This homework is out of 10 points and graded for completion. Working together on a homework is okay, copying answers is not. Some homeworks will include challenge extra credit problems that will be worth 2 points each.

1. Insert 10, 15, 8, 2, 23, 13 in that order into a hash table of size 10 using separate chaining and the hash function $h(x) = 3x + 7$.

```
0 1 2 3 4 5 6 7 8 9
8 15 2 23 10
  13
```

Suppose the hash table uses dynamic resizing and is doubled in size when the load factor threshold reaches 0.9. Continue inserting the values 7, 11, 3, 24. (Notice how elements can be moved during the resize)

```
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
11 13 7 8 15 2 23 10 24
  3
```

2. Why can’t a load factor threshold of 1.1 be chosen for a hash table when using linear probing?

Because when the hashtable reaches a load factor of 1, every single slot in the array will be full, and it will not be possible to insert the next element.

3. In terms of $n$, what is the runtime for the following piece of code?

```cpp
std::vector<int> v;
for(int i = 0; i < n; i++)
    v.insert(v.begin(), i);
```

$O(n^2)$
For this piece of code?

```cpp
std::vector<int> v;
for(int i = 0; i < n; i++)
    v.push_back(i);
```

$O(n)$

What if the type of $v$ is changed to be a `std::list`?

$O(n)$ for both

4. As seen in class, an in-order traversal of a binary search tree will list the elements of the tree in sorted order in $O(n)$ time. How quickly can a comparison based binary search tree construction algorithm take?

A comparison based binary search tree construction algorithm must take $\Omega(n \log n)$ steps, because running the construction algorithm followed by an inorder traversal would be a comparison based sorting algorithm and we cannot comparison sort faster than $\Omega(n \log n)$.

5. Given a list of numbers, how can you create a binary rooted tree with that list as its in-order traversal? as its pre-order traversal? as its post-order traversal?

There are several ways to accomplish this. The simplest is to build a binary rooted tree that only uses left or right child pointers. So a tree with the list $[1, 2, 3, 4, \ldots, n]$ as its pre-order traversal would be:

```
1
  2
  3
  4
  ...
  n
```

This tree also has the list as its in-order traversal. To get the post-order traversal we can reverse the numbers and have $n$ at the root going down to 1 at the leaf.