```cpp
#include <iostream>
#include <math.h>
#include <map>
#include <string>
#include <vector>
#include <stdio.h>
using namespace std;

class Algorithms{
    public:
        int rand_int(int min, int max);
        void random_unique_list(vector<int> &arr, int min, int max, int num);
        int * find_next_largest_recursive(int l_idx, int r_idx, int & sigma);
        int * find_next_largest(int x, int & sigma);
        int * next_permutation(int & sigma);
        template <typename T> void print_vector(vector<T> array);
};

// Generate a random number in a given range
int Algorithms::rand_int(int min, int max){
    return (min + rand()) % max;
}

// Simple function to print the elements of vector
template <typename T> void Algorithms::print_vector(vector<T> array){
    int length = array.size();
    cout << "{ ";
    for(int i=0; i< length; i++){
        cout << array[i];
        if(i != length - 1){
            cout << ", ";
        }
    }
    cout << " }\n";
};

// Create random list of unique vals (Called a set)
void Algorithms::random_unique_list(vector<int> &uniqueList, int min, int max, int num){
    int i = 0;
    map<int, bool> uniqueVals;
    int randVal;
    pair<int, bool> kvp;// Key Value Pair
    while(i < num){
        randVal = rand_int(min, max);
        // check if the value is in the map
        if(uniqueVals.find(randVal) != uniqueVals.end()){
            // The value is allready in the list so don't add it
        }
        else{// The value is unique so add it
            uniqueList.push_back(randVal);
            kvp.first = randVal;// Key
            kvp.second = true;// Value
            uniqueVals.insert(kvp);
            i++;
        }
    }
}


// Lets create a Heap Class
class Heap{
            public:
                        int par(int i){return i/2;}
                        int left(int i){return 2*i + 1;}
                        int right(int i){return 2*i + 2;}
                        void MaxHeapify(int A[], int hSize, int i);
                        void MaxHeapifyIter(int A[], int hSize, int i);
                        void BuildMaxHeap(int A[], int length, bool itter);
                        void printHeap(int A[], int n);

};

// Recursively Heapify a value "bubble down"
void Heap::MaxHeapify(int A[], int hSize, int i){
            int parent = par(i);
            int l = left(i);
            int r = right(i);
            int largest = i;
            if(l < hSize && A[l] > A[i]){
                        largest = l;
            }
            if(r < hSize && A[r] > A[largest]){
                        largest = r;
```

```
            }
            if(largest != i){
                        int tmp = A[i];
                        A[i] = A[largest];
                        A[largest] = tmp;
                        MaxHeapify(A, hSize, largest);
            }
}

// Iteratively Heapify a value "bubble down"
void Heap::MaxHeapifyIter(int A[], int hSize, int i){
            int l;
            int r;
            int largest;// Remember this is the index of the largest value not the value itself!!
            int nextVal;
            // We use the same array every time
            int count = 0;
            while(true){
                        l = left(i);
                        r = right(i);
                        largest = i;
                        if(l < hSize && A[l] > A[i]){
                                    largest = l;
                                    nextVal = l;
                        }
                        if(r < hSize && A[r] > A[largest]){
                                    largest = r;
                                    nextVal = r;
                        }
                        if(largest != i){// Loop hasn't broken yet..
                                    int tmp = A[i];
                                    A[i] = A[largest];
                                    A[largest] = tmp;
                                    i = largest;
                        }
                        else{
                                    break;
                        }
                        count ++;
            }
}

// Lets create the MaxHeap from an array
void Heap::BuildMaxHeap(int A[], int length, bool itter){
            int startIdx = (length/2) - 1;
            for(int i=startIdx; i >= 0; i--){
                        if(itter){// Solve the problem with the iterative solution
                                    MaxHeapifyIter(A, length, i);
                        }
                        else{
                                    MaxHeapify(A, length, i);
                        }
            }
}

void Heap::printHeap(int A[], int n){
            cout << " { ";
            int j = 0;
            while(j<n - 1){
                        cout << A[j] << ", ";
                        j++;
            }
            cout << A[j] << "}";
};

int main(){

            Algorithms myAlg;

    cout<<"Homework 1:\n";
    vector<int> values;
    myAlg.random_unique_list(values, 3, 40, 20);
    myAlg.print_vector(values);
    cout << "\n";

    cout << "Homework 2:\n";
    cout << "Test MaxHeap: \n";
    int heapTest[] = { 1, 3, 5, 4, 6, 13, 10, 9, 8, 15, 17 };
    int length = sizeof(heapTest)/sizeof(heapTest[0]);
    Heap myHeap;
    cout << "InputArray : ";
    myHeap.printHeap(heapTest, length);
    myHeap.BuildMaxHeap(heapTest, length, true);
```

```
    cout << "\nResultantHeap: ";
    myHeap.printHeap(heapTest, length); // Schould Be: { 17, 15, 13, 9, 6, 5, 10, 4, 8, 3, 1}
    cout << "\n Should Be:  { 17, 15, 13, 9, 6, 5, 10, 4, 8, 3, 1}\n";

/*
InputArray :  { 1, 3, 5, 4, 6, 13, 10, 9, 8, 15, 17}
ResultantHeap:  { 17, 15, 13, 9, 6, 5, 10, 4, 8, 3, 1}
Should Be:  { 17, 15, 13, 9, 6, 5, 10, 4, 8, 3, 1}
 */
    return 0;
}
```