experiment No-8

Program:

```
#include<bits/stdc++.h>
using namespace std;
class Solution {
  public:
    void solve(int ind, vector < int > & arr, int n, vector < int > & ans, int sum) {
      if (ind == n) {
        ans.push_back(sum);
        return;
      }
      solve(ind + 1, arr, n, ans, sum + arr[ind]);
      solve(ind + 1, arr, n, ans, sum);
    }
  vector < int > subsetSums(vector < int > arr, int n) {
    vector < int > ans;
    solve(0, arr, n, ans, 0);
    return ans;
  }
};
int main() {
  vector < int > arr{3,1,2};
  Solution ob;
  vector < int > ans = ob.subsetSums(arr, arr.size());
  sort(ans.begin(), ans.end());
  cout<<"The sum of each subset is "<<endl;
  for (auto sum: ans) {
    cout << sum << " ";
  }
  cout << endl;
}
```

Output:

```
The sum of each subset is
0 1 2 3 3 4 5 6
```

Experiment No-9

Program:

```
#include<iostream>

#define s 100

using namespace std;

int main(){

   int n, i, j, mat[s][s], arr[s], inv=0, num=0, input, row;

   cin>>n;

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

      for(j=1; j<=n; j++){

         cin>>input;

         arr[num++]=input;

         if(arr[num-1]==0)

            row=n-i+1;

      }

   }

   cout<<endl<<num<<endl;

   for(i=0; i<num; i++){

      for(j=i+1; j<num; j++){

         if( arr[i]>arr[j] && arr[i]!=0 && arr[j]!=0){

            inv++;

         }

      }

   }

   cout<<"NO of Inversion : "<<inv<<endl;

   if((n%2==1 && inv%2==0) || (n%2==0 && inv%2==1 && row%2==0)|| (n%2==0 && inv%2==0 &&
row%2==1))

      cout<<"Solve able\n";

   else

      cout<<"Not Possible\n";

}
```

Output:

```
2
2 3 1 0

4
NO of Inversion : 2
Solve able
```

Experiment No-10

Program:

```
#include <bits/stdc++.h>
using namespace std;
void computeLPSArray(char* pat, int M, int* lps);
void KMPSearch(char* pat, char* txt){
	int M = strlen(pat);
	int N = strlen(txt);
	int lps[M];
	computeLPSArray(pat, M, lps);
	int i = 0; // index for txt[]
	int j = 0; // index for pat[]
	while (i < N) {
		if (pat[j] == txt[i]) {
			j++;
			i++;
		}
		if (j == M) {
			printf("Found pattern at index %d ", i - j);
			j = lps[j - 1];
		}
		else if (i < N && pat[j] != txt[i]) {
			if (j != 0)
				j = lps[j - 1];
			else
				i = i + 1;
		}
	}
}
void computeLPSArray(char* pat, int M, int* lps){
	lps[0] = 0; // lps[0] is always 0
	int i = 1;
	int len = 0;
```

```
        while (i < M) {
                if (pat[i] == pat[len]) {
                        len++;
                        lps[i] = len;
                        i++;
                }
                else // (pat[i] != pat[len])
                {
                        if (len != 0) {
                                len = lps[len - 1];
                        }
                        else // if (len == 0)
                        {
                                lps[i] = 0;
                                i++;
                        }
                }
        }
}
int main(){
        char txt[] = "ABABDABACDABABCABAB";
        char pat[] = "ABABCABAB";
        KMPSearch(pat, txt);
        return 0;
}
```

Output:

```
Found pattern at index 10
```

Experiment 11:

Program:

```
#include <iostream>
using namespace std;
// an index in arr[]. n is size of heap
void heapify(int arr[], int n, int i)
{
	int largest = i; // Initialize largest as root
	int l = 2 * i + 1; // left = 2*i + 1
	int r = 2 * i + 2; // right = 2*i + 2
	// If left child is larger than root
	if (l < n && arr[l] > arr[largest])
		largest = l;
	// If right child is larger than largest so far
	if (r < n && arr[r] > arr[largest])
		largest = r;
	// If largest is not root
	if (largest != i) {
		swap(arr[i], arr[largest]);
		// Recursively heapify the affected sub-tree
		heapify(arr, n, largest);
	}
}
void heapSort(int arr[], int n)
{
	// Build heap (rearrange array)
	for (int i = n / 2 - 1; i >= 0; i--)
		heapify(arr, n, i);
	// One by one extract an element from heap
	for (int i = n - 1; i > 0; i--) {
		// Move current root to end
		swap(arr[0], arr[i]);
		// call max heapify on the reduced heap
```

```
                heapify(arr, i, 0);
        }
}
/* A utility function to print array of size n */
void printArray(int arr[], int n)
{
        for (int i = 0; i < n; ++i)
                cout << arr[i] << " ";
        cout << "\n";
}
int main()
{
        int arr[] = { 12, 11, 13, 5, 6, 7 };
        int n = sizeof(arr) / sizeof(arr[0]);
        heapSort(arr, n);
        cout << "Sorted array is \n";
        printArray(arr, n);
}
```

Output:

```
Sorted array is
5 6 7 11 12 13
```