Pass 1.a
input[8,7,6,9]
output[]
Stack[]

Pass 1.b
input[7,6,9]
output[8]
Stack[]

Pass 1.c
input[6,9]
output[8,7]
Stack[]

Pass 1.d
input[9]
output[8,7]
Stack[6]

Quote: Move an item from input directly to output. Do this if the item just dequeued from the input
is less than than the item now on the front of the input queue.
Move an item from input to the stack. Do this if the item just dequeued from the input is greater
than the item now on the front of the input queue.

Pass 1.e (Where do I place 9?)
input[]
output[8,7]
Stack[6]

My Algorithm: Compare 9 with last item of output, which is 7, 9>7, so move to stack

Pass 1.e
input[]
output[8,7]
Stack[9,6]

Now move all items of stack to output, then Swap Input and Output

input[8,7,9,6]
output[]
Stack[]

Pass 2.a
input[7,9,6]
output[8]
Stack[]

Pass 2.b
input[9,6]
output[8]
Stack[7]

Pass 2.c
input[6]
output[8,9]
Stack[7]

Pass 2.d
input[6]
output[8,9,7]
Stack[]

Pass 2.e
input[]
output[8,9,7,6]
Stack[]

Now the input and stack is empty, but the output is not sorted, so I have to run isSorted() to verify the output queue.
I heard from class that when the stack is empty, the output should be sorted. I guess there is something wrong, need help, thanks.

Subject: Re: Algorithm clarification
Posted by jarobinson3 on Sun, 05 Feb 2017 03:40:08 GMT

I am confused by your example and how you do your queue so here is what I would expect from the first pass

Here is what should happen in the first pass:

    // Initial
    [9,6,7,8] -> []
    Stack: []

    // 8 is out of place, put in stack
    [9,6,7] -> []
    Stack: [8]
// 7 is out of place, put in stack
[9,6] -> []
Stack: [7,8]

// 6 looks like it is in place 6 < 9 and 7 > 6
Stack: [7,8]

// 9 is remaining item in input queue, but 7 <= 9 and 7 >= 6 so put it in output
[9] -> [7,6]
Stack: [8]

// 9 is remaining item in input queue, but 8 <= 9 and 8 >= 7 so put it in output
[9] -> [8,7,6]
Stack: []

// 9 is remaining and stack is empty, put 9 on output queue
[] -> [9,8,7,6]
Stack: []

Subject: Re: Algorithm clarification
Posted by sbcarp on Sun, 05 Feb 2017 04:29:37 GMT
View Forum Message <> Reply to Message

I thought queue is First In First Out, so in this case:
Quote:[9,6,7] -> [] // 8 is out of place, put in stack
Stack: [8]

I guess 9 goes out first?

Subject: Re: Algorithm clarification
Posted by jarobinson3 on Sun, 05 Feb 2017 21:16:04 GMT
View Forum Message <> Reply to Message

A queue is first in first out. Convention (as far as I have seen in multiple places) is that you read left to right and therefore you enqueue onto the left and dequeue from the right.

Subject: Re: Algorithm clarification
Posted by sbcarp on Sun, 05 Feb 2017 21:25:57 GMT
View Forum Message <> Reply to Message

Yes, I got you, the reason I enqueue onto the right because of the test result sent from lusth's test dropbox.
TEST #1
queue *a = newQueue(displayInteger);
enqueue(a,newInteger(0));
enqueue(a,newInteger(1));
enqueue(a,newInteger(2));
enqueue(a,newInteger(3));
enqueue(a,newInteger(4));
enqueue(a,newInteger(5));
enqueue(a,newInteger(6));
enqueue(a,newInteger(7));
enqueue(a,newInteger(8));
enqueue(a,newInteger(9));
displayQueue(stdout,a);

Your output:
[0,1,2,3,4,5,6,7,8,9]
Expected output:
[0,1,2,3,4,5,6,7,8,9]

Output matches exactly!

Subject: Re: Algorithm clarification
Posted by hsknutson on Sun, 05 Feb 2017 21:47:38 GMT
In assign0 our queue enqueues at the end and dequeues from the front, sort of like a line.

Subject: Re: Algorithm clarification
Posted by jarobinson3 on Sun, 05 Feb 2017 22:54:49 GMT
You can enqueue/dequeue anywhere as long as you keep the FIFO property. I just mean when you see someone draw out a queue they will usually enqueue on the left side and dequeue from the right. Just do whatever the example says.

Subject: Re: Algorithm clarification
Posted by sbcarp on Mon, 06 Feb 2017 00:15:01 GMT
I'm not able to do it because the result would be smallest to largest order.
Quote:// Initial
[8,7,6,9] -> []
Stack: []

// 8 is out of place, put in stack
[7,6,9] -> []
Stack: [8]

// 7 is out of place, put in stack
[6,9] -> []
Stack: [7,8]

// 6 looks like it is in place 6 < 9 and 7 > 6
Stack: [7,8]

// 9 is remaining item in input queue, but 7 <= 9 and 7 >= 6 so put it in output
[9] -> [6,7]
Stack: [8]

// 9 is remaining item in input queue, but 8 <= 9 and 8 >= 7 so put it in output
[9] -> [6,7,8]
Stack: []

// 9 is remaining and stack is empty, put 9 on output queue
[] -> [6,7,8,9]
Stack: []

Subject: Re: Algorithm clarification
Posted by hsknutson on Mon, 06 Feb 2017 03:54:51 GMT
View Forum Message <> Reply to Message

Yeah the sorting algorithm also gives me smallest to largest. It's probably because the queues we use are flipped compared to the examples.

---

Subject: Re: Algorithm clarification
Posted by jarobinson3 on Mon, 06 Feb 2017 04:39:28 GMT
View Forum Message <> Reply to Message

Yeah I guess Dr. Lusths output is not what I expected.

---

Subject: Re: Algorithm clarification
Ignore the previous posts to this thread. My convention, and the one you should take, is for the queue:

[a,b,c,d]

the element a is the next item to be dequeued and the element d was the last enqueued.

Also, the sorted order, increasing or decreasing, depends on the comparator you use (it's in the assignment 1 spec). If you are getting the wrong sorted order, flip the logic of your comparator.

---

Just to clarify, if stack is not empty we just swap out and in and run again?

---

Still have the issue after flipping the logic of my comparator,

the result would be

input[]
output[8,9,7,6]
Stack[]

Stack is empty, but output is not sorted.

---

georgecoll wrote on Mon, 06 February 2017 14:42 Just to clarify, if stack is not empty we just swap out and in and run again?

On each pass, the stack should be emptied after the input queue is emptied.
sbcarp wrote on Mon, 06 February 2017 15:07: Still have the issue after flipping the logic of my comparator,

the result would be

input[]
output[8,9,7,6]
Stack[]

Stack is empty, but output is not sorted.

A greater than comparator should return a negative number if the first element is greater than the second.

You should not have to change to logic of your sorting function at all.

---

In the project description the case [2.6 1.5 3.8 1.9] is used as an example. I'm having a hard time understanding the steps it takes to get to the first pass through:

[2.600000,3.800000,1.900000,1.500000].

I'm going to write out the steps for the first pass and if someone could correct where I'm going wrong with my logic would be greatly appreciated

//Initial
in: [2.6, 1.5, 3.8, 1.9]
out: []
stack:[]

//2.6 > 1.5, ordering is ok, enqueue to out
in: [1.5, 3.8, 1.9]
out: [2.6]
stack: []

//1.5 !> 3.8, ordering is wrong, push to stack
in: [3.8, 1.9]
out: [2.6]
stack: [1.5]

//3.8 > 1.8, but 2.6 < 3.8, ordering is wrong, push to stack
in: [1.9]
out: [2.6]
stack: [3.8, 1.5]

//1.9 < 2.6, enqueue to out
in: []
out: [2.6, 1.9]
stack: [3.8, 1.5]

//items on stack, push onto out
in: []
out: [2.6, 1.9, 3.8, 1.5]
stack: []

So I'm ending with [2.6, 1.9, 3.8, 1.5] on the first pass. Where am I wrong with my logic??

---

Subject: Re: Algorithm clarification
Posted by Jbmizzell1 on Tue, 07 Feb 2017 21:38:43 GMT
View Forum Message <> Reply to Message

You're messing up in this step

Quote:
//3.8 > 1.8, but 2.6 < 3.8, ordering is wrong, push to stack
in: [1.9]
out: [2.6]
stack: [3.8, 1.5]

It doesn't matter that 2.6 < 3.8. In order for something to go to from input to output it just has to be greater than the next value in input.

Here's each step for the first pass

Quote:
[2.600000,1.500000,3.800000,1.900000]
//2.6 > 1.5 send output
Input: [1.500000,3.800000,1.900000]
Stack: []
Output: [2.600000]

//1.5 < 3.8 send stack
Input: [3.800000,1.900000]
Stack: [1.500000]
Output: [2.600000]

//3.8 > 1.9 send output
Input: [1.900000]
Stack: [1.500000]
Output: [2.600000,3.800000]

---
//since 1.5 < 1.9 you won't pop but send 1.9 to output
Input: []
Stack: [1.500000]
Output: [2.600000,3.800000,1.900000]

//move the remaining values in the stack to output
[2.600000,3.800000,1.900000,1.500000]