```
void convex_hull(point p[], int n) {
    for(short i=0; i<n-1; i++) {
        for(short j=i+1; j<n; j++){
            float a = p[j].y - p[i].y;
            float b = p[i].x - p[j].x;
            float c = (p[j].x * p[i].y) - (p[i].x * p[j].y);
            if(is_convex(a, b, c, p, n, i, j))
            {   cout<<"{"p[i].x<<", "<< p[i].y
                cout<< "& ";
                cout<< p[j].x <<", "<< p[j].y<<"}";
            }
        }
    }
}
int main() {
    int i, n; cin>>n;
    point p[n];
    for(int i=0; i<n; i++)  cin>>p[i].x>>p[i].y;
    cout<<"\n Edges are: ";
    convex_hull(p, n);
}
```

ALGORITHMS USING DP:-

1) 0/1 knapsack:

```
#include<bits/stdc++.h>
using namespace std;
int ks(int wt[], int val[], int n, int w, vector<vector<int>>& v){
    if(n==0 or w<=0) return v[n][w]=0;
    else if (v[n][w] != -1) return v[n][w];
    else if (wt[n-1] <= w)
        return v[n][w] = max(val[n-1] + ks(wt, val, n-1, w-wt[n-1]),
                             ks(wt, val, n-1, w, v));
    return v[n][w] = ks(wt, val, n-1, w, v);
}
```

Tabulation:-

```
int ks(int n, int w, int wt[], int val[]) {
    int a[n+1][w+1], i, j;
    for (i=0; i<=n; i++) {
        for(j=0; j<=w; j++) {
            if(i==0 or j==0) a[i][j]=0;
            else if (wt[i-1]<=j)
                a[i][j]= max(val[i-1]+a[i-1][j-wt[i-1]],
                             a[i-1][j]);
            else a[i][j] = a[i-1][j];
        }
    }
    return a[n][w];
}
```

2) Sum of subsets.

Check for whether it is possible to form given sum.

```
#include<bits/stdc++.h>
using namespace std;
bool whether_possible(int n, int w, int wt[], vector<vector<int>>
{
    if(w==0) return v[n][w]=1;
    if(n==0) return v[n][w]=0;
    if(v[n][w] != -1) return v[n][w];
    if(wt[n-1]<=w] return v[n][w]= whether_possible(
        n-1, w-wt[n-1], wt, v) or whet_p-[n-1,w,wt,
    return v[n][w] = whether_possible(n-1, w, wt, v);
}
```

Tabulation:-

```
bool whether_possible(int n, int w, int wt[])
{
    int i, j;
    vector<vector<int>> v;
    for(i=0; i<=n; i++) {
        vector<int> temp; v.push_back(temp);
        for(j=0; j<=w; j++) {
            if(j==0) v[i].pushback(1);
            else if(i==0) v[i].push_back(0);
            else if(wt[i-1]<=j)
                v[i].push_back(v[i-1][j-wt[i-1]] or v[i-1][j]
            else v[i].push_back(v[i-1][j]);
        }
    }
}
```

3) Coin Change problem:-

```
int no_of_ways(int n, int w, int wt[]){
    if(w==0) return 1;
    if(n==0) return 0;
    if(wt[n-1]<=w)
        return(no_of_ways(n, w-wt[n-1], wt) + no_of_ways(n-1, w, wt));
    else return no_of_ways(n-1, w, wt);
}
```

min_coin problem:-

```
int fun(vector<int>&wt, int n, int w, vector<vector<int>>&dp)
{
    if(w==0) return dp[n][w]=0;
    if(n==0) return dp[n][w]= INT_MAX-10;
    if(dp[n][w] != -1) return dp[n][w];
    if(wt[n-1]<=w)
        return dp[n][w]= min(1+fun(wt, n, w-wt[n-1], dp),
                             fun(wt, n-1, w, dp));
    else
        return dp[n][w] = fun(wt, n-1, w, dp);
}
```

4) Matrix Chain multiplication:-

```
int mcm(int a[], int i, int j, vector<vector<int>> &v){
    if(i==j) return v[i][j]=0;
    if(v[i][j]!=-1) return v[i][j];
    int min=INT_MAX;
    for(int k=i; k<j; k++){
        int val = mcm(a,i,k,v)+mcm(a,k+1,j,v)+
                  (a[i-1]*a[k]*a[j]);
        if(min>val) min=val;
    }
    return v[i][j]=min;
}
```

Tabulation:-

```
int mcm(vector<vector<int>> &v, int arr[], int n){
    for(int k=0; k<n; k++){
        for(int i=0; i<n+k; i++){
            int j=i+k;
            if(i==j) v[i][j]=0;
            else{
                int min=INF_MAX;
                for(int a=i; a<j; a++){
                    int val = v[i][a]+v[a+1][j]+arr[i-1]*arr[a]*
                    if(min>val) min=val;
                }
                v[i][j]=min;
            }
        }
    }
    return v[i-1][j];
}
```