CBMC : Priority Queues

The program I chose to test using CBMC is one that I found online dealing with priority queues.
It can be found by going to the following link:
http://www.sanfoundry.com/c-program-priority-queue/

The code implements a simple priority queue using an array as the data structure.
I thought this would be nice to test using CBMC, also seeing as how priority queues are an important data abstraction in computer science.

At first I had difficulty looking at the code and seeing how it implemented the priority queue, because the code did not follow the semantics of a priority queue (from what I studied).
For instance, the code to delete an element from a priority queue is supposed to just delete (and return) the highest priority element, but this code deletes any element the user specifies, and doesn't return anything.
I don't think this counts as a bug, but doing this static analysis helped me to see inconsistencies in semantics.

Because of above, I decided to leave out the code for deleting an element from the priority queue, and just test the other functionality which is adding an item to the queue.
The elements are added in sorted order from highest to lowest priority (highest priority is at the front/beginning of the queue/array, and the lowest priority is at the rear/end of the queue/array).

As far as specifying the functionality, it was not too difficult other than specifying deleting an element.
I didn't specify that functionality for writing my first test harness (harness.c).
I only had the functionality for adding an element to the priority queue.
The main thing for the harness is to check whether the queue is sorted by priority after each operation.
If that is the case, then the code is correct, so I think.

RESULTS FOR HARNESS

I ran the first harness (which tested adding items to the queue) on CBMC, and verification was successful all the way up to loop bound of 10.
I didn't test larger sizes because it was taking too long.
As far as I can tell the priority queue code to add items to the queue is not buggy.

I will talk about bugs I introduced into the code and how my harness responded, but first a breakdown of some timing data:

Verifying with different loop bounds
Here I just used different queue sizes and ran CBMC on those, as loop bounds are directly related to the size.
Size 1 = --unwind 2, Size 2 = --unwind 3, Size 3 = --unwind 4, etc.

Loop Bound	Run Time
2	0.002 s
3	0.007 s
4	0.057 s
5	0.294 s
6	1.345 s
7	1.966 s
8	4.88 s
9	12.525 s
10	26.009 s

We are only at a loop bound of 10 and already it takes almost a half minute to verify, so that is why it is tough to verify for larger queue sizes.
At a queue size of 9 (loop bound of 10) there are close to 33K variables and 124K clauses, so it is no surprise it takes a while.

Turning on/off Bounds and Pointer checker

Loop Bound	Runtime with Pointer/Bounds checker ON	Runtime with Pointer/Bounds checker OFF (same as above)
2	0.003 s	0.002 s
3	0.009 s	0.007 s
4	0.055 s	0.057 s
5	0.246 s	0.294 s
6	0.849 s	1.345 s
7	2.141 s	1.966 s
8	5.103 s	4.88 s
9	11.731 s	12.525 s
10	22.066 s	26.009 s

From the results of the table above, I see only a benefit to turning on bounds checking and pointer checking, as there is no performance hit from that.
At the very least it doesn't affect the run time much, and in some cases it even makes the run time faster.
I don't know how to explain these results, but that is what I got.

INTRODUCING BUGS

I introduced three bugs in the priority queue code for adding an item, and those 3 files containing each bug respectively are called evil_queue1.c, evil_queue2.c, and evil_queue3.c.
I tried to be as subtle as possible, and guess what, ALL 3 bugs ended up getting through my harness without being detected.

The first bug is omitting incrementing the rear pointer of the queue when an item is added.
The second bug is incrementing the front of the queue instead of the rear when an item is added.
The third bug is traversing the array to put the item and if the priority of the item is less than the item you are examining, then you overwrite that item.

The fact that my harness didn't catch any of these bugs means I need to test more than just whether the queue is sorted.
All 3 bugs above can occur even if the queue is sorted in the end.

I modified my harness so that it checks that the number of elements in the queue is increasing as we add items.
It should always be (rear-front) + 1.
This fixes the pointer issues that are caused by the first 2 bugs.
When I test this new harness (called better_harness.c), CBMC is able to catch the first 2 bugs, but the 3rd bug still gets through.
That is because the 3rd bug doesn't change the pointers but changes the items itself, and my code doesn't check if items are being changed.
The only thing that should change with each add item operation is the size of the list and the new item that is added, all the other items should be the same.
I made a reference copy of the queue and compared it to the actual queue, I used the bubblesort function to sort the reference copy so that both should be the same.
If there are differences then that means there is an error, and the harness will catch it.
This is contingent on the fact that bubblesort is correct, but we verified in class that the code for bubblesort is correct.

I made those changes in a new harness called even_better_harness.c, and CBMC is able to catch the 3rd bug as well.
I think at this point I have a solid harness.

Just another note, I have a file called harness2.c along with a complete implementation of the priority queue (with the delete item functionality, the files are called pri_que_update).
This adds more complexity to the harness, I don't think it is ready to test yet, but I will test it soon.