Approach:

I implemented a Q-learning approach to solve the three problems using sampled transitions from provided datasets.

1. Importing Libraries:
    - numpy (as np) for numerical operations.
    - pandas (as pd) for data manipulation.
    - time for measuring the execution time.
2. Q-learning Functions:
    - initialize_q_table: Initializes a Q-table with zeros.
    - epsilon_greedy_action: Chooses an action based on epsilon-greedy strategy for exploration and exploitation.
    - update_q_table: Updates the Q-table based on the Q-learning update rule.
    - q_learning: Applies Q-learning to learn a policy from the given dataset.
3. Policy Extraction and Saving Functions:
    - get_policy: Extracts the policy from the final Q-table.
    - save_policy: Saves the policy to a file in the required format.
4. Example Usage:
    - Reads three different datasets: 'small.csv', 'medium.csv', and 'large.csv'.
    - Applies Q-learning for each dataset, extracting the policy and saving it to a file.
    - Prints the length of each policy and the time taken for each Q-learning process.
5. Example Parameters:
    - Small Problem: 100 states, 4 actions, discount factor (gamma) of 0.95.
    - Medium Problem: 50,000 states, 7 actions, undiscounted (gamma = 1).
    - Large Problem: 312,020 states, 9 actions, discount factor of 0.95.
6. Timing Execution:
    - Measures and prints the time taken for each Q-learning process.
7. Output Files:
    - Outputs policies to 'small.policy', 'medium.policy', and 'large.policy' respectively.

Results:

Outputting policy of length 100 as small.policy

Time taken: 1.799976110458374 seconds

Outputting policy of length 50000 as medium.policy

Time taken: 3.7789571285247803 seconds

Outputting policy of length 312020 as large.policy

Time taken: 3.716930627822876 seconds

Code:

```
import numpy as np
import pandas as pd
import time

def initialize_q_table(num_states, num_actions):
    return np.zeros((num_states, num_actions))

def epsilon_greedy_action(q_table, state, epsilon):
    if np.random.rand() < epsilon:
        return np.random.randint(q_table.shape[1])  # Explore
    else:
        return np.argmax(q_table[state])  # Exploit

def update_q_table(q_table, state, action, reward, next_state, alpha, gamma):
    current_value = q_table[state-1, action-1]
    max_future_value = np.max(q_table[next_state-1])
    new_value = (1 - alpha) * current_value + alpha * (reward + gamma *
max_future_value)
    q_table[state-1, action-1] = new_value

def q_learning(dataset, num_states, num_actions, alpha, gamma, epsilon,
num_episodes):
    q_table = initialize_q_table(num_states, num_actions)

    for episode in range(num_episodes):
        state = dataset.at[episode, 's']
        action = dataset.at[episode, 'a']
        reward = dataset.at[episode, 'r']
        next_state = dataset.at[episode, 'sp']

        update_q_table(q_table, state, action, reward, next_state, alpha, gamma)
```

```
    return q_table

def get_policy(q_table):
    return np.argmax(q_table, axis=1) + 1  # Adding 1 to convert zero-based
indexing to action space (1, 2, 3, ...)

def save_policy(policy, filename):
    with open(filename, 'w') as file:
        for action in policy:
            file.write(str(action) + '\n')


small_dataset = pd.read_csv('project2/data/small.csv')
medium_dataset = pd.read_csv('project2/data/medium.csv')
large_dataset = pd.read_csv('project2/data/large.csv')


# Small
start_time = time.time()
final_q_table = q_learning(dataset=small_dataset, num_states=100, num_actions=4,
alpha=0.1, gamma=0.95, epsilon=0.1, num_episodes=len(small_dataset))
policy = get_policy(final_q_table)
print("Outputting policy of length " + str(len(policy)) + " as small.policy")
save_policy(policy, 'project2/small.policy')
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")

# Medium
start_time = time.time()
final_q_table = q_learning(dataset=medium_dataset, num_states=50000,
num_actions=7, alpha=0.1, gamma=1, epsilon=0.1, num_episodes=len(medium_dataset))
policy = get_policy(final_q_table)
print("Outputting policy of length " + str(len(policy)) + " as medium.policy")
save_policy(policy, 'project2/medium.policy')
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")

# Large
start_time = time.time()
final_q_table = q_learning(dataset=large_dataset, num_states=312020,
num_actions=9, alpha=0.1, gamma=0.95, epsilon=0.1,
num_episodes=len(large_dataset))
```

```
policy = get_policy(final_q_table)
print("Outputting policy of length " + str(len(policy)) + " as large.policy")
save_policy(policy, 'project2/large.policy')
end_time = time.time()
elapsed_time = end_time - start_time
print(f"Time taken: {elapsed_time} seconds")
```