3.3*a* What is the state of the stack after the following sequence of pushes and pops:

```
Queue<int> q;
q.push( 3 );
q.push( 5 );
q.push( 2 );
q.push( 15 );
q.push( 42 );
q.pop();
q.pop();
q.push( 14 );
q.push( 7 );
q.pop();
q.push( 9 );
q.pop();
q.pop();
q.push( 51 );
q.pop();
q.pop();
```

3.3*b* A queue is a first-in—first-out data structure. Suppose you have two queues accepting requests for a particular service, but at some point, the server for one queue goes down. You must now merge the two queues. If no other information is available, what might a reasonable strategy be for forming a single queue out of the two queues? What additional information would be necessary to form a single queue in such a way to be most fair to all requests?

3.3*c* A queue is a first-in—first-out data structure. Suppose a queue is accepting requests for a particular service and that the queue is finite in size and cannot be resized. If the queue is full and another requests comes in, is it fairer to replace an item in the queue or to discard the request.

3.3*d* A unix *pipe* is a data structure that allows communication between different executing programs: the output of one program becomes the input of the second. For example, `find /` will list all directories and files in a Unix system which you have access to view (on one system, I can view three-quarter-of-a-million files and directories. (`find .` will list all files and directories in or below the current directory.) The command `more` will display its output in individual pages on the console, requiring the user to press the spacebar before the next page is display. The `|` operator in the shell says: take the output of the left process and send it to the input of the right process:

```
$ find / | more
```

Now, the pipe data structure will temporarily store the output of find in a finitely sized data structure. If the left process tries to put information into the pipe when the pipe is full, the left process is *put to sleep*—that is, it is not allowed to continue executing until the right process takes some data out of the pipe on its side. Likewise, if the right process attempts to take something out of an empty pipe, it is put to sleep until the left process has put more data into the pipe. When the left process is finished, it will send an *end-of-text* character (Ctrl-C), which tells the right process that the left process has finished outputting data. Explain how a pipe is related to a queue data structure.

ECE 250 *Algorithms and Data Structure*
Department of Electrical and Computer Engineering
University of Waterloo
Please send any comments or criticisms to dwharder@alumni.uwaterloo.ca
with the subject ECE 250 Questions 2.4.
Assistances and comments will be acknowledged.

3.3*e* An implementation could potentially use any of the following data structures to implement the Queue ADT. Indicate the best-case run times if we use the listed data structures in as optimal a sense as possible. Choose the implementation that minimizes the run time of as many operations as possible. In the case of arrays, you may have to move entries if the array is not full—an $\Theta(n)$ operation.

	Singly linked list a head pointer	Singly linked list with head and tail pointers	Doubly linked list with head and tail pointers	One-ended array	Two-ended array	Circular array
`void push(…)`						
`Type pop()`						
`Type top()`						

3.3*f* The `%` operator is the modulo operator in C++. The following works well if `n` is a positive integer, but fails if `n` is a strictly negative integer.

```
int array[N];
// ...
array[n % N] = n;
```

How would you fix this so that this statement works regardless of the value of `n`. You can use a conditional statement or the `?:` operator.

3.3*g* Find the size of a queue using an internal array by using only the member variables `ifront`, `iback`, and `array_capacity`.

```
template <datatype Type>
int Queue<Type>::size() const {




}
```

3.2*h* If a queue using an internal circular array is initially empty with capacity 8, what is the number of copies from old arrays to new arrays if $n = 2^m$ objects are pushed onto the queue without any intermediate pops and if the array capacity is doubled each time the array is full? You will have two formulas: one for when $m < 4$ and one for when $m \geq 4$.

$$\begin{cases} & m < 4 \\ & \\ & m \geq 4 \end{cases}$$

ECE 250 *Algorithms and Data Structure*
Department of Electrical and Computer Engineering
University of Waterloo

Please send any comments or criticisms to dwharder@alumni.uwaterloo.ca
with the subject ECE 250 Questions 2.4.
Assistances and comments will be acknowledged.

3.3*i* Suppose we are doing a breadth-first traversal of the tree in Figure 1 using a queue. What is the state of the queue just before "I" is to be popped off the front of the queue.

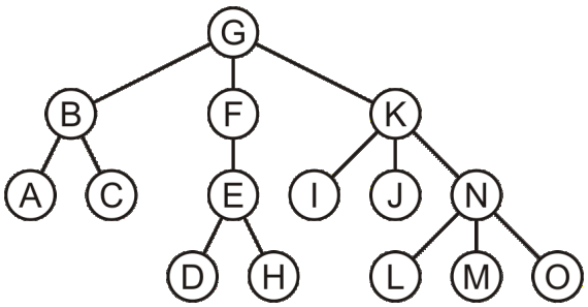

Figure 1. A general tree.

3.3*j* If the tree in Figure 1 is an unordered tree—that is, the order in which the children appear is not relevant—which of the following breadth-first traversals are valid:

G F B K E C A I N J D H M L O

G F B K E C A I N J D H M L O

G K F B I J N E A C O M L H D

G F B K E C A I N J O M L H D