**A) The heuristic evaluation function used for the A\*algorithm.**

When A\* enters into a problem, firstly it calculates the cost to travel to the neighboring nodes and chooses the node with the lowest cost. If The f(n) denotes the cost, A\* chooses the node with the lowest f(n) value. Here ‘n’ denotes the neighboring nodes. The calculation of the value can be done as shown below:

**f(n)=g(n)+h(n)**

**g(n)** = shows the shortest path’s value from the starting node to node n  
**h(n)** = The heuristic approximation of the value of the node

h(n) -cost to go to goal node form current node this will be determined by the distance between the current cell to the goal cell (**√ [ (current. Row – goal. Row)^ 2 + (current. Col – goal. Col )^2 ])**

**Then the node with the shortest distance to the goal will be traversed**

**b) A comparison of the performance of the four search methods in solving problems of differing difficulty.**

1. Before moving on to the next path, a **depth-first search** algorithm exhausts each one. The border is controlled as a stack data structure in these circumstances. “Last-in, first-out” is the key word to remember here. The first node to remove and consider after nodes are introduced to the frontier is the last one to be added. As a result, the search algorithm proceeds as far as it can in the first direction that gets in its way, leaving all other options until later.

**Performance Report**

This algorithm is the fastest at its finest. If it "lucks out" and always chooses the correct path to the solution (by chance), **depth-first search** will take the shortest time possible.

Its worst performance is when there are multiple paths that do not lead to a goal and the algorithm will choose to traverse them first.

1. A **breadth-first search** algorithm will go in several directions at the same time, taking one step in each direction before moving on to the next. The border is maintained as a queue data structure in this scenario. “First-in, first-out” is the key phrase to remember here. In this scenario, all new nodes are added in order, and nodes are prioritized according to which one was added first (first come, first served!). As a consequence, the search algorithm takes one step in each of the potential directions before taking a second step in any of them.

**Performance Report**:

Despite the fact that this method is nearly certain to take longer to execute

than the minimum time, it is sure to find the best solution.

It works **best** when there is a small number of rows.

And its **worst** when every cell has multiple valid adjacent cells or there are too many rows.

1. A **best-first search** extends the node that is closest to the goal, as defined by a heuristic function h(n), which estimates how near the next node is to the goal but can be wrong.

**Performance**:

**Best:** When the path that is closest to the goal's direction achieves it.

**Wors**t: When there is a path that goes to the objective but does not really

achieve the goal.

1. **A\* search** takes into account both **h(n),** the estimated cost from the present position to the objective, and **g(n),** the cost incurred up to the current location. The algorithm can more accurately determine the cost of the solution and optimize its selections on the fly by integrating both of these variables.

It takes less time and space to find the shortest path between the source and destination.

**Performance Report:**

**Best:** When the path that is closest to the goal's direction achieves it.

**Worst**: When there is a road that goes to the objective but does not really achieve there.