A. Priority Queue

Pseudocode	**Cost**
intialise queue with n Patients	O(n)
repeat 100,000 times:	100,000
dequeue a Patient from queue	O(log n)
enqueue new Patient on queue	O(log n)

Biggest step:
Initialising (O(n))

B. ArrayList, head at front

Pseudocode	**Cost**
intialise list with n Patients	O(n)
repeat 100,000 times:	100,000
remove Patient from list(0) //dequeue	O(n)
add new Patient to list //enqueue	O(1)
sort list	O(n log(n)) the first time O(n) the other times

Biggest step:
Sorting List (O(n log n))

C. ArrayList, head at end

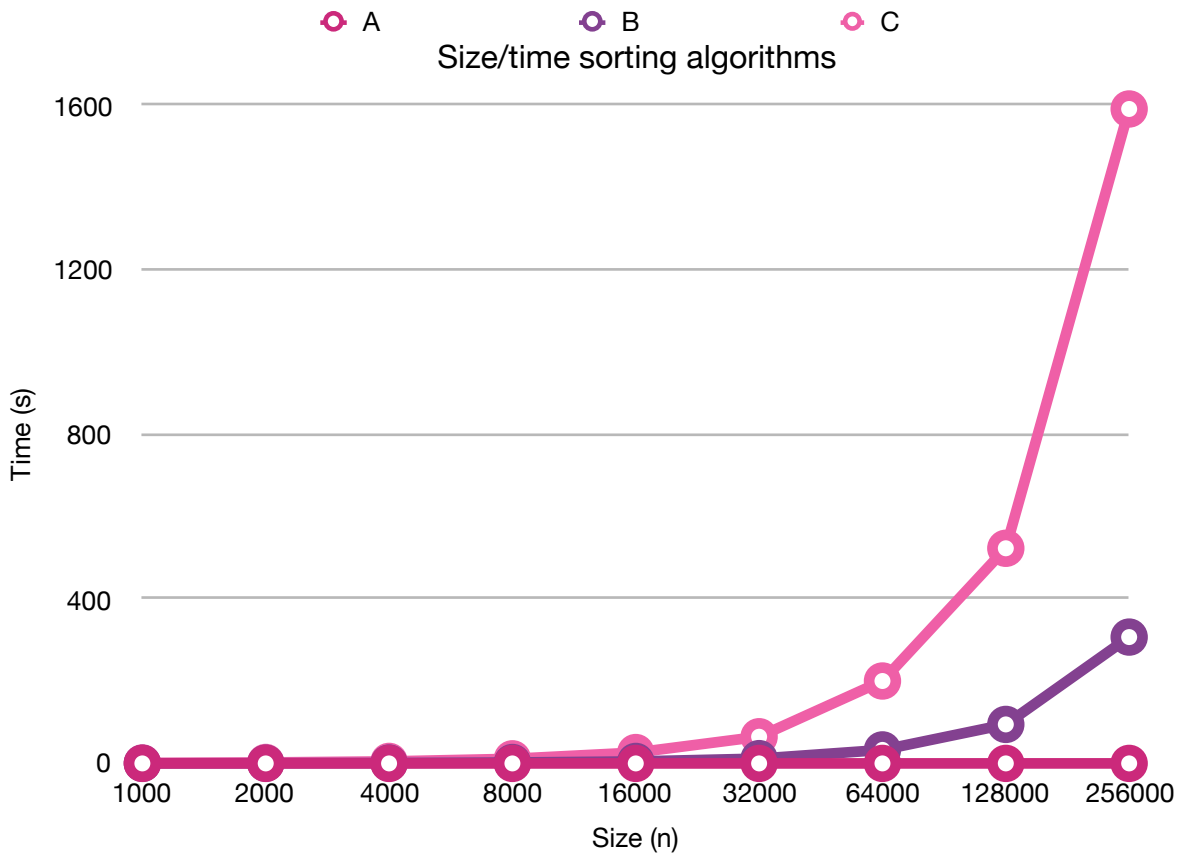
Pseudocode	**Cost**
intialise list with n Patients	O(n)
repeat 100,000 times:	100,000
remove Patient from end of list //dequeue	O(1)
add new Patient to list //enqueue	O(n)
sort list in reverse order (sort + reverse)	O(2n log n)

Biggest step:
Sorting List (O(2n + log n))

Conclusion, summary and findings:

Size	A (seconds)	B (seconds)	C (seconds)
1000	0.039	0.257	1.281
2000	0.020	0.497	2.365
4000	0.016	0.985	4.929
8000	0.015	2.229	11.224

16000	0.015	4.827	26.414
32000	0.018	11.684	64.847
64000	0.035	32.867	200.261
128000	0.062	94.779	522.724
256000	0.067	306.971	1590.057



As it can be seen from the graph and table above, there are no results present for a size over 246000, there are two reasons for this. 1: Algorithm B takes a very large amount of time to complete, taking 30 minutes for 246000 items, and time increasing by almost a factor of 3 every time, this meant that letting the algorithm run would take a very long time, while only continuing to show the trends already present. 2: Due to the large increase in time for B with every step, the increase in time that can be seen with C started to disappear, as it was negligent compared to the increase in B, I therefore decided to only include these points.

From both the table and graph it can be seen that as n grows, the time taken for algorithms B and C to complete the task grows rapidly. By contrast, A stays very consistent, always taking under 1 second to complete the task.

This confirms that there are definitely good ways and bad ways to do the same tasks, where even though it may not seem a lot slower, it can be in some cases, like the one above. It can also be seen that this issue doesn't immediately become clear, as all three algorithms can complete the task small values of n.

In this case, using a priority queue is significantly faster as it automatically sorts items, while this has to be done using a seperate command for the array list.