1.

```
from collections import deque
graph = {}
def add_edge(u, v):
graph.setdefault(u, []).append(v)
graph.setdefault(v, []).append(u)
# Build graph
add_edge('A', 'B')
add_edge('A', 'C')
add_edge('B', 'D')
add_edge('C', 'E')
print("Graph:")
print(graph)
# Recursive DFS
def dfs(visited, graph, root):
if root not in visited:
print(root, end=' ')
visited.add(root)
for neighbour in graph[root]:
dfs(visited, graph, neighbour)
# Recursive BFS
def bfs_recursive(visited, graph, queue):
if not queue:
return
node = queue.popleft()
if node not in visited:
print(node, end=' ')
visited.add(node)
for neighbour in graph[node]:
if neighbour not in visited:
queue.append(neighbour)
bfs_recursive(visited, graph, queue)
# Call DFS
print("\nDFS Traversal:")
visited = set()
dfs(visited, graph, 'A')
# Call BFS (Recursive)
print("\nBFS Traversal:")
visited = set()
queue = deque(['A'])
bfs_recursive(visited, graph, queue)
```

2.

```
import heapq
def heuristic(a, b):
# Manhattan distance
return abs(a[0] - b[0]) + abs(a[1] - b[1])
def a_star(grid, start, goal):
rows, cols = len(grid), len(grid[0])
open_set = []
heapq.heappush(open_set, (0 + heuristic(start, goal), 0, start, [start])) # (f, g, node, path)
visited = set()
while open_set:
f, g, current, path = heapq.heappop(open_set)
if current in visited:
continuevisited.add(current)
if current == goal:
return path
neighbors = [
(current[0] + dx, current[1] + dy)
for dx, dy in [(-1, 0), (1, 0), (0, -1), (0, 1)]
]
for neighbor in neighbors:
x, y = neighbor
if 0 <= x < rows and 0 <= y < cols and grid[x][y] == 0:
if neighbor not in visited:
new_g = g + 1
new_f = new_g + heuristic(neighbor, goal)
heapq.heappush(open_set, (new_f, new_g, neighbor, path + [neighbor]))
return None # No path found
# Example Grid
grid = [
[0, 0, 0, 0, 0],
[1, 1, 0, 1, 0],
[0, 0, 0, 1, 0],
[0, 1, 1, 1, 0],
[0, 0, 0, 0, 0]
]
start = (0, 0)
goal = (4, 4)
path = a_star(grid, start, goal)
if path:
print("Path found:", path)
else:
print("No path found.")
```

3.

```
def selection_sort(arr):
n = len(arr)
for i in range(n):
min_index = i
# Find the minimum element in the remaining unsorted array
```

```
for j in range(i + 1, n):
if arr[j] < arr[min_index]:
min_index = j
# Swap the found minimum with the first unsorted element
arr[i], arr[min_index] = arr[min_index], arr[i]
return arr
# Example usage
arr = [64, 25, 12, 22, 11]
sorted_arr = selection_sort(arr)
print("Sorted array:", sorted_arr)
```

4.

```
def print_board(board):
""" Function to print the board with 'Q' for queens and '.' for empty spaces """
for row in board:
print(" ".join("Q" if col else "." for col in row))
print()
def solve_n_queens(n):
""" Function to solve the N-Queens problem using Backtracking + Branch and Bound """
board = [[0]*n for _ in range(n)] # Initialize the board
col_used = [False] * n
# Columns under attackdiag1 = [False] * (2*n - 1)
# Diagonal (row + col) under attack
diag2 = [False] * (2*n - 1)
# Diagonal (row - col + (n-1)) under attack
def backtrack(row):
""" Helper function to place queens row by row """
if row == n: # All queens placed successfully
print_board(board)
return True # Found a valid solution
for col in range(n):
# Check if placing a queen in this column and diagonals is safe (no conflict)
if not col_used[col] and not diag1[row + col] and not diag2[row - col + n - 1]:
# Place the queen
board[row][col] = 1
col_used[col] = diag1[row + col] = diag2[row - col + n - 1] = True
# Print board after placing the queen
print(f"Placing queen at ({row}, {col}):")
print_board(board)
# Recursively place queen in the next row
if backtrack(row + 1):
return True
# Backtrack: remove queen and reset constraints
board[row][col] = 0
col_used[col] = diag1[row + col] = diag2[row - col + n - 1] = False
print(f"Backtracking from ({row}, {col}):")
print_board(board)
return False # No valid placement found in this row
# Start the backtracking process from the first row
if not backtrack(0):
print("No solution exists.")
else:
print("Solution found!")
# Example usage: Solving the 4-Queens problem
n = 4
solve_n_queens(n)
```

5.

```
# Simple Customer Support Chatbot in Python
def chatbot():
print("Welcome to ShopSmart Support!")
print("Type 'exit' to end the chat.")
while True:
user_input = input("You: ").lower()
if user_input in ["hi", "hello", "hey"]:
print("Bot: Hello! How can I assist you today?")
elif "order" in user_input:
print("Bot: You can check your order status in 'My Orders' section.")
elif "refund" in user_input or "return" in user_input:
print("Bot: To request a refund or return, go to your recent orders and select 'Request
Return'.")
elif "delivery" in user_input or "shipping" in user_input:
print("Bot: Most deliveries are made within 3-5 business days. You can track your order
online.")
elif "account" in user_input:
print("Bot: You can manage your account settings from the Profile section.")
elif user_input == "exit":
print("Bot: Thank you for contacting ShopSmart. Have a great day!")break
else:
print("Bot: I'm sorry, I didn't understand that. Could you please rephrase?")
# Run the chatbot
chatbot()
```

6.

```
def evaluate_performance(attendance, projects_completed, teamwork_score):
if attendance >= 90 and projects_completed >= 5 and teamwork_score >= 8:
return "Excellent"
elif attendance >= 75 and projects_completed >= 3 and teamwork_score >= 6:
return "Good"
elif attendance >= 60 and projects_completed >= 2 and teamwork_score >= 5:
return "Average"
else:
return "Needs Improvement"
```

```
def main():
print("=== Employee Performance Evaluation System ===")
try:
attendance = float(input("Enter attendance percentage (0-100): "))
projects_completed = int(input("Enter number of projects completed: "))
teamwork_score = float(input("Enter teamwork score (1-10): "))
result = evaluate_performance(attendance, projects_completed, teamwork_score)
print(f"\nPerformance Rating: {result}")
except ValueError:
print("Invalid input. Please enter numeric values only.")
if __name__ == "__main__":
main()
```

8.

```
$ curl -o
https://dl.google.com/dl/cloudsdk/channels/rapid/downloads/google-cloud-sdk-380.0.0-linux
x86_64.tar.gz

$ ./google-cloud-sdk/install.sh
$ gcloud init
```

9.

```
public class Email {
public void sendMail(String [] addresses, String [] subjects, String [] messages) {
Messaging.SingleEmailMessage [] emails = new Messaging.SingleEmailMessage[] {};
Integer totalMails= addresses.size();
for(Integer i=0; i<totalMails; i++) {
Messaging.SingleEmailMessage email = new Messaging.SingleEmailMessage();
email.setSubject(subjects[i]);
email.setToAddresses(new List<String> { addresses[i] });
email.setPlainTextBody(messages[i]);
emails.add(email);
Messaging.sendEmail(emails); }
}
```