```
int main(){
    int n,w; cin>>n>>w;
    int a[n], wt[n], val[n];
    for(int i=0; i<n; i++) cin>>wt[i];
    for(int i=0; i<n; i++) cin>>val[i];
    per(a,0,n,w,wt,val);
}
```

DIVIDE AND CONQUER TECHNIC:-

1) Closest Pair Problem:-

```
#include<bits/stdc++.h>
using namespace std;
bool comparator(pair<int,int> p1, pair<int,int> p2){
    return p1.second < p2.second;
}
int distance(pair<int,int> p1, pair<int,int> p2){
    return sqrt((p1.first-p2.first)*(p1.first-p2.first))
        +((p1.second-p2.second)*(p1.second-p2.second));
}
int bruteForce(pair<int,int> points[], int n){
    int d=INT_MAX;
    for(int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            d=min(d, distance(points[i], points[j]));
        }
    }
    return d;
}
```

```
int stripClosest (pair<int,int> strip[], int n, int d)
{
    sort (strip, strip+n, comparator);
    for (int i=0; i<n; i++){
        for(int j=i+1; j<n; j++){
            if (abs(strip[i].second - strip[j].second
                d=min (d, distance (strip[i], strip[j]))
            }
        }
    }
    return d;
}

int closestPair(pair<int, int> points [], int n){
    if (n<3) return bruteForce(points, n);
    int mid= n/2;
    int dl= closestPair (points, mid);
    int dr= closestpair(points+mid, n-mid);
    int d=min(dl, dr);
    auto strip[n];
    int j=0;
    for (int i=0; i<n; i++) {
        if (abs (points[i].first - points[mid].first)<d){
            strip[j]= points[i]; j++;
        }
    }
    return min (d, stripClosest (strip, j, d));
}

int main(){
    int n; cin>>n;
    pair<int, int> points [n];
    for(int i=0; i<n; i++) {
        int x, y; cin>>x>>y;
        points[i] = {x,y};
    }
    sort (points, points+n);
    cout<< "Pair :" << closestPair (points, n);
}
```

BRUTE FORCE :-

1) Closest Pair :-

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

using namespace std;

struct point {

    int x, y;

};

int main() {

    int n, x1, x2, y1, y2;

    cin>>n;

    point p[n];

for (int i=0; i<n ; i++) cin>>p[i].x>>p[i].y;

int min= INT_MAX;

for (int i=0 ; i<=n-1 ; i++) {

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

        float dist = sqrt( pow ((p[i].x-p[j].x), 2)

                    + pow ((p[i].y-p[j].y), 2));

        if(dist<min) {

            min = dist;

            x1 = p[i].x ; y1 = p[i].y

            x2 = p[j].x ; y2 = p[j].y ;

} } }
```

```
cout<< "closest pair : " << x1 << " " << y1 << " and ";
cout<< x2 << " " << y2 << "\n min dist = " << min;
}
```

2) String Matching :-

```
# include < bits/stdc++.h>
using namespace std;
int main(){
    string m, n;
    cin>>m>>n;
    int i, flag=0, k;
    for(int i=0; i<=m.length()-n.length; i++){
        if(m[i]=n[0]){
            k=1, flag=1;
            for(int j=i+1; j<i+n.length(); j++){
                if(m[j] != n[k++]){
                    flag=0; break;
                }
            }
            if(flag){ break; }
            continue;
        }
    }
    if(flag)
        cout << "String matched from index:" << i << endl;
```

```
else
    cout << "String Not Matched! " << endl;

}
```

3) Convex-Hull Problem:-

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

using namespace std;

struct point {

    float x, y;

};

bool is_convex (float a, float b, float c, point p[], int n,
                int p1, int p2)

{   int pos_count = 0, neg_count = 0;

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

        if (r != p1 and r != p2) {

            float sign = a * p[r].x + b * p[r].y + c;

            if (sign < 0) neg_count++;

            else pos_count++;

        }

    }

    if (pos_count == n-2 and neg_count == n-2) return true;

    else return false;

}
```