# FIT2004 – ASSIGNMENT 4 – ANALYSIS

*James Schubach – 29743338*

## Task 1

For task 1, what I did was create a min-heap with max-distances as the values, from here the function will then go through the heap either until empty or until the node equals the target. Given the node, it will then go to that index of the graph area and loop through it until we get to the end, while looping it checks to see if the distance of that edge is less then the distance in the min heap, if it is then it will update the node at the destination position while also adding to the path variable. Then it will extract the next min item in the heap and repeat the process.

**Time Complexity**: O(E log V) The logV comes from the decreasing of the heap and at most, the loop will go through E times being the number of edges in the graph. This leaves us with E log V complexity

**Space Complexity**: `O(E + V), The V comes from the path which at most will be the size of the graph. The E comes from building of the heap which is the amount of edges in the graph.`

## Task 2

This task uses the augmented graph to find the path, it does it by adding to the heap if there is no camera at a given vertex, if it is will add and -1 to the heap. From here it checks if the given node is -1, if it is, it will grab the next node. The algorithm works as normal until it reaches the for loop which then checks to see if the edge is a toll road, if it does, skips past this edge and continues to the next.

**Time Complexity**: `O(ElogV)- This is simple as we do not add any extra complexity to the algorithm, it merely adds 2 if statements into the algorithim. Thus the complexity stays the same`

**Space Complexity**: `O(E + V), As stated before we add no complexity to the function, merely 2 if statements thus the space complexity also stays the same`

## Task 3

This function will return the quickest path by visiting at least a single vertex. It does this by using the addService function to calculate the distances from source to detour and then detour to target. It then adds this to a new graph pretending that a given vertex to detour is a single edge and detour to another given vertex. From here the algorithm runs on the new graph and will return a path from source to detour to target. We then run the quickest path algorithm twice to get the actual path, from source to detour, then detour to target. I chose to do it this way, because storing the path in the graph makes it extremely large, which is not needed.

**Time Complexity**: `O(elogv) once again this is very similar to the other algorithms, however we run it three times so we get O(3elogv) which is <= O(elogv)`

**Space Complexity**: `O(E + V)`, This is also similar to the other algorithms, the extra values we have is 2(V + 1) as the path will be at most V, this gives O(E + 2V + 2) which is <= O(E +V)