# Assignment report

1. **Depth first search –** **Search.py – depthFirstSearch()*
   Data structure used: Stack
   Stack node structure:
   Node[0] = Coordinates
   Node[1] = List containing the path from start node followed to reach the node
   Node[2] = Cost of action

   The worst case for DFS is when the goal is at the last child of start node (At a shallow level). This makes the time complexity an order of B^m where m is the depth.
   The algorithm stores the path from the leaf node to the goal, which is in order of (Bm + 1)

2. **Breadth first search –**
   **Search.py – breadthFirstSearch()*
   Data structure used: Queue
   Queue node structure:
   Node[0] = Coordinates
   Node[1] = List containing the path from start node followed to reach the node
   Node[2] = Cost of action

   Space complexity – Every depth 'd' will have B^d number of nodes generated (B is branching factor, 3 in case of Pacman). So, if goal is at depth 'D', the space and time complexity will be in order of B^D.

3. **UCS first search –**
   **Search.py - uniformCostSearch():*
   Data structure used: Priority Queue
   Priority Queue node structure:
   Node[0] = Coordinates
   Node[1] = List containing the path from start node followed to reach the node
   Node[2] = Cost of action + Null heuristic

   C = Total cost to goal
   E = Smallest cost
   If the path to goal will involve 'n' number of branching, time and space complexity will depend on b^n where b is branching factor.
   n will at the max be C/E as the number of steps taken will be less than or equal to C/E.
   Thus, complexity's upper bound is b^(C/E)

4. **A Star search –**
   **Search.py – aStarSearch()*
   Data structure used: Priority Queue
   Priority Queue node structure:
   Node[0] = Coordinates
   Node[1] = List containing the path from start node followed to reach the node
   Node[2] = Cost of action + Heuristic

   Time complexity of the A star search is dependent on the heuristic function.

5. **Corner search –**
   *SearchAgents.py – class CornersProblem*
   **__init__()*
   not_visited_corners – Contains list of Corners Not visited by Pacman

   **getStartState()*
   Returns start state of problem

   *getSuccessors()
   Return the neighbour nodes for the input node if neighbour isn't a wall and isn't visited already

   *isGoalState()
   Returns true if the pacman is on last corner, and has visited all other corners. If all corners not visited, and current node is a corner, the function removes the corner from not_visited_corners.

   * allVisited()
   Returns true if not_visited_corners is empty i.e all corners have been visited.

   The implemented method has isGoalState() returning False unless the search has covered every corner. This ensures that the pacman follows a path that covers all the corners.

6. **Corner heuristics –**
   The heuristic implemented uses Manhattan distance to calculated the shortest path. The heuristic keeps finding the distance from a state to next nearest corner. The total distance followed in this manner is returned. This heuristic ensures that the distance travelled to cover all the corners is the minimum. This renders it admissible and consistent.

7. **Food search problem –**
   The heuristic implemented uses Manhattan distance from the state to farthest food position. For any path considered, the Pacman will have to travel a minimum path equal to direct distance between the state and the farthest food node. The path might deviate to reach other food nodes, but will never be of a lesser distance. Hence, Manhattan distance from a state to the farthest food node is a consistent and admissible heuristic for the food search problem.

| Algorithm | Maze | Score | Nodes expanded |
|---|---|---|---|
| DFS | Tiny | 500 | 15 |
| DFS | Medium | 380 | 146 |
| DFS | Big | 300 | 390 |
| BFS | Tiny | 502 | 15 |
| BFS | Medium | 442 | 269 |
| BFS | Big | 300 | 620 |
| UCS | Tiny | 502 | 15 |
| UCS | Medium | 442 | 269 |
| UCS | Big | 300 | 620 |
| A Star | Tiny | 502 | 15 |
| A Star | Medium | 442 | 269 |
| A Star | Big | 300 | 620 |

**Memory profiler results:**

BFS > DFS > UCS > AStar

BFS Max 40MB

DFS Max 39MB

UCS 32.3MB

AStar 10.4MB