Question 1

1.1.
An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuator.

Single Agent
A single agent environment is an environment that is explored by a single agent. All actions are performed by a single agent in the environment. An example of this would be playing tennis against a ball where there is only one player.

Mutli-Agent
A multi-agent environment is one where two or more agents are present, perceiving and performing actions in the environment. The environment may be competitive, or cooperative. An example of this would be playing a soccer match

1.2.
Deterministic
These are the environments where the next state of the environment is completely predictable from the current state and the action executed by the agent. There is no uncertainty in the environment.

Stochastic
This environment is the opposite of a deterministic environment. The next state is totally unpredictable for the agent. Uncertainty exists due to randomness, lack of good environment model or lack of complete sensor coverage.

1.3. A game of chess is fully observable. In this environment, the agent sensor is able to sense/access the complete state of and agent at each point in time.

There is no need to keep track of the keep track of the history of the surrounding so the game is not necessarily partially observable.

If the agent has no sensors in all environments, the environment would be unobservable which is not the case in a chess game.

Question 2

2.1.

A well-defined problem can be described by:

A start or initial state	initial state that the agent starts in
Actions	A description of the possible actions available to the agent.
Transition model	This is specified by a function $RESULT(s,a)$. A transition model is a description of what each action does. A successor is any state reachable from a given state by a single action.
Path cost	function that assigns a numeric cost to a path. Cost of a path is the sum of costs of individual actions along the path
Goal test	test to determine if at goal state

2.2.

2.3.

In a search, a graph can be divided into three parts: nodes explored, nodes to be explored next and the remaining unexplored nodes. The explored set keeps track of the nodes that have already been expanded. This aims to prevent the situation where we might get into an infinite loop, expanding the same nodes over and over.

2.4.

Priority Queue

This type of queue is used in a best-first search. A best-first search is a graph traversal algorithm that uses and evaluation function to decide which adjacent node is most promising and then explore.

In this type of search a priority queue is used to store the cost of nodes, with the nodes of the highest priority stored at the front of the queue.

FIFO (First-in-first-out) queue

This type of queue is used in a breadth-first search. A breadth-first search is a graph traversal algorithm that traverses a graph in a breadth-ward (or wide) motion: it explores the closest vertices first and moves outwards away from the source.

In this type of search, it is important to store which vertices have been visited and in what order. To facilitate this process, a FIFO queue is used to insert the nodes that have been visited first and returns the oldest element, based on the order it was inserted.

LIFO (Last-in-first-out) queue/stack
This type of queue is used in a depth-first search. A depth -first search is a graph traversal algorithm that traverses a graph in a depth-ward (or deep) motion. It explores as far as possible along each branch first and then backtracks.

In this type of search, a stack works best as it is LIFO. The search needs to remember where it should go when it reaches a dead end.

2.5.
List and explain the measures used to determine problem solving performance.

A certain list of criteria is used and considered to evaluate an algorithm's performance:

Completeness
A complete algorithm must be capable of systematically exploring every state that is reachable from the initial state. The algorithm should be guaranteed to find a solution when there is one, and to correctly report failure when there is not. A search algorithm must be systematic in the way it explores an infinite state space, making sure it can eventually reach any state that is connected to the initial state.

Cost optimality
A solution should be guaranteed to be optimal. The algorithm should find a solution with the lowest path cost of all existing solutions.

Time Complexity
This attribute also considers the measure of difficulty of the problem. The algorithm should take the least time to find a solution, measured in seconds, or more abstractly by the number of states and actions considered.

Space Complexity
This attribute also considers the measure of difficulty of the problem. The algorithm should utilize the least amount of memory needed to perform the search.

Question 3

3.1.

Objects of the State World:
$H\ H\ H\ C\ C\ C\ B$

3 hikers, 3 children, 1 boat, a left river bank, and a right river bank. C represents a child, H represents a hiker, and B represents the location of the boat.

To describe a state, we need to know how many hikers and children are on either side of the river, and which side of the river the boat is. To achieve this, we keep 3 values per state (h, c, b), where:

Representation of a State of the World:
$L < H\ C\ B > \ R < H\ C\ B >$
A state of the world is represented as 2 lists:
L is the left bank.
R is the right bank.
C represents the location's amount of children $(C \in \{0,1,2,3\})$
H represents the location's amount of hikers $(H \in \{0,1,2,3\})$
B is 1 when the boat is on the shore and 0 when it is on the opposite shore $(B \in \{0,1\})$

3.2.

Initial State
$L < 3\ 3\ 1 > \ R < 0\ 0\ 0 >$

Goal State:
$L < 0\ 0\ 0 > \ R < 3\ 3\ 1 >$

3.3.
1 unit per crossing

3.4.
Action or successor function $S(x)$
The successor function returns feasible states formed:
- adding (or subtracting) 1 to each H and C
- adding (or subtracting) 2 to each C (where $C > 1$)
- toggling b (it is either 1 or 0)

Each state could have at most 8 possible successors (or descendants) since it is not possible to add 2 and to subtract 2 from the same number of hikers or children without going out of range.

Question 4

4.1.

Both tree and graph searches produce a tree. A tree is used to look for a particular node that satisfies conditions mentioned in the problem. It starts at the root and explores nodes from there. They differ in the following ways:

Tree Search
In a tree search, the same node can be visited multiple (or even infinitely many) times, which means that the produced tree (by the tree search) may contain the same node multiple times.

Graph Search
In the case of a graph search, we use a list, called the closed list (also called explored set), to keep track of the nodes that have already been visited and expanded, so that they are not visited and expanded again.

4.2.
Breadth-first search (BFS)
Just like a tree search, BFA aims to traverse a graph/tree by starting at the tree root and exploring neighbour nodes.

BFS differs from a general tree search by exploiting state description to estimate how promising each search node is: it explores all of the neighbour nodes at the present depth prior to moving on to the nodes at the next depth level.

An evaluation function f maps each search node N to positive real number f(N) Traditionally, the smaller f(N), the more promising N.

4.3.

The sliding block puzzle in figure 1 is an example of a n-puzzle. The n represents how many blocks can be moved in the puzzle so the puzzle in figure 1 is a 4x4 15-puzzle.

We would have to map all available tiles to a set $\{1..16\}$.
So then, let the possible number of permutations on the set be Π_{16}.
The set of possible actions A_s is therefore a subset of $\{up, down, left, right\}$

The above is to demonstrate that the 15-puzzle has $16!$ states. But in order to find a solution, there are only $\frac{16!}{2}$ solvable states.

In order to reach the goal state, we would need to find an optimal solution that uses a search tree that uses 17 different boards that are the furthest away from the goal state at a depth of 80 moves from the initial state

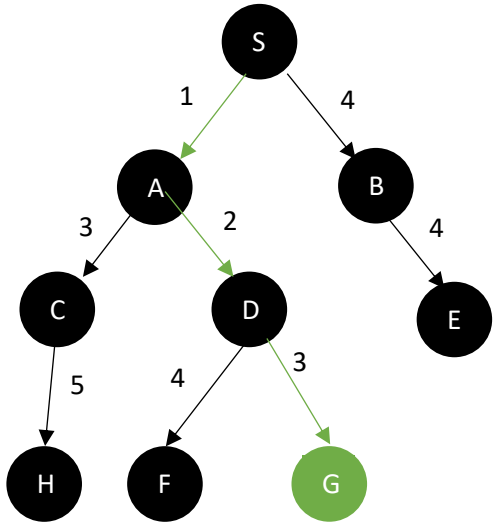
Question 5
5.1.

Level	Nodes Expanded	Nodes goal tested (left to right)
0	None	O
1	O	M E
2	O M E	D C L B
3	O M D C E L	K I J H F G
4	None	

Question 6
6.1.

The Uniform Cost Search (UCS) is an uninformed search algorithm. It used to move around a directed weighted search tree to go from a start node to one of the ending nodes with a minimum cumulative cost. It is used to find the path with the lowest cumulative cost in a weighted graph where nodes are expanded according to their cost of traversal from the root node. This is implemented using a priority queue where lower the cost higher is its priority.

Example: Each edge has a cost. The green edges are the path UCS traverses to get to the goal G



6.2.

Step	Node Expanded	Frontier (priority queue)
1		C($\hat{g} = 0$)
2	C	F-C(4), A-C(20), E-C(24)
3	F-C	H-F-C(13), A-C(20), E-C(24)
4	H-F-C	A-C(20), E-C(24), G-H-F-C(25)
5	A-C	E-C(24), G-H-F-C(25), A-C-B(31)
6	E-C	G-H-F-C(25), A-C-B(31), D-E-C(46)
7	G-H-F-C	A-C-B(31), D-E-C(46)
8	A-C-B	D-E-C(46), D-B-A-C(46)
9	D-E-C	D-B-A-C(46),G-D-E-C(60)
10	D-B-A-C	G-D-E-C(60), G-D-B-A-C(60)
11	G-D-E-C	G-D-B-A-C(60)
12	G-D-B-A-C	

∴ G-H-F-C is the lowest cost path

Question 7
7.1.

Admissible heuristic
This is a pathfinding algorithm. It is admissible because it never overestimates the cost of reaching the goal state.

Consistent heuristic
A heuristic function is said to be consistent if its estimate of the cost to the goal is always less-than-or-equal to the sum of the actual cost to any successor node plus its estimate from that node to the goal.

Question 8

8.1.

Step 0:
Expanded node: none
Goal tested node: A
Explanation: This is the initial state A.

Step 1:
Expanded node: A
Goal tested node: B, C, D
Explanation: From the initial state the algorithm looks at its neighbouring states and goal tests each of them

Step 2:
Expanded node: D
Goal tested node: I,L,K
Explanation: The node D has the lowest function cost between the neighbours of A and was thus expanded.

Step 3:
Expanded node: K
Goal tested node: M,E
Explanation: The node K has the lowest function cost between the neighbours of D and was thus expanded.

Step 4:
Expanded node: M
Goal tested node: N.
Explanation: The node M has the lowest function cost between the neighbours of K and was thus expanded.

Step 5:
Expanded node: N
Goal tested node: none. The goal has been reached.
Explanation: This the goal state of the algorithm and thus the search is terminated

8.2.

8.3.

Step	Node Expanded	Frontier
1		A($\hat{f} = \widehat{10},\ \hat{g} = 0,\ \hat{h} = 10$)
2	A	B-A (19,7,12), C-A (12,8,4), D-A (8,3,5)
3	D-A	B-A (19,7,12), C-A (12,8,4), I-D-A (14,10,4), L-D-A (10,9,1), K-D-A (9,7,2)
4	K-D-A	B-A (19,7,12), C-A (12,8,4), I-D-A (14,10,4), L-D-A (10,9,1), M-K-D-A (10,9,1), E-K-D-A (16,12,4)
5	M-K-D-A	B-A (19,7,12), C-A (12,8,4), I-D-A (14,10,4), L-D-A (10,9,1), E-K-D-A (16,12,4), M-K-D-A-N(10,10,0)
6	M-K-D-A-N	B-A (19,7,12), C-A (12,8,4), I-D-A (14,10,4), L-D-A (10,9,1), E-K-D-A (16,12,4)

8.4.

Question 9

9.1.

Local Maxima

A local maximum is a peak that is higher than each of its neighbouring states but lower than the global maximum. In the case of Hill-climbing algorithms, they local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go.

Local Minima

A local minima, in contrast to a local maxima, is a valley which is lower that its neighbouring states but higher than the global minima.

9.2.

9.3.

To explain the difference between hill-climbing and simulated annealing, we first need to understand what local search algorithms are.

Local search

The are algorithms used to solve optimization problems. They aim to find the best state by utilizing an objective function. Each state in this context has an “elevation”, which is defined by the objective function.

Hill climbing

In the context mentioned above, the aim is to find the highest peak (a global maximum). This process is called hill-climbing. If elevation corresponds to cost, then the aim is to find the lowest valley—a global minimum—and we call it gradient descent.

Hill climbing gradually improves a solution recursively by selecting the best neighbour based on an evaluation function recursively, until there is not a neighbour better than the current. If there is more than one best successor, a random from the set of best successors is selected.

Hill climbing is incomplete because it gets stuck in a local maxima because “downward” moves are not allowed.

Simulated Annealing

Simulated annealing is technique that allows downward steps in order to escape from a local maxima. Annealing emulates the concept in metallurgy; where metals are heated to very high temperature and then gradually cooled so its structure is frozen at a minimum energy configuration.

The idea behind annealing is that, at high temperatures the algorithm should jump out of a local maxima.

9.4.