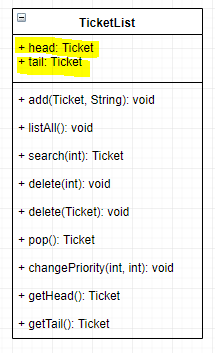
Data Structures and algorithms assignment

# Identified data structure

The data structure that I chose to represent the IT ticketing system was a priority queue with the use of a linked list. Each ticket would hold its own data as well as point to the next ticket in the linked list. A generic queue would take on a first in last out approach (FILO), although with the priority queue this was not necessarily true. For example, for the queue data structure to have a priority, every time a new ticket is added to the list, it must be inserted at the correct place in the queue regarding its priority. Therefore, ticket priority determines when I ticket will leave the list and not when the ticket was inserted into the list.

I felt that this was the best data structure to use because of the requirements of the IT ticketing system; each ticket must be solved in priority and a ticket cannot be solved or worked on if there is a ticket with a higher priority. Using a priority queue met these requirements because items that are ‘popped’ off the list will always have the highest priority, making it efficient.

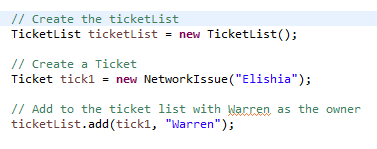


[Figure 1: TicketList Class design]

The performance then for general use of the Ticket List is efficient with regards to popping a ticket off the list. Referring to the TicketList class design in Figure 1, you can see that the Ticket List always has reference to the head of the list, therefore the performance of popping from the list will always be O(1). The most expensive tasks are ones that involve searching for tickets. This action happens every time a ticket needs to be: deleted, ticket priority needs to be changed or the owner of a ticket needs to be changed. It comes with a cost of O(n), with n being how many tickets along the list the searched for ticket is. Hence, this is not the most efficient data structure in terms of searching for items within the linked list.

# Define and implement data structure

The data structure was implemented by using a ticket class as seen Figure 1, with two Ticket pointers, pointing to the previous or null ticket if it is the tail and then to the next or null ticket if it was at the head. Tickets are defined and then added to the TicketList as seen in Figure 2.



[Figure 2: Defining and implementation of a ticket]

When creating a ticket you must pass in the ‘creator’ as a string so the ticket object can refer to its creator, then when you add a ticket to the ticketList you must pass in both the ticket as well as the owner of that ticket. The ticket owner is the person responsible for solving the ticket.