```
// Author: Eric Gustin
// Assignment: HW#3 vector_collection.h
// Description: Contains VectorCollection class definition and
// implementation. This class inherits from the pure abstract
// class called Collection. Makes use of a vector of tuples
// to represent a list of Key-Value pairs. Includes functionality
// to insert a key-value pair, remove a key-value pair,
// find and return a value associated with a key,
// find a range of keys that are between two values
// return all of the keys in the list, return the size,
// and sort the list in accending order of keys.

#ifndef VECTOR_COLLECTION_H
#define VECTOR_COLLECTION_H

#include <vector>
#include <algorithm>
#include "collection.h"

template <typename K, typename V>
class VectorCollection : public Collection<K,V>
{
 public:

  // insert a key - value pair into the collection
  void insert(const K& key, const V& val);

  // remove a key - value pair from the collection
  void remove(const K& key);

  // find and return the value associated with the key
  bool find(const K& key, V& val) const;

  // find and return the list of keys >= to k1 and <= to k2
  void find(const K& k1, const K& k2, std::vector<K>& keys)const;

  // return all of the keys in the collection
  void keys(std::vector<K>& keys) const;

  // return all of the keys in ascending ( sorted ) order
  void sort(std::vector<K>& keys) const;

  // return the number of keys in collection
  int size() const;

 private:
  std::vector<std::pair<K,V>> kv_list;

};
```

```
template<typename K, typename V>
void VectorCollection<K,V>::insert(const K& key, const V& val)
{
 std::pair<K,V> p(key, val);
 kv_list.push_back(p);
}

template<typename K, typename V>
void VectorCollection<K,V>::remove(const K& key)
{
 K curr_key;
 // iterate through kv_list's keys. if the current key
 // equals the argument that was passed though the function,
 // then the key-value pair is erased from the vector.
 for (int i = 0; i < size(); ++i) {
  curr_key = (kv_list.at(i)).first;
  if (curr_key == key) {
   kv_list.erase(kv_list.begin()+i);
  }
 }
}

template<typename K, typename V>
bool VectorCollection<K,V>::find(const K& key, V& val) const
{
 K curr_key;
 // iterates through kv_list and if the desired key is
 // found, then val is assigned to the corresponding
 // value, and returns true. Else false and val is
 // not assigned to anything. Implemented before we
 // went over for-each loops.
 for (int i = 0; i < size(); ++i) {
  curr_key = (kv_list.data()+i)->first;
  if (curr_key == key) {
   val = (kv_list.data()+i)->second;
   return true;
  }
 }
 return false;
}

template<typename K, typename V>
void VectorCollection<K,V>::find(const K& k1, const K& k2,
std::vector<K>& keys) const
{
 //for-loop variable to keep track of current index of keys vector
 unsigned int curr_index = 0;
 // set keys vector equal to kv_list
 this->keys(keys);
 //iterate through keys and remove pairs that dont meet requirements
```

```
 //curr_index is not incremented if an element is erased since all
 //elements coming after will be moved forward 1 index.
 for (int i = 0; i < size(); ++i) {
  if ((keys[curr_index] < k1) || (keys[curr_index] > k2)) {
   keys.erase(keys.begin()+curr_index);
  }
  else
   ++curr_index;
 }
}

template<typename K, typename V>
void VectorCollection<K,V>::keys(std::vector <K>& keys) const
{
 // extracts all of the keys from kv_list and assigns them to keys.
 for (int i = 0; i < size(); ++i)
  keys.push_back(kv_list[i].first);
}

template<typename K, typename V>
void VectorCollection<K,V>::sort(std::vector <K>& keys) const
{
 // calls key member function on the keys vector, then uses
 // the standard library's sort function to sort it.
 this->keys(keys);
 std::sort(keys.begin(), keys.end());
}

template<typename K, typename V>
int VectorCollection<K,V>::size() const
{
 return kv_list.size();
}

#endif
```