ASSIGNMENT 1

CS747

SAAVI YADAV
170020003

Implementation Details:-

1) Epsilon Greedy

```
def epsilon(means, ep, hz):
    arms = len(means)
    count = np.zeros((arms,1))
    pulls = np.zeros((arms,hz))
    history = np.zeros((hz,1))
    reward_perarm = np.zeros((arms,1))
    empirical = np.zeros((arms,1))
    for i in range(min(arms,hz)):
        pulls[i,i] = pullArm(means[i])
        history[i,0] = pulls[i,i]
        count[i,0] = count[i,0]+1
        reward_perarm[i] = reward_perarm[i] + pulls[i,i]
        empirical[i] = float(reward_perarm[i])/float(count[i,0])
    for i in range(min(arms,hz),hz):
        k = pullArm(ep)
        if k:
            arm = np.random.randint(arms)
        else:
            arm = np.argmax(empirical)
        pulls[arm,i] = pullArm(means[arm])
        history[i,0] = pulls[arm,i]
        count[arm,0] = count[arm,0]+1
        reward_perarm[arm] = reward_perarm[arm] + pulls[arm,i]
        empirical[arm,0] =float(reward_perarm[arm])/float(count[arm,0])
    actual_max = np.max(means)
    return actual_max*hz-np.sum(history)
```

We first estimate values for empirical means by pulling all arms in a round robin fashion n times where n = min(arms, horizon). Then for the remaining turns, with a probability of epsilon we either explore or exploit the best arm. During exploration, an arm is selected at random and pulled and subsequent reward is added in the history. During exploitation, the arm with the highest empirical mean is selected and the reward is captured.

2) UCB

```
def ucb(means, hz):
    arms = len(means)
    count = np.zeros((arms,1))
    history = np.zeros((hz,1))
    pulls = np.zeros((arms,hz))
    reward_perarm = np.zeros((arms,1))
    ucbt = np.zeros((arms,1))
    for i in range(min(arms,hz)):
        pulls[i,i] = pullArm(means[i])
        history[i,0] = pulls[i,i]
        count[i,0] = count[i,0]+1
        reward_perarm[i] = reward_perarm[i] + pulls[i,i]
    for i in range(min(arms,hz),hz):
        for j in range(arms):
            if count[j,0]>0:
                ucbt[j,0] = reward_perarm[j,0] / count[j,0] +
np.power((2*np.log(i))/count[j,0],0.5)
        arm = np.argmax(ucbt)
        pulls[arm,i] = pullArm(means[arm])
        history[i,0] = pulls[arm,i]
        count[arm,0] = count[arm,0]+1
        reward_perarm[arm] = reward_perarm[arm] + pulls[arm,i]
    actual_max = np.max(means)
    return actual_max*hz-np.sum(history)
```

In the same fashion, for the first n turns, where n = min(arms, horizon) we pull the arms in a round robin fashion and record the rewards obtained. For the remaining turns i.e. (horizon - min(arms, horizon)), For the remaining turns, we define upper confidence bound for an arm as follows:-

$$ucb_a^t = \widehat{p}_a^t + \sqrt{\frac{2\ln t}{u_a^t}}$$

Here p_hat = rewards obtained by arm/ total times the arm is pulled.
Arm with the highest upper confidence bound is selected and subsequently pulled to obtain a reward which is recorded in history and total reward obtained by the arm.

3) KL-UCB

```
def pickArm(reward_perarm,t,c,count):
    ucbkl = []
    for i in range(len(reward_perarm)):
        rhs = np.log(t)+c*np.log(np.log(t))
        rhs = rhs/count[i]
        emp = float(reward_perarm[i])/float(count[i])
        low = emp
        high = 1
        maxIter = 10
        precision = 1e-5
        for j in range(maxIter):
            m = (low + high)/2
            kl = KL(emp,m)
            if (abs(kl - rhs) < precision): break
            if (kl <= rhs): low = m
            else: high = m
        ucbkl.append(m)
    return np.argmax(ucbkl)

def klucb(means,hz):
    arms = len(means)
    count = np.zeros((arms))
    history = np.zeros((hz))
    pulls = np.zeros((arms,hz))
    reward_perarm = np.zeros((arms))
    c = 3
    for i in range(min(arms,hz)):
        pulls[i,i] = pullArm(means[i])
        history[i] = pulls[i,i]
        count[i] = count[i]+1
        reward_perarm[i] = reward_perarm[i] + pulls[i,i]
    for i in range(min(arms,hz),hz):
        arm = pickArm(reward_perarm,i,c,count)
        pulls[arm,i] = pullArm(means[arm])
        history[i] = pulls[arm,i]
        count[arm] = count[arm]+1
        reward_perarm[arm] = reward_perarm[arm] + pulls[arm,i]
    return np.max(means)*hz - np.sum(history)
```

Similarly for the first n turns, the process is similar to UCB. For the remaining turns i.e. (horizon - min(arms, horizon)), we pick the arm which has the highest q value in the bound:-

$$ucb_a^t = max\,(q),\quad q \in [\, \widehat{p_a^t},\ 1\,] \ \ \&$$

$$KL\,(\widehat{p_a^t}, q) \leq \frac{ln\, t + 3\, ln\,(\, ln\, t\,)}{u_a^t}$$

Here p_hat = rewards obtained by arm/ total times the arm is pulled. Binary search is used on the interval [p_hat,1] to obtain q. For convergence, we have fixed the max iterations to 10 and tolerance to 1e-5.

4) Thompson Sampling

```
def thompson(means,hz):
    arms = len(means)
    count = np.zeros((arms,1))
    history = np.zeros((hz,1))
    pulls = np.zeros((arms,hz))
    reward_perarm = np.zeros((arms,1))
    for i in range(hz):
        thoms = np.zeros((arms,1))
        max_value = 0
        best_arm = 0
        for j in range(arms):
            thoms[j] = np.random.beta(reward_perarm[j]+1, count[j]-
reward_perarm[j]+1)
            if thoms[j]>max_value:
                max_value = thoms[j]
                best_arm = j
            arm = best_arm
        pulls[arm,i] = pullArm(means[arm])
        history[i,0] = pulls[arm,i]
        count[arm,0] = count[arm,0]+1
        reward_perarm[arm] = reward_perarm[arm] + pulls[arm,i]
    return np.max(means)*hz-np.sum(history)
```

For Thompson sampling, rewards for all arms are sampled for each time point in horizon from a beta distribution.

$$x_a^t \sim Beta\,(s_a^t + 1, f_a^t + 1)$$

Here $s_a{}^t$ = successes at time t for arm(a) which is equal to total reward obtained by the arm(a) at time(t) and $f_a{}^t$ = failures at time t which is also equal to total times arm(a) is pulled - sum of rewards obtained by arm(a) at a time(t).
Arm with the highest x values is selected and pulled to obtain reward, R.

5) Thompson Sampling with Hint

```
def thompsonWithHint(means,hz,sortedMeans):
    arms = len(sortedMeans)
    hint = sortedMeans[arms-2]
    count = np.zeros((arms,1))
    history = np.zeros((hz,1))
    pulls = np.zeros((arms,hz))
    reward_perarm = np.zeros((arms,1))
    ep = 0.07
    for i in range(hz):
        thoms = np.zeros((arms,1))
        safe = {}
        max_value = 0
        best_arm = 0
        for j in range(arms):
            alpha = reward_perarm[j,0]+1
            beta = count[j,0]-reward_perarm[j,0]+1
            thoms[j] = np.random.beta(alpha,beta)
            betaMeans = (alpha)/(alpha+beta)
            if betaMeans > hint:
                safe[j] = betaMeans
                if safe[j] > max_value:
                    max_value = safe[j]
                    best_arm = j
        if len(safe) != 0:
            arm = best_arm
        else:
            arm = np.argmax(thoms)
        pulls[arm,i] = pullArm(means[arm])
        history[i,0] = pulls[arm,i]
        count[arm,0] = count[arm,0]+1
        reward_perarm[arm] = reward_perarm[arm] + pulls[arm,i]
    return np.max(means)*hz-np.sum(history)
```

For Thompson sampling with hint, the implementation is almost same as Thompson sampling except for a few additions. We are defining a variable hint which is equal to the second highest mean in the true means set. For every time point, a set safe is defined which contains all the arms with beta Means higher than hint.
Now, if the set is empty then we will select an arm using thompson algorithm, otherwise the arm with the highest beta Mean is selected and pulled.

T3:-

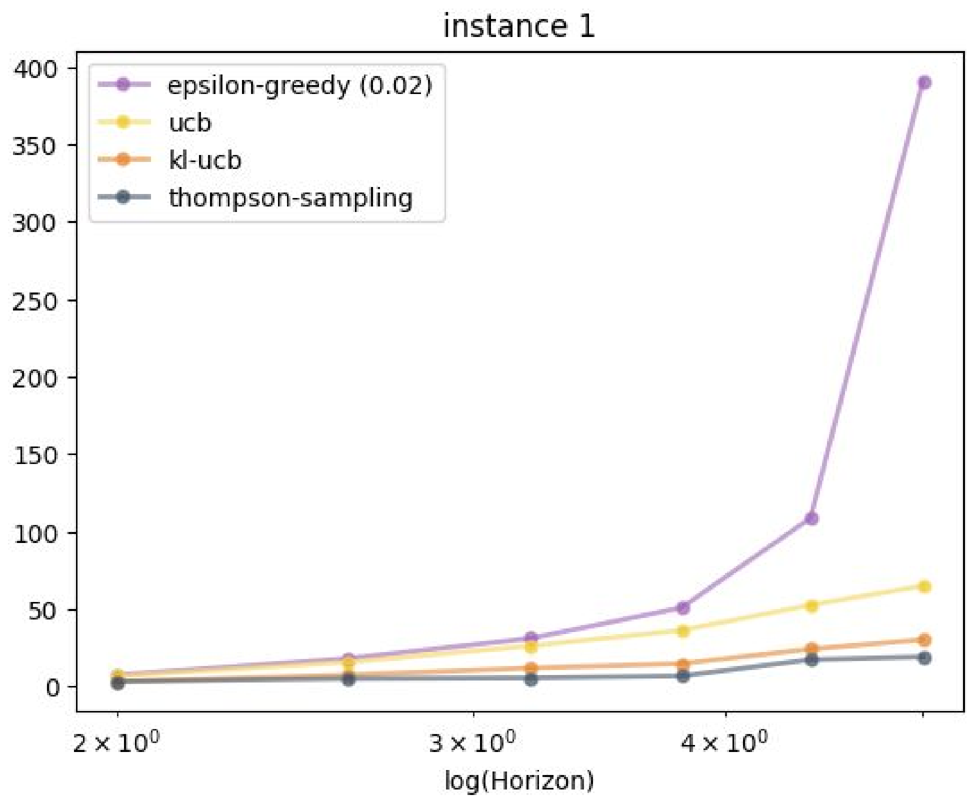
Average regret is calculated for three epsilon values for Epsilon Greedy method with horizon 102400 and seed from [0-49].
For instance 1:-
Avg regret obtained by epsilon 0.002 is the least out of 0.0001 and 0.02.
Regret = [347.44 210.9 390.72]

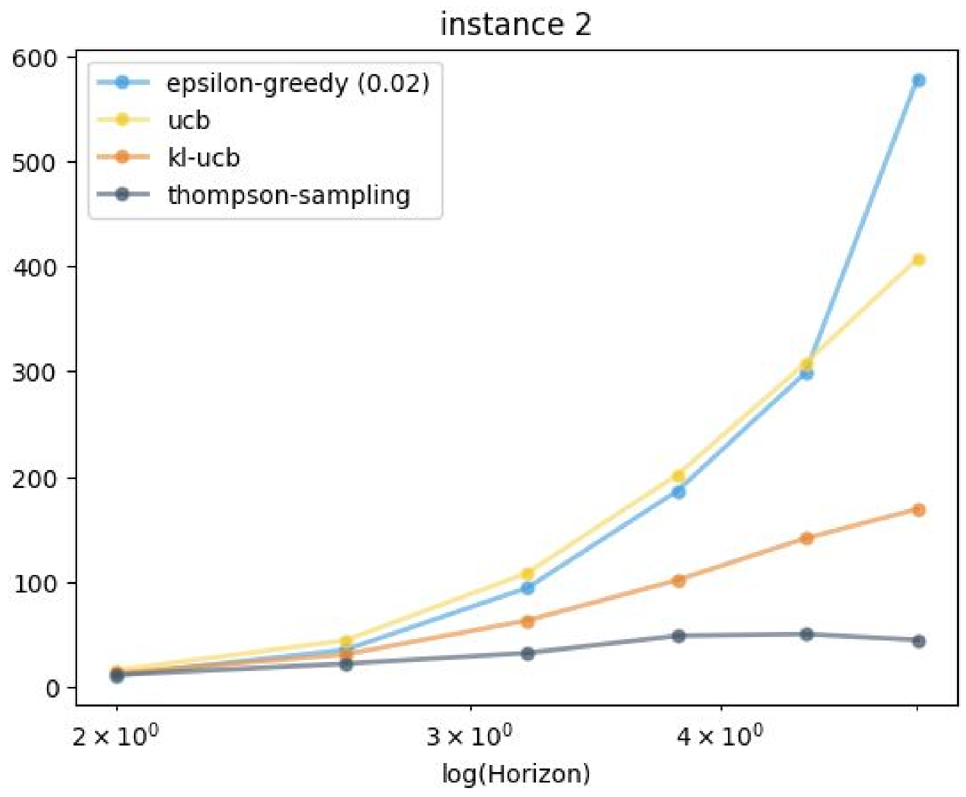
For instance 2:-
Avg regret obtained by epsilon 0.02 is the least out of 0.002 and 0.2.
Regret = [1439.32 578.88 4089.38]

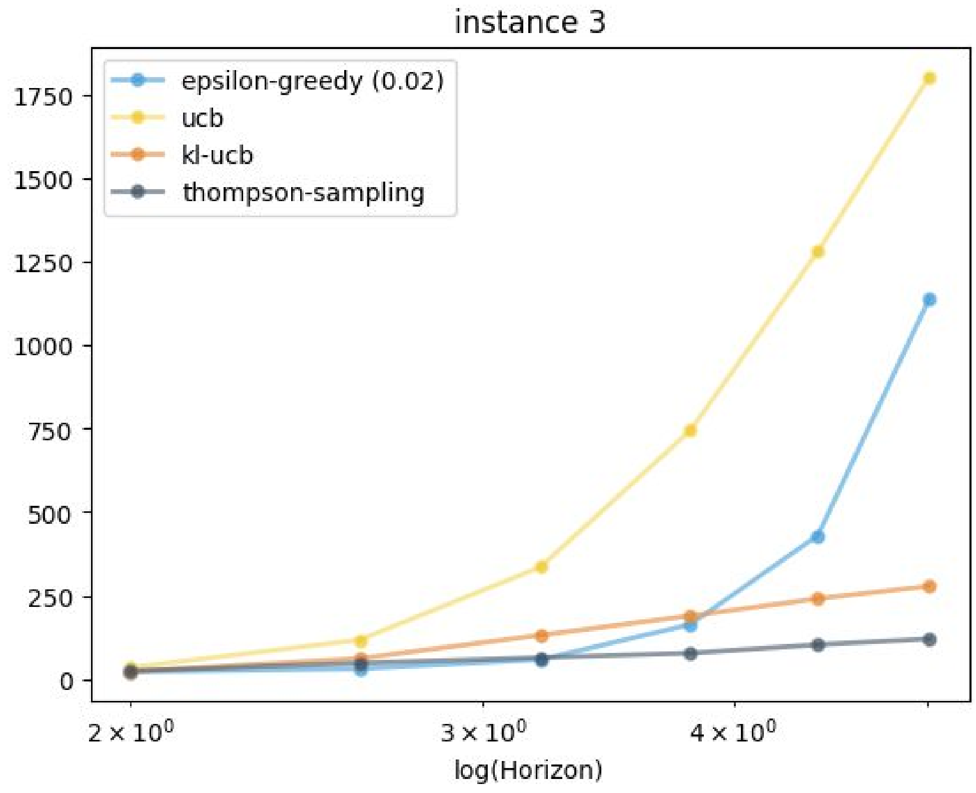
For instance 3:-
Avg regret obtained by epsilon 0.02 is the least out of 0.002 and 0.2.
Regret = [1745.44 1140.7 8523.74]

Graphs:-

T1:-


instance 1
epsilon-greedy (0.02)
ucb
kl-ucb
thompson-sampling
400
350
300
250
200
150
100
50
0
2×10^0
3×10^0
4×10^0
log(Horizon)

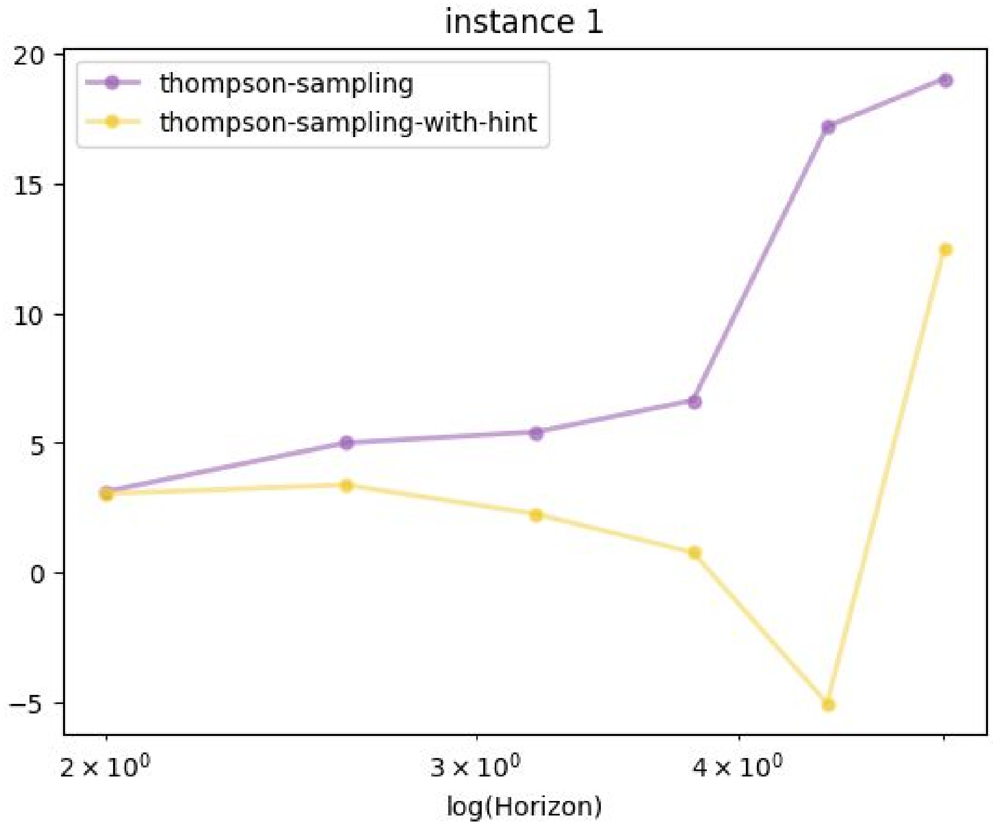

instance 2
epsilon-greedy (0.02)
ucb
kl-ucb
thompson-sampling
600
500
400
300
200
100
0
2×10^0
3×10^0
4×10^0
log(Horizon)

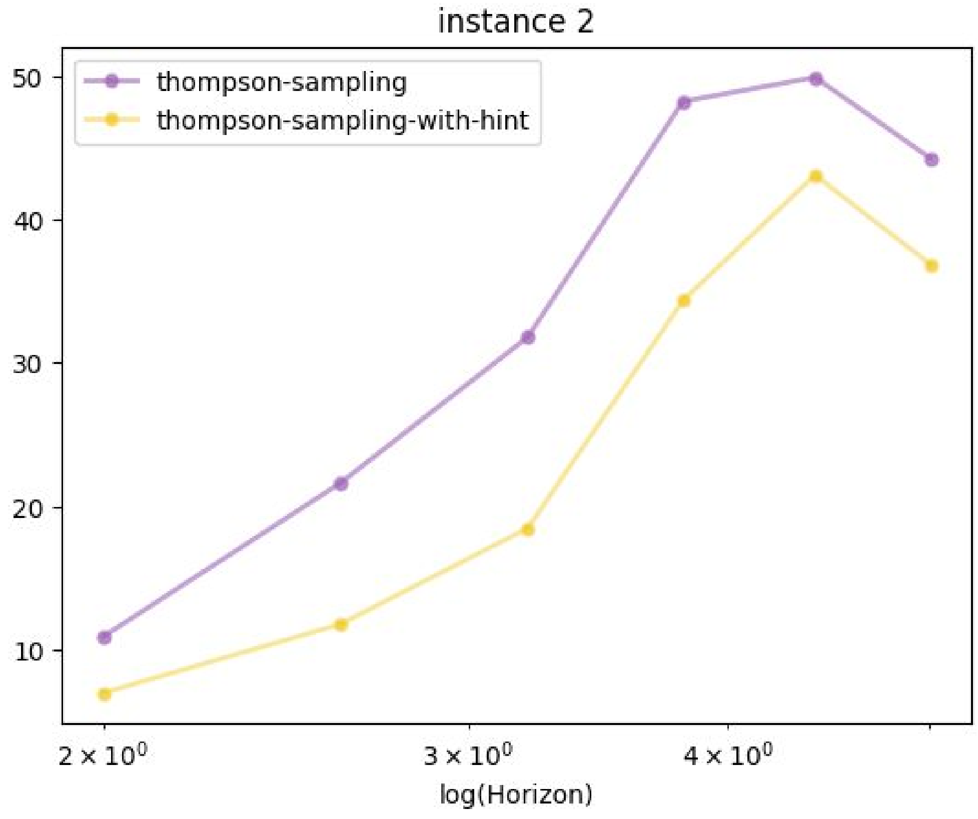


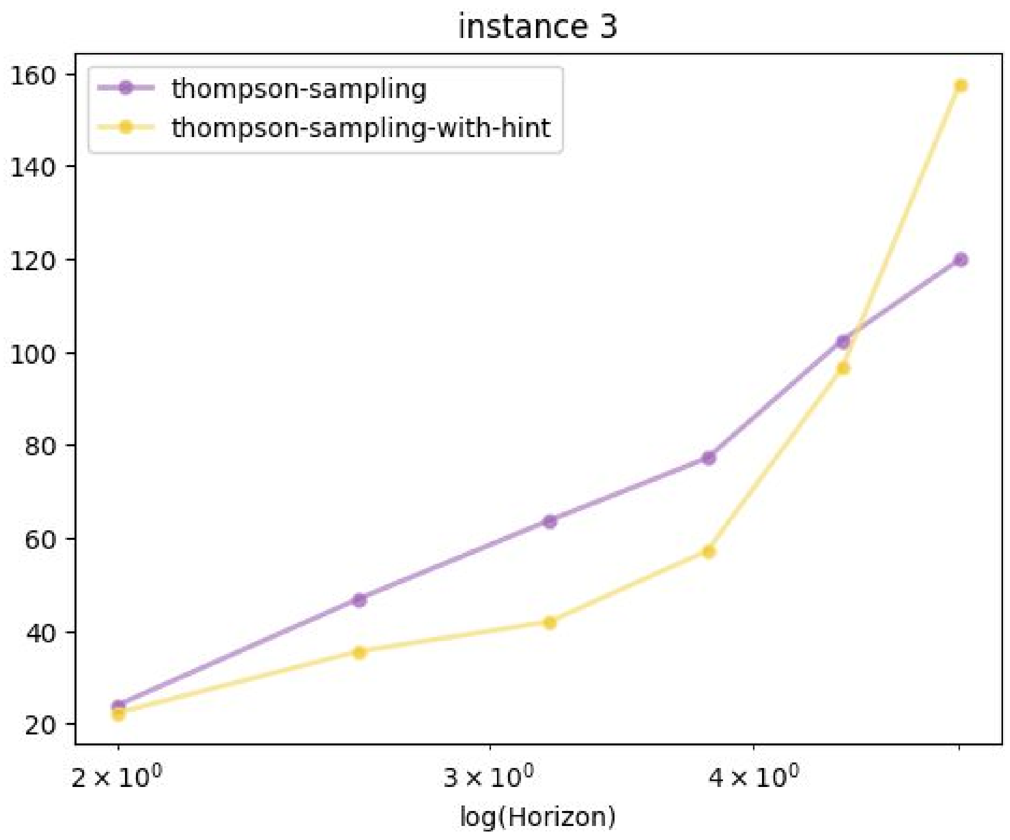
Observations and Inference:-

1) Thompson sampling seems to be doing the best at detecting the best arm and accordingly minimizing the regret even for larger instances and horizons.
2) Epsilon-greedy as expected performs the worst for instance 1 and 2 but for instance 3 we observe that it does better than algorithms with log regret like UCB. It might be due to the fact that as the time horizon increases with a higher number of arms, the term ln(t)/ut dominates over the empirical mean and what we obtain is just a random selection of arms.
3) Kl-Ucb performs better than Ucb for all cases. The regret of kl-Ucb seems to be linear with log(Horizon).
4) For UCB and epsilon-greedy the regret is exponential with log(horizon) which implies linear with horizon as axis.

T2:-


instance 1
thompson-sampling
thompson-sampling-with-hint
20
15
10
5
0
−5
2×10^0
3×10^0
4×10^0
log(Horizon)


instance 2
thompson-sampling
thompson-sampling-with-hint
50
40
30
20
10
2×10^0
3×10^0
4×10^0
log(Horizon)



Observations and Inference:-

1) Thompson sampling with hint does better generally than thompson sampling.
2) For instance 3, thompson and thompson with hint start off with the same regret but as horizon increases, the algorithm thompson sampling with hint might be repeatedly sampling the same wrong arm again and again due to its previous successes. Hence, the arm best suited is not selected as its beta mean is lower and it continues to do so which might lead to shoot up.