Week 2 Homework: Krystal M.

Statement of Purpose: Intersect Bag1 with Bag2 and push to Bag3 the common elements between Bag1 and Bag2

Pre-Condition: Assume Bag1 and Bag2 are not empty and may be sorted (although sorted may not be a requirement). Assume elements may be duplicated in both lists and contain elements of the same type. Assume same depth of both Bags (ie non-nested values). If Bag1 or Bag2 is empty, return -1. Bag1 and Bag2 may contain duplicates and lengths of Bag1 and Bag2 may not be same. Since Bag 3 is the intersection between two bags, Bag3 should be empty. Max size for Bags (1, 2, 3) is 20 (small sample size).

Method: Intersects -> Arguments: Bag1, Bag2, returns-> push_back -> add to Bag3

Class Bag

+Bag() //default Bag constructor
+bool isEmpty() //check bag empty
+bool add(add items to bag)
+bool remove(remove item from bag)
+void clear(remove all items from bag)
+bool contains(elem)
+bool intersects(Bag1, Bag2)
+int bagSize(int count)
+int Frequency(int freqnum) -> number of times an elem is in smaller Bag

-itemCount -maxItems -elem -> given element in larger bag -freqnum -> number of times in bag

```
1. //Bag1 = {1, 2, 3, 4, 5}
2. //Bag2 = {2, 4, 6, 8, 10, 12}
3.
4. Test-> is bag empty?
5. bool isEmpty(Bag1) //false
6. bool isEmpty(Bag2) //false
7.
8. find out which is larger bag:
9.bagSize(Bag1) -> returns count  //5
10.bagSize(Bag2) -> returns count //6
11.
12.//check larger Bag against smaller bag -> find items in larger bag.
13.//Once you know what is in larger bag, you would just have to go through second, smaller Bag....
14.//...Once (O(n)) to see if it (ie elem) is contained in smaller Bag(ie Bag1)
15.//in this example, Bag2 is larger
16.//elem -> elements in larger bag //we now know elements of this bag by checking Bag2.contains(elem)
17.
18.
19.//now we check whether Bag1.contains(elem) //ie elements of Bag2
20.
21.//we know the elements of Bag2-> are they contained in Bag1?
22.//if we do know elem, how many times is elem in Bag1 (use Frequency function) -> return freqnum
23.//using this, we can add these values to Bag3 (elem * freqnum) // we push the value of elem to Bag3 number of freqnum times
24.
25. //Pseudocode
26.template<class ItemType>
27.Bag<T> intersects(BagInterface<T> const & Bag1, BagInterface<T> const& Bag2):
28.
29.for (unsigned int i= 0; i < Bag1.size(); i++){
30.                     if (Bag1.contains(elem)) // if elem in Bag2 is contained in Bag1
31.                        Frequency( elem1) // get frequency of item
32.                     Bag3.add(add(elem) * freqnum); // add elem * frequency (add item to Bag3 number of times in freqnum)
33.               }
34.        }
35.
36.          // check if Bag3 contains -1. If it does, Bag3 shows no intersection between Bag1 and Bag2
37.
38.
```

```
36.          // check if Bag3 contains -1. If it does, Bag3 shows no intersection between Bag1 and Bag2
37.
38.
39.        return cout << "Intersection of Bag1 and Bag2 are " << Bag3; // complete intersect  Bag3
40.            (or cout << "Intersection Bag1 and Bag2 is [ ]") // no common elements
41.}
```