```cpp
#include "Methods.h"
#include <iostream>

void Methods::copyAll(const Methods& N){
    int i=0;

    while ( i <= N.last ){
        this->AllMethods[i] = N.AllMethods[i];
        i++;
    }
    this->last = N.last;
}

Methods::Methods() : last(-1) {}

bool Methods::isFull() {
    return last == -1;
    }

void Methods::setNumber(const RandomNumbers& n) {
    AllMethods[++last] = n;
    }

long int Methods::getNumber(const int& pos) {
    return AllMethods[pos].getRandom();
    }

void Methods::swapAllMethods(RandomNumbers& a,RandomNumbers& b) {
    RandomNumbers aux;
    aux = a;
    a = b;
    b = aux;
    }
int Methods::getLastPos() {
    return last;
    }
///Metodo Burbuja
void Methods::bubbleSort() {
    int i(last), j;
    bool flag;

    do {
        flag = false;
        j = 0;
        while(j < i) {
            if(AllMethods[j].getRandom() > AllMethods[j+1].getRandom()) {
                swapAllMethods(AllMethods[j],AllMethods[j+1]);
                flag = true;
                }
            j++;
            }
        i--;
        }
    while(flag);
    }
///Metodo Shell
void Methods::shellSort() {
    float fact(3.0/4.0);
    int dif( (last + 1 ) * fact), lim, i;

    while(dif>0) {
        lim = last - dif;

        i=0;
        while( i <= lim ) {
            if( AllMethods[i].getRandom() > AllMethods[i+dif].getRandom()) {
```

```
                swapAllMethods(AllMethods[i],AllMethods[i+dif]);
                }

            i++;
            }

        dif*=fact;
        }
    }
///Metodo insercion
void Methods::insertSort() {
    int i(1), j;
    RandomNumbers aux;

    while(i <= last) {
        aux=AllMethods[i];
        j=i;
        while( j > 0 and aux.getRandom() < AllMethods[j-1].getRandom() ) {
            AllMethods[j] = AllMethods[j-1];
            j--;
            }
        if(i!=j)
            AllMethods[j]= aux;
        i++;
        }
    }
///Metodo seleccion
void Methods::selectionSort() {
    int i(0), j, m;
    while( i < last ) {
        m = i;
        j=i+1;
        while(j<last) {
            if(AllMethods[j].getRandom() < AllMethods[m].getRandom() )
                m = j;
            j++;
            }
        if(m!=i)
            swapAllMethods(AllMethods[i],AllMethods[m]);
        i++;
        }
    }
///Metodo Mezcla
void Methods::mergeSort() {
    mergeSort(0,getLastPos());
    }

void Methods::mergeSort(const int& leftEdge, const int& rightEdge) {

    if( leftEdge >= rightEdge ) {
        return;
        }

    int  m((leftEdge + rightEdge)/2);

    mergeSort(leftEdge,m);
    mergeSort( m + 1, rightEdge);

    for( int z(leftEdge) ; z <= rightEdge ; z++ ) {
        temp[z] = AllMethods[z];
        }

    int i(leftEdge), j( m + 1 ), x(leftEdge);

    while( i <= m and j <= rightEdge) {
        while( i <= m and temp[i].getRandom() <= temp[j].getRandom() ) {
```

```
            AllMethods[x++] = temp[i++];
            }
        if( i <= m) {
            while( j <= rightEdge and temp[j].getRandom() <= temp[i].getRandom() ) {
                AllMethods[x++] = temp[j++];
                }
            }
        }
    while ( i <= m ) {
        AllMethods[x++] = temp[i++];
        }
    while( j <= rightEdge ) {
        AllMethods[x++] = temp[j++];
        }
    }
///Metodo QuickSort
void Methods::quickSort() {
    quickSort(0,getLastPos());
    }

void Methods::quickSort(const int& leftEdge, const int& rightEdge) {
    if( leftEdge >= rightEdge ) {
        return;
        }

    int i(leftEdge), j(rightEdge);

    while( i < j ) {
        while( i < j and AllMethods[i].getRandom() <= AllMethods[rightEdge].getRandom() ) {
            i++;
            }
        while( i < j and AllMethods[j].getRandom() >= AllMethods[rightEdge].getRandom() ) {
            j--;
            }
        if( i != j ) {
            swapAllMethods(AllMethods[i],AllMethods[j]);
            }
        }
    if( i != rightEdge ) {
        swapAllMethods(AllMethods[i],AllMethods[rightEdge]);
        }

    quickSort(leftEdge, i - 1);
    quickSort(i + 1, rightEdge);
    }
Methods& Methods::operator=(const Methods& n){
    copyAll(n);
    return *this;
}
```