EEE 448 HW3

Section: 1

Student Name: Utku Kurtulmuş
Student ID: 21903025

Q1)

- **a)** arm 2, since it generates 1 with prob 0.8 > 0.4
- **b)** if we set N = ((T*sqrt(ln(k/δ)))/2k)^(2/3) We get Regret T <= O(T^(2/3)*k^(1/3)*ln(k/δ)^(1/3)) for the explore and commit algorithm. By putting the numbers in:
  N = (T*sqrt(ln(2/0.05)/4)^⅔ = T^(⅔)*(sqrt(ln(2/0.05)/4)^⅔ = 0.6131T^(⅔)

  For T = 500 ⇒ N = 38.62

Q2)

a) N = 16 is the best according to my plot. It is less than the computed N = 38.62. The N calculated in b is only an upper bound for N. It doesn't say that the best N. However, it is a tight upper bound, so we might expect that the best N approximates the upper bound, but in simulation, everything can happen. I'd expect N to be closer to the 38.62.

b) Again, N = 16 is the best according to my plot. However, I would expect this to be closer to the N = 38.62 compared to part A. The reason is the fact that we only consider the samples that are in the Hoeffding bound. In part b of question 1, we derived the inequality for regret_t with probability 1 -δk. In this part, we only considered the samples that satisfy this regret_t bound with the given probability, but the result didn't change.

Q3)

ε = 0.104 is the best according to my plot. It corresponds to the N = 26. This time the results get closer to the calculated upper bound of 38.62.In general, ε-greedy is a better algorithm in terms of optimal regret. Therefore, I wasn't surprised that N got closer to the calculated N.

Code & Plots for Q2 & Q3:

EEE448 HW3

Q2

Part A

Importing the libraries

```
import numpy as np
import matplotlib.pyplot as plt
```

Explore and Commit Algorithm

```
# Function to simulate the Explore & Commit Algorithm
def explore_commit(T, k, delta, N, arm_probs):
    total_rewards = np.zeros(T)
    explore_rewards = np.zeros((N, k))
    commit_rewards = np.zeros(T - k * N)

    # Explore Phase
    for i in range(N):
      for j in range(k):
        explore_rewards[i, j] = np.random.binomial(1, arm_probs[j])

    # Commit Phase
    avg_explore_rewards = np.mean(explore_rewards, axis=0)
    chosen_arm = np.argmax(avg_explore_rewards)

    # Simulate the chosen arm for the remaining time (exploit phase)
    for i in range(T - k * N):
        commit_rewards[i] = np.random.binomial(1,
arm_probs[chosen_arm])

    # Combine exploration and exploitation phases
    total_rewards[:N * k] = explore_rewards.flatten()  # Include all
exploration rewards
    total_rewards[N * k:T] = commit_rewards

    return np.mean(total_rewards)
```

Set parameters

```
# Parameters
T = 500
k = 2  # Number of arms
```

```python
delta = 0.05

# Values of N to be considered
Ns = [1, 6, 11, 16, 21, 26, 31, 41, 46]

# Initialize arrays to store results
avg_rewards = np.zeros(len(Ns))

# True probabilities for each arm (given in Problem 1)
arm_probs = [0.4, 0.8]
```

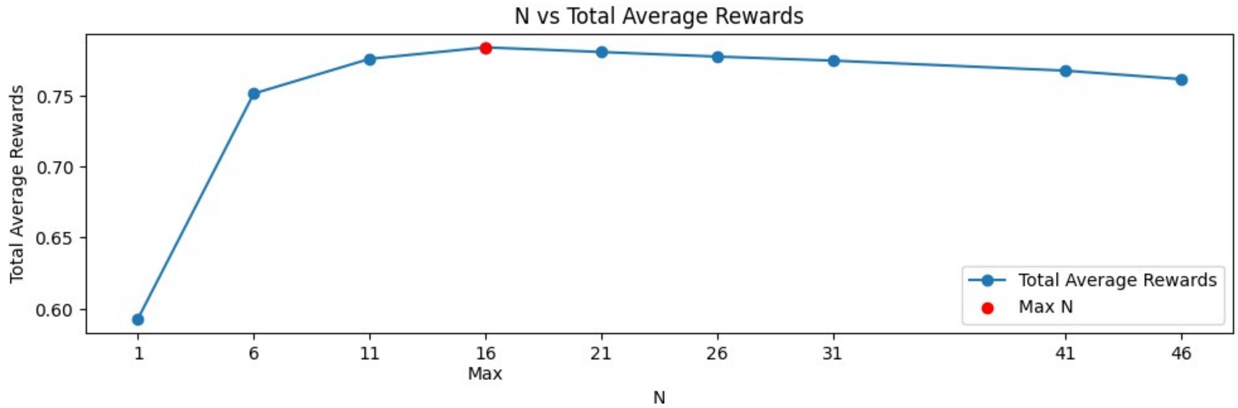
Simulate Explore & Commit

```python
# Simulate the Explore & Commit Algorithm for each N
for i, N in enumerate(Ns):
  for j in range(1000):
      avg_rewards[i] = avg_rewards[i] + (explore_commit(T, k, delta,
N, arm_probs) - avg_rewards[i]) / (j + 1)
```

Plot Explore & Commit Results

```python
# Find the index of the maximum average reward
max_reward_index = np.argmax(avg_rewards)

# Plot N vs the total average rewards
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)

# Plot the entire curve with a lower zorder
plt.plot(Ns, avg_rewards, marker='o', label='Total Average Rewards',
zorder=1)

# Mark the point with maximum Total Average Reward by changing its
color
plt.scatter(Ns[max_reward_index], avg_rewards[max_reward_index],
color='red', label='Max N', zorder=2)

# Set custom x-axis ticks and labels
plt.xticks(Ns)
xticks_labels = [f'{n}\nMax' if n == Ns[max_reward_index] else f'{n}'
for n in Ns]
plt.xticks(Ns, xticks_labels)

plt.title('N vs Total Average Rewards')
plt.xlabel('N')
plt.ylabel('Total Average Rewards')

plt.legend()  # Show legend to identify the markers
```

```
plt.tight_layout()
plt.show()
```



Part B

```
def hoeffding_bound(n, k, T, delta):
    return np.sqrt(np.log(k*T/ delta) / (n))

def explore_commit_hoeffding(T, k, delta, N, arm_probs):
    commit_rewards = np.zeros(T - k * N)
    N_t = [0, 0]
    sample_mean = [0 , 0]

    #make one trial
    N_t[0] += 1
    N_t[1] += 1
    sample_mean[0] = np.random.binomial(1, arm_probs[0])
    sample_mean[1] = np.random.binomial(1, arm_probs[1])

    #explore
    for i in range(1, N):
        for j in range(k):
            sample = np.random.binomial(1, arm_probs[j])
            hoeffding_bound_j = hoeffding_bound(N_t[j], k, T, delta)
            true_mean_diff = np.abs(arm_probs[j] - sample)

            if true_mean_diff <= hoeffding_bound_j:
                N_t[j] += 1
                sample_mean[j] = sample_mean[j] + (((1.0)*sample - 
sample_mean[j])/N_t[j])

    total_explore_rewards = sample_mean[0] * N_t[0] + sample_mean[1] * 
N_t[1]
    chosen_arm = np.argmax(sample_mean)

    for i in range(T - k * N):
```

```
        commit_rewards[i] = np.random.binomial(1,
arm_probs[chosen_arm])

    total_rewards = np.sum(commit_rewards) + total_explore_rewards
    total_average_reward = ((1.0)*total_rewards / (np.sum(N_t) +  T -
k * N))

    return total_average_reward, (np.sum(N_t)/ (k*N)) * 100
```

Set parameters

```
# Parameters
T = 500
k = 2  # Number of arms
delta = 0.05

# Values of N to be considered
Ns = [1, 6, 11, 16, 21, 26, 31, 41, 46]

# Initialize arrays to store results
avg_rewards = np.zeros(len(Ns))
percent_condition_holds = np.zeros(len(Ns))

# True probabilities for each arm (given in Problem 1)
arm_probs = [0.4, 0.8]
```

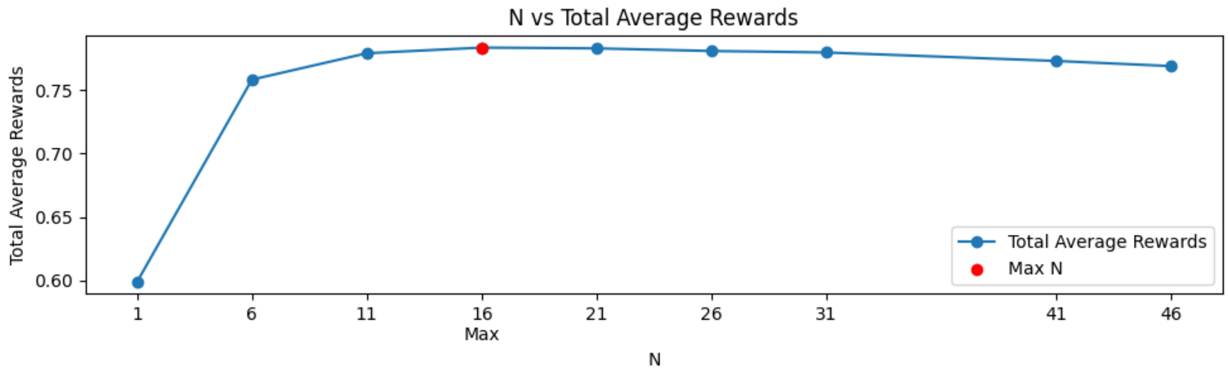
Simulate Explore & Commit Hoeffding

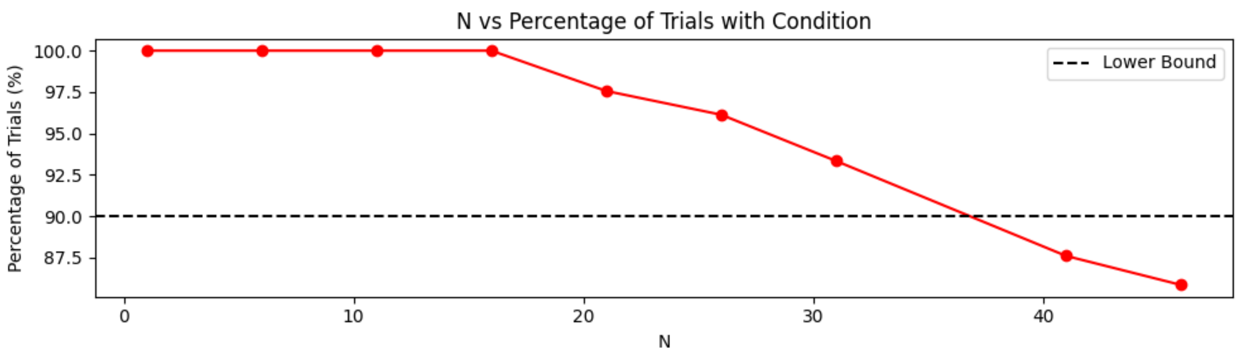
```
# Simulate the Explore & Commit Algorithm for each N
for i, N in enumerate(Ns):
  for j in range(1000):
    avg_rewards[i] = avg_rewards[i] + (explore_commit_hoeffding(T, k,
delta, N, arm_probs)[0] - avg_rewards[i]) / (j + 1)
    percent_condition_holds[i] = percent_condition_holds[i] +
(explore_commit_hoeffding(T, k, delta, N, arm_probs)[1] -
percent_condition_holds[i]) / (j + 1)
```

Plot Explore & Commit Results Hoeffding

```
# Find the index of the maximum average reward
max_reward_index = np.argmax(avg_rewards)

# Plot N vs the total average rewards
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)

# Plot the entire curve with a lower zorder
plt.plot(Ns, avg_rewards, marker='o', label='Total Average Rewards',
zorder=1)
```

```python
# Mark the point with maximum Total Average Reward by changing its color
plt.scatter(Ns[max_reward_index], avg_rewards[max_reward_index], color='red', label='Max N', zorder=2)

# Set custom x-axis ticks and labels
plt.xticks(Ns)
xticks_labels = [f'{n}\nMax' if n == Ns[max_reward_index] else f'{n}' for n in Ns]
plt.xticks(Ns, xticks_labels)

plt.title('N vs Total Average Rewards')
plt.xlabel('N')
plt.ylabel('Total Average Rewards')

plt.legend()  # Show legend to identify the markers

# Plot N vs the percentage of trials where the condition holds
plt.subplot(2, 1, 2)
plt.plot(Ns, percent_condition_holds, marker='o', color='r')
plt.axhline(y=(1 - k * delta) * 100, linestyle='--', color='k', label='Lower Bound')
plt.title('N vs Percentage of Trials with Condition')
plt.xlabel('N')
plt.ylabel('Percentage of Trials (%)')
plt.legend()

plt.tight_layout()
plt.show()
```





Q3

ε-greedy algorithm

```
# Function to simulate the Explore & Commit Algorithm
def greedy(T, k, delta, N, arm_probs, e):
    commit_rewards = np.zeros(T - k * N)
    N_t = [0, 0]
    sample_mean = [0 , 0]

    for i in range(T):
      if np.random.binomial(1, e):
        #Explore
        chosen_arm = np.random.choice(k)
      else:
        #Exploit
        chosen_arm = np.argmax(sample_mean)
      N_t[chosen_arm] += 1
      sample = np.random.binomial(1, arm_probs[chosen_arm])
      sample_mean[chosen_arm] = sample_mean[chosen_arm] +
(((1.0)*sample - sample_mean[chosen_arm])/N_t[chosen_arm])

    total_explore_rewards = sample_mean[0] * N_t[0] + sample_mean[1] *
N_t[1]
    total_average_reward = (1.0 * total_explore_rewards) /
(np.sum(N_t))
```

```
    return total_average_reward
```

Set parameters

```
# Parameters
T = 500
k = 2  # Number of arms
delta = 0.05

# Values of N to be considered
Ns = [1, 6, 11, 16, 21, 26, 31, 41, 46]
# Values of e to be considered
Es = np.zeros(len(Ns))
for i in range(len(Ns)):
  Es[i] = (k * Ns[i]) / T

# Initialize arrays to store results
avg_rewards = np.zeros(len(Ns))

# True probabilities for each arm (given in Problem 1)
arm_probs = [0.4, 0.8]
```

Simulate ε-greedy algorithm

```
# Simulate the Explore & Commit Algorithm for each N
for i in range(len(Ns)):
  for j in range(1000):
    avg_rewards[i] = avg_rewards[i] + (greedy(T, k, delta, Ns[i],
arm_probs, Es[i]) - avg_rewards[i]) / (j + 1)
```

Plot ε-greedy Results

```
# Find the index of the maximum average reward
max_reward_index = np.argmax(avg_rewards)

# Plot ε vs the total average rewards
plt.figure(figsize=(10, 6))
plt.subplot(2, 1, 1)

# Plot the entire curve with a lower zorder
plt.plot(Es, avg_rewards, marker='o', label='Total Average Rewards',
zorder=1)

# Mark the point with maximum Total Average Reward by changing its
color
plt.scatter(Es[max_reward_index], avg_rewards[max_reward_index],
color='red', label='Max Es', zorder=2)

# Set custom x-axis ticks and labels
```

```
plt.xticks(Es)
xticks_labels = [f'{e}\nMax' if e == Es[max_reward_index] else f'{e}'
for e in Es]
plt.xticks(Es, xticks_labels)

plt.title('ε vs Total Average Rewards')
plt.xlabel('ε')
plt.ylabel('Total Average Rewards')

plt.legend()  # Show legend to identify the markers

plt.tight_layout()
plt.show()
```