Cél:

- Osztályok és objektumok
- A standard könyvtár:
  - A dinamikus tömb: `vector<T> (#include <vector>)`
  - Algoritmusok: `max_element, min_element, sort, unique (#include <algorithm>)`

I. A dinamikus tömb

```
//Definíció - üres dinamikus tömb
vector<int> v;

//Feltöltés: új elem hozzáadása: push_back VAGY emplace_back
for( int i=0; i<10; ++i ){
   v.push_back( i * 10 );
}
//Kiíratás
for(int i=0; i<v.size(); ++i ){
   cout<<v[ i ]<<" ";
}
cout<<endl;
```

II. Algoritmusok

http://www.cplusplus.com/reference/algorithm/

```
#include <iostream>
#include <vector>
#include <cstdlib>
#include <algorithm>
#define MAX 1000
using namespace std;

int main()
{
  int x[100];
  int n = 10;
  cout<<"x = [";
  for( int i=0; i<n; ++i ){
       x[i] = rand() % MAX;
       cout<<x[i]<<" ";
  }
  cout << "]" <<endl;
   cout << "Legkisebb : " << *min_element(x, x+n) << endl;

  vector<int> v;
```

```
  v.reserve(100);
  cout<<"v = [";
  for( int i=0; i<n; ++i ){
      v.emplace_back( rand() % MAX );
      cout<<v[ i]<<" ";
  }
  cout << "]" <<endl;
  cout << "Legnagyobb: " << *max_element(v.begin(),v.end()) << endl;
  return 0;
}
```

III. Kitűzött feladat

Adott a következő `Point.h` állomány.

```
#ifndef POINT_H
#define    POINT_H
#define M 2000

class Point{
private:
   int x, y;
public:
   Point( int x=0, int y=0);
   int getX() const;
   int getY() const;
   double distanceTo(const Point& point)const ;
};
#endif /* PONT_H */
```

Implementálja a tagfüggvényeket egy `Point.cpp` állományban.

Készítsen egy olyan ponthalmaz osztályt, amely páronként különböző pontokat tartalmaz. Ehhez adott egy `PointSet.h` állomány.

```
#ifndef POINTSET_H
#define    POINTSET_H

#include "Point.h"
#include <vector>
using namespace std;

class PointSet{
   //különböző pontok
   vector<Point>   points;
```

```
   //különböző pontok száma
   int n;
   //pontok kozotti tavolsagok
   vector<double> distances;
   //segédfüggvény, amely feltölti a distances tömböt
   void computeDistances();
public:
   //konstruktor: előállítja a points tömböt
   //kiszámítja a pontok közötti távolságokat (tavolsagok tömb),
   PointSet( int n = 100 );
   //maximum kiválasztás a tavolsag tömbből
   double maxDistance()   const;
   //minimum kiválasztás a tavolsag tömbből
   double minDistance()   const;
   //összesen hány távolság értelmezhető n
   //különböző pont között
   int numDistances()  const;
   //a pontok tároló kiíratása
   void printPoints()     const;
   //a tavolsagok tároló kiíratása
   void printDistances() const;
   //növekvő sorrendbe rendezi a pontokat x koordináta szerint
   void sortPointsX();
   //u. a. y koordináta szerint
   void sortPointsY();
   //a tavolsagok tároló rendezése
   void sortDistances();
   //hány darab különböző távolságot tartalmaz a tavolsagok tömb
   int numDistinctDistances();
};
#endif /* POINTSET_H */
```

1. Készítse el az osztály implementációját egy `PointSet.cpp` állományban.
2. Ellenőrizze, hogy az n növekedésével hogyan sűrűsödnek a pontok a ponthalmazban. Ehhez használhatja a következő kódrészletet:

```
int n = 2;
cout<<"Pontok\tMinTav\t  MaxTav\t    #tavolsagok\t#kulonbozotavolsagok"<<endl;
cout<< fixed;
for( int i= 0; i<12; ++i ){
   PointSet pSet( n );
   cout<<setw(6)<<n<<" ";
   cout<<setw(8)<<setprecision(2)<<pSet.minDistance()<<"  ";
   cout<<setw(8)<<setprecision(2)<<pSet.maxDistance()<<"  ";
   cout<<setw(10) << pSet.numDistances()<<"  ";
   cout<<setw(16) << pSet.numDistinctDistances()<<endl;
   n = n << 1;
}
```

3. Milyen n értékre jelenik meg a legközelebbi pontokra az 1-es távolság?
4. Milyen n értékre kezdenek ismétlődni a távolságok?