Week 10 Section Handout
This week’s section handout has practice with Hashing as well as some final review.

1. Hash Codes for Strings

If our hash table has 6 buckets, diagram the result of putting the following values into the hash table, using a hash function that adds up the values of each letter in the string (where ‘a’ is 1, ‘b’ is 2, etc.) and mods by the hash table length (6). If two strings collide, put them into a linked list.

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
a b c d e f g h i j k l m n o p q r s t u v w x y z

cabbage, baggage, deadbeef, cafe, badcab, feed

Bonus: diagram the resulting bucket arrangement with 12 buckets.

2. Tracing Trees (BST)

Draw what the resulting binary search tree would be if the elements were added in the following order.

a. 1, 2, 3, 4, 5, 6, 7, 8
b. 8, 6, 4, 2, 1, 3, 5, 7
c. 5, 7, 3, 8, 1, 2, 4, 6

3. CountDuplicateStrings (Linked List)

Write a function `countDuplicateStrings` that returns the number of duplicates in a given (sorted) list of strings, case-insensitively. Assume that the list will be in sorted (alphabetical) order, case-insensitively, so all of the duplicates will be grouped together. For example, for the list below, the call should return 7 because there are 2 duplicates of "apple", 1 duplicate of "bat", 1 duplicate of "car", 2 duplicates of "dog" and 1 duplicate of "fox" (duplicates underlined below). Note that we will be using the `ListNodeString` structure.

```
ListNodeString
struct ListNodeString{
    string data;
    ListNode* next;
};
```

Thanks to CS106B and X instructors and TAs for contributing problems on this handout.
4. Popular (Graphs)

Write a function named `popular` that accepts a reference to a `BasicGraph` as a parameter and returns a `Set` of pointers to all vertexes in that graph that are "popular" by the following definition. Suppose that the graph represents users on a social network such as Facebook. A vertex represents a user, and a directed edge from A to B represents the fact that user A "likes" user B, or is "friends" with B. The weight of the edge represents how much A "likes" B.

A user $v$ is "popular" if all of the following conditions are met:

- At least 2 other users "like" $v$.
- More users "like" $v$ than $v$ "likes" other users. (More arrows are coming in than going out.)
- The combined weight of all "likes" toward $v$ is more than the combined outbound weight of all the edges to other users that $v$ "likes". (More total edge weight is coming in than going out.)

5. Big-Oh

```cpp
int countDistinct1(Vector& v) {
    int numDistinct = 0;
    for (int i = 0; i < v.size(); i++)
    {
        bool isDistinct = true;
        for (int j = 0; j < i - 1; j++) {
            if (v[i] == v[j]) {
                isDistinct = false;
            }
        }
        if (isDistinct) {
            numDistinct++;
        }
    }
    return numDistinct;
}
```

```cpp
int countDistinct2(Vector& v) {
    Set seen;
    for (int i = 0; i < v.size(); i++)
    {
        seen.add(v[i]);
    }
    return seen.size();
}
```

```cpp
int countDistinct3(Vector& v) {
    mysterySort(v);
    int numDistinct = 1;
    for (int i = 0; i < v.size() - 1; i++) {
        if (v[i] != v[i + 1]) {
            numDistinct++;
        }
    }
    return numDistinct;
}
```

a) What is the runtime of `countDistinct1`?

b) What is the runtime of `countDistinct2`?

c) What is the runtime of `countDistinct3` if `mysterySort` is insertion sort? What about if `mysterySort` is merge sort?

Thanks to CS106B and X instructors and TAs for contributing problems on this handout.