If the input order is 9 6 7 8 then our queue will be [9,6,7,8] but we dequeue from the front, so the first item to dequeue will be 9, contrary to an example I saw earlier on this forum where 8 was dequeued first.

If the input queue was [9,9,6,7] initially, none of the operations are applicable, because the stack is empty, qinput[0] is not less than qinput[1], and qinput[0] is not greater than qinput[1]. qinput[0] == qinput[1] in this case.

If the qinput was [9] initially, then none of the operations apply because the stack is empty and we can't compare qinput[0] with qinput[1] since qinput[1] is null.

What does it mean for stack_top to be between qinput_front and qoutput_front? It could mean any of the following:
* qinput_front < stack_top < qoutput_front
* qinput_front > stack_top > qoutput_front
* qinput_front <= stack_top <= qoutput_front
* qinput_front >= stack_top >= qoutput_front
* qinput_front < stack_top < qoutput_front OR * qinput_front > stack_top > qoutput_front
* qinput_front <= stack_top <= qoutput_front OR * qinput_front >= stack_top >= qoutput_front

When we swap qoutput with qinput before a pass, qinput should always be empty right? That way we can just do like union(qinput, qoutput) which will make qinput current qoutput, and qoutput will become empty.

Also the sorting algorithm seems to be doing smallest to largest for me.

Example input: 9 6 7 8

qinput: [9,6,7,8]
qoutput: []
stack: []
// qinput[0] > qinput[1], so put 9 into stack.

qinput: [6,7,8]
qoutput: []
stack: [9]
// qinput[0] < qinput[1], so put 6 into qoutput.

qinput: [7,8]
qoutput: [6]
stack: [9]
// qinput[0] < qinput[1], so put 7 into qoutput.

qinput: [8]
qoutput: [6,7]
stack: [9]
// stack_top not between qinput[0] and qoutput_end. only 1 item in qinput, so put it in qoutput.

qinput: []
qoutput: [6,7,8]
stack: [9]
Pass 1
//empty stack into qoutoput, swap qoutput and qinput

qinput: [6,7,8,9]
qoutput: []
stack: []
// qinput[0] < qinput[1], so put 6 into qoutput.

qinput: [7,8,9]
qoutput: [6]
stack: []
// qinput[0] < qinput[1], so put 7 into qoutput.

qinput: [8,9]
qoutput: [6,7]
stack: []
// qinput[0] < qinput[1], so put 8 into qoutput.

qinput: [9]
qoutput: [6,7,8]
stack: []
// only 1 item in qinput, so put it in qoutput.

qinput: []
qoutput: [6,7,8,9]
stack: []

Done.

Subject: Re: Sorting algorithm clarifications
Posted by bhpauken on Sun, 05 Feb 2017 20:21:38 GMT
View Forum Message <> Reply to Message

I am also confused on the sorting algorithm. Take for example the integer example on the website.

input: 2 4 5 3 1

Pass 1
input:[2,4,5,3,1]
Stack:
output:

Pass 1.1
input[4,5,3,1]
stack:
output:[2]

Pass 1.2
input[5,3,1]
stack:
output:[2,4]

Pass 1.3
input[3,1]
stack:[5]
output:[2,4]

Pass 1.4
input[1]
stack:[3,5]
output:[2,4]

Pass 1.5
input[1]
stack:[5]
output:[2,4,3]

Pass 1.6
input[]
stack:[5]
output:[2,4,3,1]

Pass 1.7
input[]
stack[]
output:[2,4,3,1,5]

I've gone over the logic a few times. Where am I messing up?

---

Subject: Re: Sorting algorithm clarifications
Posted by sbcarp on Sun, 05 Feb 2017 20:22:09 GMT

Same thing happens to me, it's smallest to largest if I follow the description.
Subject: Re: Sorting algorithm clarifications
Posted by hsknutson on Sun, 05 Feb 2017 20:46:39 GMT

qoutput_front should be qoutput_tail in my examples btw. forgot that the last item added is at the back.

Subject: Re: Sorting algorithm clarifications
Posted by jjlukas on Sun, 05 Feb 2017 22:18:06 GMT

Your pass 1.5 should be [2,4,1] : output since 3 > 1

Subject: Re: Sorting algorithm clarifications
Posted by lusth on Mon, 06 Feb 2017 15:02:06 GMT

If you look at the text immediately after the description of the algorithm, it states the assumption that the comparator is implementing "less than". If your comparator implements "greater than", you should get the opposite ordering.