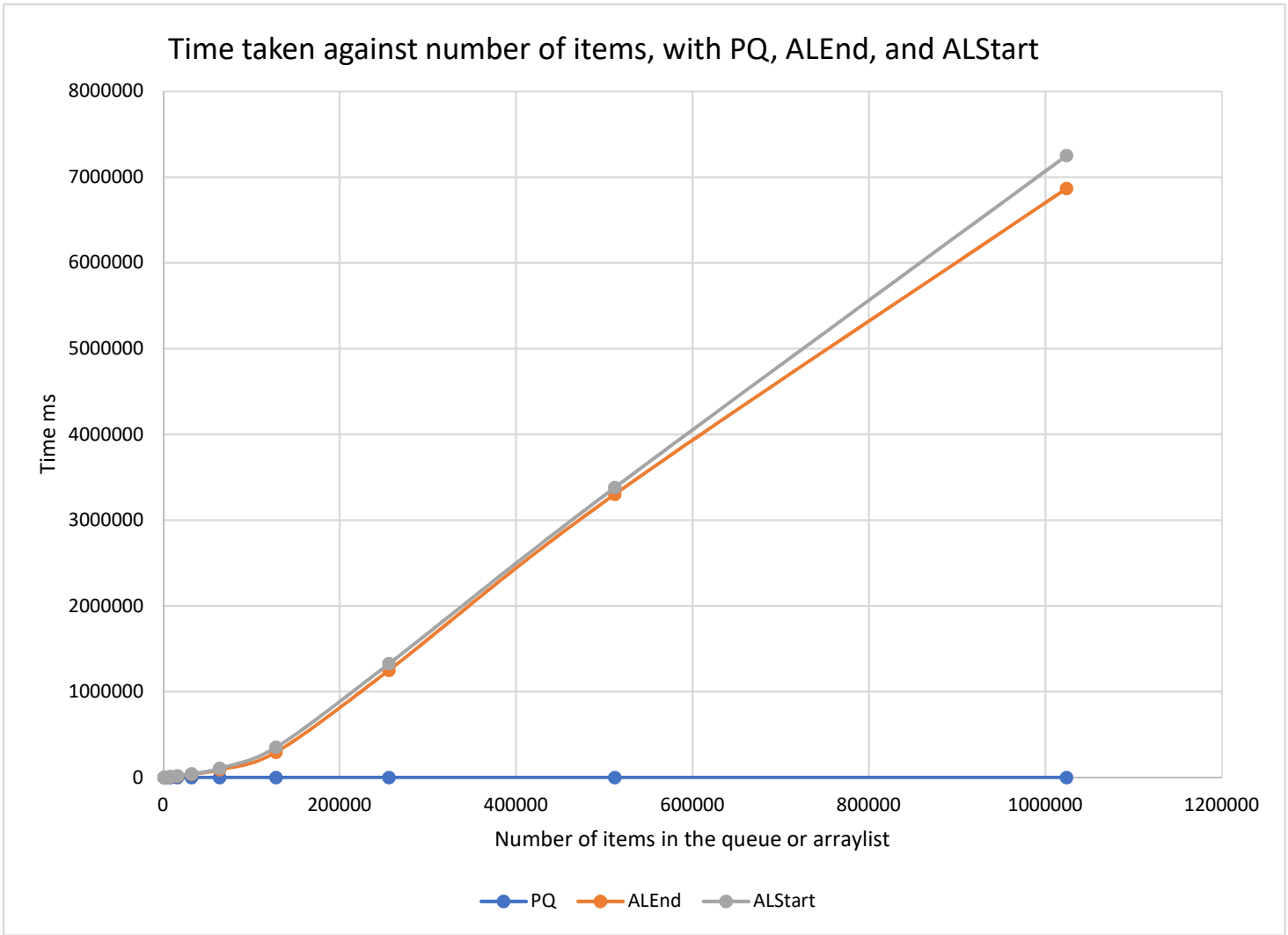
Measuring methods using 100000 repetitions, on queues of size 1000 up to 512,000

n	PQ	ALEnd	ALStart
|1,000|	|21.00|	|547.00|	|564.00|
|2,000|	|19.00|	|1,518.00|	|1,572.00|
|4,000|	|23.00|	|3,541.00|	|3,620.00|
|8,000|	|25.00|	|7,715.00|	|9,358.00|
|16,000|	|32.00|	|19,495.00|	|20,915.00|
|32,000|	|41.00|	|41,988.00|	|44,796.00|
|64,000|	|49.00|	|107,963.00|	|119,621.00|
|128,000|	|98.00|	|310,333.00|	|340,295.00|
|256,000|	|128.00|	|1,074,945.00|	|1,233,757.00|
|512,000|	|126.00|	|3,236,670.00|	|3,279,497.00|



Analysis

Priority Queue:

```
O(n)          Queue<Patient> priorityQueue = new PriorityQueue<>(patients);
O(1)          for (int i = 0; i < TIMES; i++){
O(log(n))             priorityQueue.poll();
O(log(n))             priorityQueue.offer(new Patient());
              }
```

Initialization cost: O(n)
Running cost: O(log(n))

ArrayList Head at the start:

```
O(n)          headAtEnd.addAll(patients);
O(n log(n))   headAtEnd.sort(Collections.reverseOrder);

O(1)          for (int i = 0; i < TIMES ; i++){
O(n)                  headAtEnd.sort(Collections.reverseOrder);
O(1)                  headAtEnd.poll();
O(1)                  headAtEnd.add(new Patient());
              }
```

Initialization cost: O(n log(n))
Running cost: O(n)

ArrayList Head at the start:

```
O(n)            headAtStart.addAll(patients);
O(n log(n))     headAtStart.sort(Patient::compareTo);

O(1)            for (int i = 0; i < TIMES ; i++){
O(n)                  headAtStart.sort(Patient::compareTo);
O(n)                  headAtStart.poll();
O(1)                  headAtStart.add(new Patient());
                }
```

Initialization cost: O(n log(n))
Running Cost: O(n)

Of the three algorithms, the most efficient is to use the priority queue. This is because the cost of initializing a priority queue is O(n), rather than O(n log(n)) for both ArrayLists. Also, the running cost of the priority queue is O(log(n)), rather than O(n). This means that both the initialisation of the priority queue, and the running of the priority queue, are more efficient than the arraylists.

Of the two arraylists, the 'head at start' arraylist is slightly less efficient than the 'head at end', this is because when you remove an element from the start of an arraylist, every other element within the list must then be moved down to fill the gap, giving it a cost of O(n), rather than a cost of O(1) for removing from the end of the list.