```
/*Author: Eric Gustin
  Assignment: CPSC223-01 HW05 binsearch_collection.h
  Description: This program uses the collection abstract class to implement
  a collection using a vector. For the insert, remove, find(2 parameter),
  find(3 parameter), this program uses the binary search algorithm. The find
  function saw the largest drop in execution time when compared with a vector
  that does not implement the binary search algorithm.
*/

#ifndef BINSEARCH_COLLECTION_H
#define BINSEARCH_COLLECTION_H

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

template <typename K, typename V>
class BinSearchCollection : 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:

// helper function for binary search
bool binsearch(const K& key, int& index) const;

// vector storage
std::vector<std::pair<K,V>> kv_list;

};

// This function returns true and sets index if key is found in
// kv_list , and returns false and sets index to where key should go in
// kv_list otherwise . If list is empty , index is unchanged .
template <typename K, typename V>
bool BinSearchCollection <K,V>::binsearch(const K& key, int& index) const
{
 // implementation of binary search
 int low = 0;
 int high = size();
 int mid = (high + low) / 2;
 while (high >= low)  {
  mid = (high + low) / 2;
  // this if-break statement prevents comparing key to an index that is beyond the
  // length of the vector.
  if (mid == size())
   break;
  if (key > (kv_list.begin()+mid)->first)
   low = mid + 1;
  else if (key < (kv_list.begin()+mid)->first)
   high = mid - 1;
  else {
   index = mid;
```

```
   return true;
  }
 }
 // index of where key would be if it was in the list
 index = low;
 return false;
}

template <typename K, typename V>
void BinSearchCollection <K,V>::insert(const K& key, const V& val)
{
 int index = -1;
 // find index to insert pair. if size() = 0 then it will insert at index 0.
 binsearch(key, index);
 std::pair<K,V> p(key, val);
 kv_list.insert(kv_list.begin()+index, p);
}

template <typename K, typename V>
void BinSearchCollection <K,V>::remove(const K& key)
{
 int index = -1;
 // if key is in the vector, then remove it
 if (binsearch(key, index))
  kv_list.erase(kv_list.begin() + index);
}

template <typename K, typename V>
bool BinSearchCollection <K,V>::find(const K& key, V& val) const
{
 int index;
 bool in_vector = binsearch(key, index);
 if (size() > 0)
  val = (kv_list.begin() + index)->second;
 return in_vector;
}

template <typename K, typename V>
void BinSearchCollection <K,V>::find(const K& k1, const K& k2, std::vector<K>& keys) const
{
 int curr_index;
 // finds the index of k1, or if k1 is not in the list then it will find where it would be
 binsearch(k1, curr_index);
 // linearly iterate through vector, appending keys to the keys vector, until
 // k2 is reached.
 while ((curr_index < size()) && (kv_list.begin()+curr_index)->first <= k2) {
  keys.push_back((kv_list.begin()+curr_index)->first);
  ++curr_index;
 }
}

template <typename K, typename V>
void BinSearchCollection <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 BinSearchCollection <K,V>::sort(std::vector<K>& keys) const
{
 // keys are inserted in order, thus the list is already sorted!
 this->keys(keys);
}

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

#endif
```