Question 1

Our A* search implementation revolved around the equation $f = g + h$, where f is the total cost of the node, g is the distance between the current node and the start node, and h is the heuristic, the estimated distance from the current node to the goal node, that we used (as described in our answer to *Q2*). Every node (i.e. hex cell) has an f, g, and h cost associated with them.

The data structures used are as follows:

- *Open*: a list of nodes (hexes), initially empty
- *Closed*: a list of nodes (hexes), initially empty
- *State*: a dictionary (hash table) of the board configuration at any point in time

Below is the pseudocode of our A* search implementation:

function A*SEARCH

Initialise *open* and *closed* as empty lists
Add the starting hex to the *open* list
Loop until *open* list is empty
  Get the hex with the smallest f cost (break ties with smaller h cost)
  Remove that hex from the *open* list
  Add the hex to the *closed* list

  If hex == goal hex
    Return the path traversed from the start hex to the goal hex

  Generate all neighbouring hexes
  Loop through all neighbouring hexes
    If the neighbouring hex is in the *closed* list
      Continue
    If the neighbouring hex is in the *open* list
      If the new g value from the current hex is larger than the neighbouring hex's g value
        Continue
      Else
        Replace the neighbouring hex's f and g costs to the new costs calculated with the current hex
        Set the neighbouring hex's parent to the current hex
    Else
      Replace the neighbouring hex's f and g costs to the new costs calculated with the current hex
      Set the neighbouring hex's parent to the current hex
    Add the neighbouring hex to the *open* list

Time complexity: $O(6^{2(n-1)})$

Space complexity: $O(n^2)$, where n is the size of the board.

Question 2

The heuristic used was cube projection. By calculating the *z* value of each hex, which gives each hex *(x, y, z)*, the horizontal lines would have a constant *y* value while the diagonal lines would have a constant *x* and *z* value. This ensures that each neighbouring hex would have a distance of 1 (would be calculated as √2 with a 2D distance measurement). The distance between 2 hexes is then described as *max(|dx|,|dy|,|dz|)*, which is the maximum difference on any axis. It is admissible as our heuristic is the minimum distance between 2 hexes, assuming that there are no obstacles. Therefore, it will never overestimate the cost. The cost of computing the heuristic is n^2 as the heuristic is calculated for each hex on a board of size *n*.

The overall time complexity of searching is the cost of calculating the heuristic + the cost of A* search itself, which is:

$$O(n^2) + O(6^{2(n-1)})$$

For $n \leq 6$,

$$O(6^{2(n-1)})$$

will dominate.

For $n > 6$, the cost of calculating the heuristic will dominate, which is:

$$O(n^2)$$

Question 3 (Challenge)

Allow the A* function to also take in a colour as an input. Run the A* function normally but do not include the specific colour as an obstacle in finding an optimal solution. When the path is returned, the number of cells that need to be captured can be calculated as:

#hexes captured = #hexes in path - #specific coloured hexes in path

The original heuristic would still be admissible as the heuristic assumed that there were no obstacles. The minimum distance from the current hex to the goal hex would be the same, regardless of colour.