For this assignment, I have chosen to use a priority queue which I have implemented using a Linked List. This is because this will sort all of the tickets in order of their priority when they are inserted into the list, so I will not have to run a separate sorting algorithm.

To search through this algorithm, I will have to use a linear or sequential search which has a worst-case time complexity of O(n). I have to use this type of search as each element of the list only points to the next element.

In a normal linked list, the worst-case time complexity for insertion would be O(1) as it would just add the element to the end of the list. However, because I am using this for a priority queue, this increases the time complexity to O(n).

I have chosen to use a Linked list over an array list as the time complexity is quicker for inserting elements. This is because in an array list, you would have to remove each element after the position you wish to insert the new element and reinsert them sequentially, as you cannot nudge every element one space.

Also, you are required to declare the array size on creation. This then allocates a set amount of memory for this array. As the size of the list is constantly changing, and I do not know how many tickets would be in the array at any one time, I would be wasting memory with empty space, or I would have to make a new array with a larger size and copy all of the elements over if more space was needed. While you can create an array list without declaring the size, each time you add a new element the memory size of the list will increase by 50%. You can see this by looking in the java Dev Kit and finding the grow method for ArrayList. This says: newCapacity = oldCapacity + (oldCapacity >> 1). The >> operator means move the bits in the binary representation of the number 1 space to the left, or halves it.

Array lists are much more efficient when accessing random elements as you do not have to search each element sequentially to find the desired one. Because of this, the time complexity is O(1) compared to a time complexity of O(n) for a Linked List. I chose to use a linked list anyway as this is a minor feature of the program, and the linked list is much more efficient for the main functionality of the program (adding and removing elements and saving memory)

I chose not to use a queue of a stack as in order to insert new elements into their correct positions I would need to remove all the elements behind (queue) or in front (stack) of the desired position and reinsert them in the correct order. This would be very inefficient, and the time complexity would increase massively as more elements are inserted.