Algorithm Time Complexity Analysis

Let c = iteration count = 100,000

I have ignored the list/queue initialisation for each of the algorithms because they all share it and it has the same cost for all.

Algorithm A - Priority Queue:

	Complexity:	Times:	Resulting complexity:
initialise queue with n Patients	O(n)	1	O(n)
repeat c times:			
dequeue a Patient from queue	O(log n)	c	O(c log n)
enqueue new Patient on queue	O(log n)	c	O(c log n)

The cost is O(2c log n), 2c is constant so the final time complexity for this algorithm is O(log n)

Algorithm B - ArrayList, head at front:

	Complexity:	Times:	Resulting complexity:
initialise list with n Patients	O(n)	1	O(n)
sort list	O(n log n)	1	O(n log n)
repeat c times:			
remove Patient from list(0)	O(n)	c	O(cn)
add new Patient to list	O(1)	c	O(c)
sort list	O(n)	c	O(cn)

The largest cost in this algorithm excluding the list initialisation is the first sort of the unsorted list. O(n log n), subsequent sorts will be much faster with only one item out of order O(n).

The total cost of the repeated loop is O(2cn) + O(1), the constants can be ignored and we get O(n) which is still smaller than O(n log n) so this is the final time complexity.

Algorithm C - ArrayList, head at end:

	Complexity:	Times:	Resulting complexity:
initialise list with n Patients	O(n)	1	O(n)
sort list in reverse order	O(n log n)	1	O(n log n)

repeat c times:

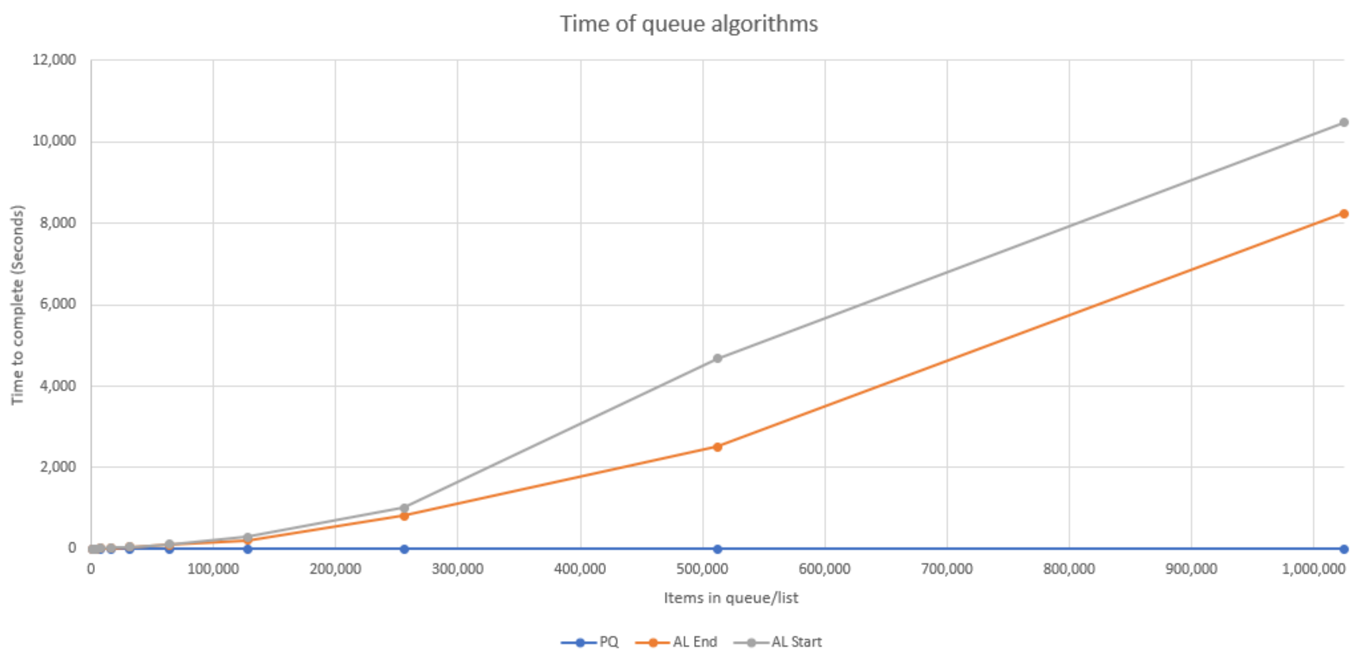
remove Patient from end of list	O(1)	c	O(c)
add new Patient to list	O(1)	c	O(c)
sort list in reverse order	O(n)	c	O(cn)

This algorithm is almost identical to algorithm B except the removal is faster. Since O(n log n) is still has the largest cost, the final time complexity of this algorithm is O(n log n).

Measurement Results

Table of time taken for each algorithm in relation to item count (Times are in seconds)

Item count	PQ	AL End	AL Start
1,000	0.072	1.471	1.468
2,000	0.048	2.471	2.990
4,000	0.052	5.888	6.614
8,000	0.069	13.131	15.898
16,000	0.080	23.406	26.256
32,000	0.075	49.360	54.976
64,000	0.091	98.307	119.529
128,000	0.123	212.704	288.618
256,000	0.187	815.969	1,011.783
512,000	0.339	2,510.243	4,681.318
1,024,000	0.807	8,235.953	10,477.096



Observations and Conclusions

I noticed that even though both ArrayList algorithms have the same cost, the time varies quite a lot. This shows that the cost is better suited for predicting growth for a given algorithm rather than comparing or predicting time values.

The priority queue is by fast the most efficient algorithm. The time hardly changes at all even though the number of items grows very quickly, I think this is due to the partially sorted nature of the queue.