CS241
Homework 1: Vectors, Lists, Stacks, Queues, Sorting

1. Recall that project 2 involved writing a SuperVector class with lots of extra methods. Suppose we want to add a method to the SuperVector class called `duplicate` that takes an argument called `times`. This function creates a new SuperVector that consists of all the elements of the original SuperVector duplicated in order the number of times requested.

   Example of use:
   ```
   SuperVector<int> svec;
   svec.append(1);
   svec.append(2);
   svec.append(3);
   SuperVector<int> svec2 = svec.duplicate(3);
   // svec2 is now[1 2 3 1 2 3 1 2 3]
   ```

   Write C++ code for this function. Your code should be as efficient as reasonably possible (don't re-allocate or copy the SuperVector more than necessary, don't loop over the SuperVector more than necessary, etc). Do not call any other functions you wrote for the project; write all of your code inside this function. In other words, don't call `insert` or append or anything like that.

   Here's the method signature:

   ```
   SuperVector<T> SuperVector<T>::duplicate(int times)
   ```

   You may assume that `times` is an integer >= 0.

2. Suppose we have a singly-linked list class set up like this:

   ```
   struct node {
     int data;
     node * next;
   };

   class SList {
     private:

     node * head;
   }
   ```

   Write a member function for `SList` called `removeMultiple(int start, int howmany)` that removes "howmany" elements from the list, starting at the "`start`" element (indexed from zero, like arrays/vectors). So if we had a linked list called `lst`, then `lst.removeMultiple(2, 3)` would remove items at positions 2, 3, and 4 from the linked list.

   Your code should be as efficient as possible (within reason), which means only traversing the list once to find the right spot to start removing elements, and not linking any node pointers that will just immediately be unlinked. Do not call any other functions from `removeMultiple` (in other words, don't rely on the existence of a `removeOne()` function or something like that).

   Here's the method signature:

   ```
   void SList::removeMultiple(int start, int howmany)
   ```

You may assume `start` and howmany have legal integer values (no error checking needed).

3. Assume we have analyzed an algorithm and its run time is determined to be:

   $T(n) = n^3 + 10n + 40$

   What is the big-oh time of this algorithm? (This should be easy.) Call this function f(n).

   Now, prove your answer.
   In other words, find a constant c and a number n_0 such that for all numbers $n >= n_0$, $T(n) <= c * f(n)$. Draw a graph showing T(n) and c * f(n) crossing, and label your axes and where n_0 is.

4. Assume we have an recursive algorithm whose running time we've determined to be $T(n) = 2T(n/2) + 1$, with $T(1) = 1$. Determine the non-recursive T(n) for this algorithm (using the iterative method from class), and then determine the big-oh running time. Show your work.

5. Determine the big-oh running time for the following algorithms in terms of n. (No justification needed.)

   a. Matrix addition:

```
for (int i = 0; i < n; i++)
{
  for (int j = 0; j < n; j++)
  {
    c[i][j] = a[i][j] + b[i][j];
  }
}
```

   b. Matrix multiplication:

```
for (int i = 0; i < n; i++)
{
  for (int j = 0; j < n; j++)
  {
    c[i][j] = 0;
    for (int k = 0; k < n; k++)
    {
      c[i][j] += a[i][k] * b[k][j];
    }
  }
}
```

   c.

```
while (n >= 1)
{
  n = n/2;
}
```

d.

```
x = 1;
for (int i = 1; i <= n-1; i++)
{
  for (int j = 1; j <= x; j++)
  {
    cout << j << endl;
  }
  x = x * 2;
}
```

6. Assume we have an empty stack that stores single characters. For the following sequence of stack operations, **list the order that the characters are popped off the stack**. Also, for every operation marked with a star (*), **draw the stack after that operation has completed** (draw the stack vertically, with the top of the stack at the top ☺)

   Push A
   Push B
   Push C
   Push D *
   Pop *
   Push E *
   Pop
   Pop *
   Push F *
   Pop
   Pop
   Push G *
   Pop
   Pop

7. Assume we have an empty queue that stores single characters. Run the same sequence of operations as the previous stack question, but change all the pushes to enqueues and all the pops to dequeues. **Show the queue after each operation marked with a star** (draw the queue horizontally with the front at the left and back at the right).

8. Suppose we have the following array of ten numbers:

   1 0 2 9 3 8 4 7 5 6

   Recall that each of the quadratic sorting algorithms (selection sort, bubble sort, insertion sort) has an inner loop and an outer loop. For each of the algorithms, run the algorithm on the starting array above, and *show the state of the array after each iteration of the outer loop* as the array is being sorted.

   For instance, for selection sort, you would begin with the following:

   Starting array: 1 0 2 9 3 8 4 7 5 6
   After iteration #1: 0 1 2 9 3 8 4 7 5 6
   [more lines here]