# Storing the nodes

I have used a vector to store the nodes to improve the performance of the vector the nodes are sorted using the merge sort algorithm by the id this allows the nodes to be quickly found at the start of the A\* pathfinding algorithm. The alternative to using a sorted vector is to use a hash map however lower memory usage was more important and the slowest part of the code is the A\* algorithm. The use of sorting also allows the use of binary search to efficiently find a node by its id in the vector. The O notation of a sorted vector is O(log n) allowing nodes to be found quicker than the standard vector O(n).

# Storing the edges

I have used a vector to store the edges however there is no way of sorting the edges and a hash map is not appropriate because the A\* algorithm requires access to all edges that share a node id preventing single key hash maps from being effectively used. There is also no method of sorting the edges.

# A\* algorithm

## Storing the open set

The open nodes are stored in a heap this allows them to be sorted so that the node with the lowest cost is evaluated first. The heap up and down operations allow the lowest cost node to always be at the top of the heap without affecting performance as much as other sorting algorithms. The heap has O(log n) for adding and removing items.

## Storing the close set

The closed nodes are stored in a vector. The most efficient data structure to store the closed nodes in a hash map however that will require a large amount of memory as the number of closed nodes grows. To improve the memory usage I only store the id of the node. This requires in a O(N) to find a node compared to a hash map O(n)

## Algorithm optimization

The slowest part of the A\* algorithm is the part where it iterates over all the edges that are linked to the current node to improve this I could have added jump point search this allows successor nodes to be found that will find the goal quicker than the lowest node in the open set. The default A\* algorithm has a O(ne) it means maps with many of edges will take longer to calculate.

## Heuristic design choice

I have chosen to use largest difference between the current node and the end node x and y coordinates this allows the algorithm to consider roads that are faster but move away from the end node in a direction. The heuristic also overestimates the distance it has to travel lowering the number of nodes it has to explore.

# Route interface

## Storage of the efficient path

Because the pathfinding algorithm creates a tree leading from the end node to the start I have used a stack to efficiently allow me to print the output in the correct order the stack has O(1) for adding or removing items however getting the size is O(n). This is better than a vector where adding or removing an item is O(n) when the vector needs to grow.

# Additional features

I have added a 3rd mode to the path find system that considered the higher speed roads this could be useful for lorry’s and diesel cars by allowing then to travel on roads that don’t have low bridges. The benefit to diesel cars is it allows the DPF filter to be cleaned quicker. The next feature I have added is the ability to avoid one way roads allowing the same calculation to be used to travel between the 2 points. The next feature I have added is the ability to avoid A roads by removing service roads from the set of edges this allows the path to be found using only B roads and country roads. To help make easier use of the above added features I have added a GUI that will allow the user to select the map they want to load and select the street from the drop down menu.