Practice with vector-based algorithms

1. *Linear search* is a fundamental algorithm that searches a data structure in one direction, looking for a specific element, or any element that matches specific criteria. It is normally used with linear data structures like arrays and lists whose elements are in *unsorted* order; if the array or list is *sorted*, normally one can use the binary search algorithm which is much faster.

   Write a function to perform a linear search on a vector of ints to look for a specific item. This function should return the index of the item if found, and -1 if not found.

   ```
   int linear_search(vector<int> vec, int lookfor)
   ```

2. *Filtering* is the idea of taking a data structure and creating a copy of it while retaining only elements that fit a certain criteria. For example, we may want to take a vector of integers and create a new vector that contains only the positive integers from the original vector. Sometimes filtering is done on the original structure (modifying it) instead of making a copy; this is called filtering *in place*.

   Write a function to filter a vector of integers, creating a copy that retains only the ones that are greater than a given value.

   ```
   vector<int> filter_greater_than(vector<int> vec, int value)
   ```

3. *Transforming* (sometimes called mapping) is the idea of taking a data structure and creating a copy of it while applying a function to all of the elements. This "function" does not have to be an actual named C++ function; it can be a piece of code that you write on the fly. For example, you may need a function that adds one to each element in a vector of integers. Transforming is often combined with filtering to create, for example, a function that takes a vector of integers, adds one to all of the positive integers, and either eliminates the negative ones or leaves them alone. Transforming, like filtering, can be done *in place* (modifying the original vector rather than making a copy first).

   Write a function that takes a vector of ints and returns a new vector (a copy) with all of the integers doubled; i.e., the vector {1, 2, 3} is transformed into the vector {2, 4, 6}.

   ```
   vector<int> double(vector<int> vec)
   ```

4. Finding the largest and/or smallest items in a data structure (commonly referred to as maximum and minimum, or just max and min) are fundamental ideas you will use a lot. Write two functions, called max and min, that search a vector of integers for the largest and smallest values.

   ```
   int max(vector<int> vec)
   int min(vector<int> vec)
   ```

5. **Challenge**: Write a function that takes an integer argument and returns the binary representation of that integer. You can return a vector of ints (each int in the vector represents a single bit, either 0 or 1), or return one int (where all the bits are put into one number; e.g., 5 gets converted to 101).

   `vector<int> decimal_to_binary(int num)` or `int decimal_to_binary(int num)`

   Hint: if you return a vector, you can use logarithms to find the number of bits the number will have when converted to binary. Just `#include <cmath>` and use the `log` function, which computes the natural log (base *e*) of its argument. How can you use this to get the number of bits in the answer?