CS747 Assignment 1

Yash Salunkhe - 210020156

1 Task 1

My implementations of the algorithms UCB, KL-UCB and Thompson Sampling are in the file `task1.py`.

1.1 UCB

The UCB(Upper Confidence Bounds) algorithm achieves sub-linear regret with the regret being approximately 1400 after 250000 turns. It is performing better than the ϵ-greedy algorithms in tems of regret achieved(ϵ-greedy gives linear regret).

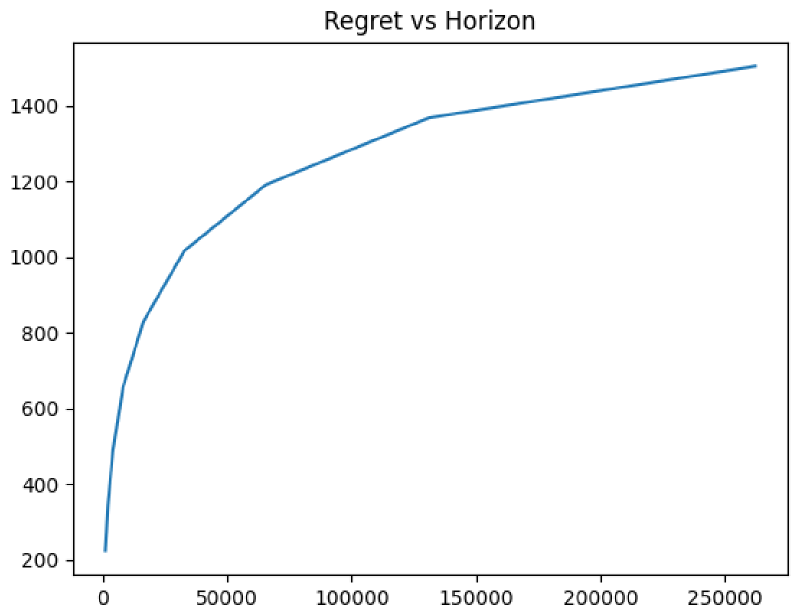

Figure 1: Regret vs Horizon plot for the UCB algorithm

Implementation - Initialized 3 arrays and time variable in the `__init__` function.
`counts[i]` = Number of times the i^{th} arm has been pulled
`values[i]` = Empirical mean of the i^{th} arm
`UCB_bounds[i]` = The UCB value of the i^{th} arm
`time` = Total number of arms pulled
The `give_pull` function returns the index of the arm having the highest UCB bound.
The `get_reward` function updates the counts of the arm pulled, time and UCB bound for every arm. I have added `1e-6` to the counts(in the denominator) in the UCB bound to not encounter any divison by 0 errors.

1.2 KL-UCB

The KL-UCB(Upper Confidence Bounds) algorithm achieves sub-linear regret with the maximum regret being approximately 110 after 250000 turns. It is performing better than the ϵ-greedy algorithms in tems of regret achieved and has a tighter regret bound than the UCB algorithm.

Figure 2: Regret vs Horizon plot for the KL-UCB algorithm

Implementation - Initialized 3 arrays and time variable in the `__init__` function.
`counts[i]` = Number of times the i^{th} arm has been pulled
`values[i]` = Empirical mean of the i^{th} arm
`bounds[i]` = The KL-UCB value of the i^{th} arm($c = 0$)
`time` = Total number of arms pulled
The `give_pull` function returns the index of the arm having the highest KL-UCB bound. The `get_reward` function updates the counts of the arm pulled, time and KL-UCB bound for every arm. A binary search is applied to find the optimal q such that the KL equation holds. For KL-divergence, I have used constants 0.0001 and 1.0000001 to handle edge cases like $x = 0$ and 1.

1.3 Thompson Sampling

The Thompson Sampling algorithm achieves sub-linear regret with the maximum regret being approximately 140 after 250000 turns. It is performing better than the ϵ-greedy algorithms in tems of regret achieved and has a tighter regret bound than the UCB, KL-UCB algorithm.

Figure 3: Regret vs Horizon plot for the Thompson Sampling algorithm

Implementation - Initialized 2 arrays in the `__init__` function.
`alpha[i]` = Number of successes(reward = 1) for the i^{th} arm.
`beta[i]` = Number of failures(reward = 0) for the i^{th} arm.
These have been initialized by 1 so that the sampling from the beta distribution does not give any errors.
The `give_pull` function returns the index of the arm having the highest Thompson Sampling value.
The `get_reward` function updates the alpha(if reward = 1) and beta(if reward = 0) for the arm.

2 Task 2

2.1 Task 2a

The plot obtained for regret vs p2 variation is as shown. The curve attains a maximum at a point between 0.8 and 0.9 and falls to zero at 0.9.

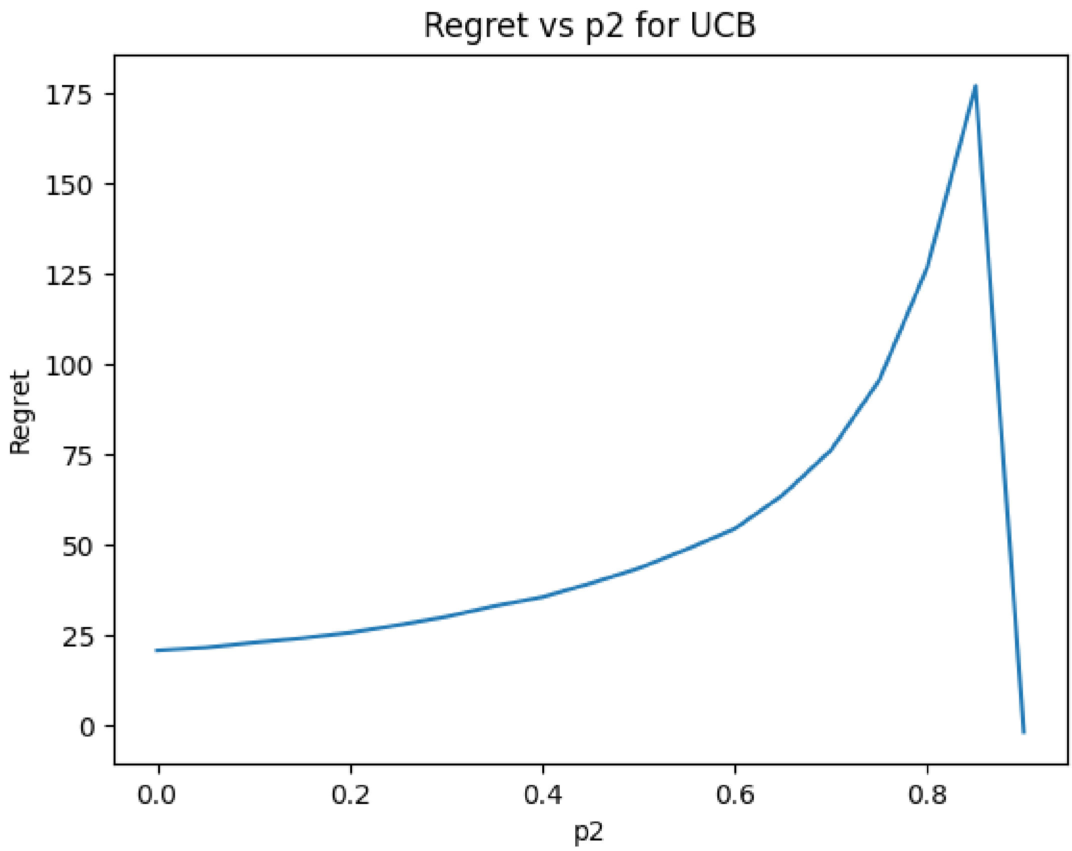

Figure 4: Variation of p2 with regret in UCB(p1 = 0.9)

Reason for this trend - As we increase p2 from 0 to 0.85, the time required for recognizing the optimal arm(the arm with probability p1) will increase. Sampling the arms with probabilities [0.9,0] would take lesser time than [0.9,0.8] to recognize that the first arm is optimal. Due to this, there would be a considerable number of pulls of the second arm which would result in increase in regret. When p2 reaches 0.9, every arm pulled is optimal, hence the expected regret goes to 0.

2.2 Task 2b

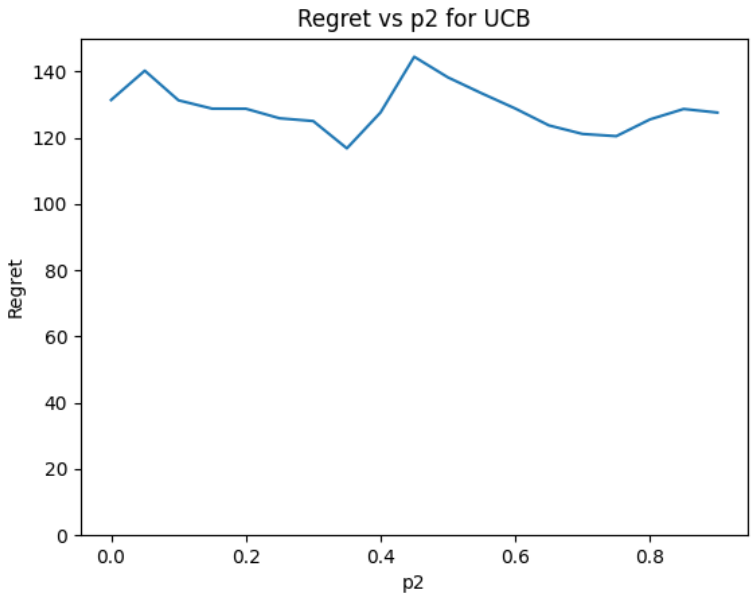

Figure 5: Variation of p2 with regret in UCB(p1 = p2 + 0.1)

Figure 6: Variation of p2 with regret in KL-UCB(p1 = p2 + 0.1)

Reason for trends -

In the UCB algorithm, a big difference is not seen in the regret as we change the value of p2. The theoretical explanation for this is the expected regret for UCB is O($\frac{log(T)}{p*-p}$). Since p* - p is constant(=0.1), the expected regret does not fluctuate much.
For KL-UCB, there is a significant fluctuation in the regret as we tweak the value of p2 signifying that the regret is proportional to c log(T) where c is the constant from Lai and Robbins' bound.

$$\frac{Regret}{log(T)} \leq \sum_{i:p_i(l)\neq p^*(l)} \frac{p*(l) - p(l)}{\mathrm{KL}(p(l), p^*(l))},$$

Here, the regret does not solely depend on p* - p, and hence, it changes significantly with change in p2.

3 Task 3

Here, I used Thompson Sampling in a faulty bandit setting as my main goal is to maximize my reward. I have no control over my faulty bandit settings, I can maximize the output I get out of the non-faulty pulls which is achieved by Thompson Sampling. I preferred Thompson Sampling over UCB, KL-UCB as it has a tighter regret bound as seen in task 1.
The file `task3.py` contains the entire implementation.

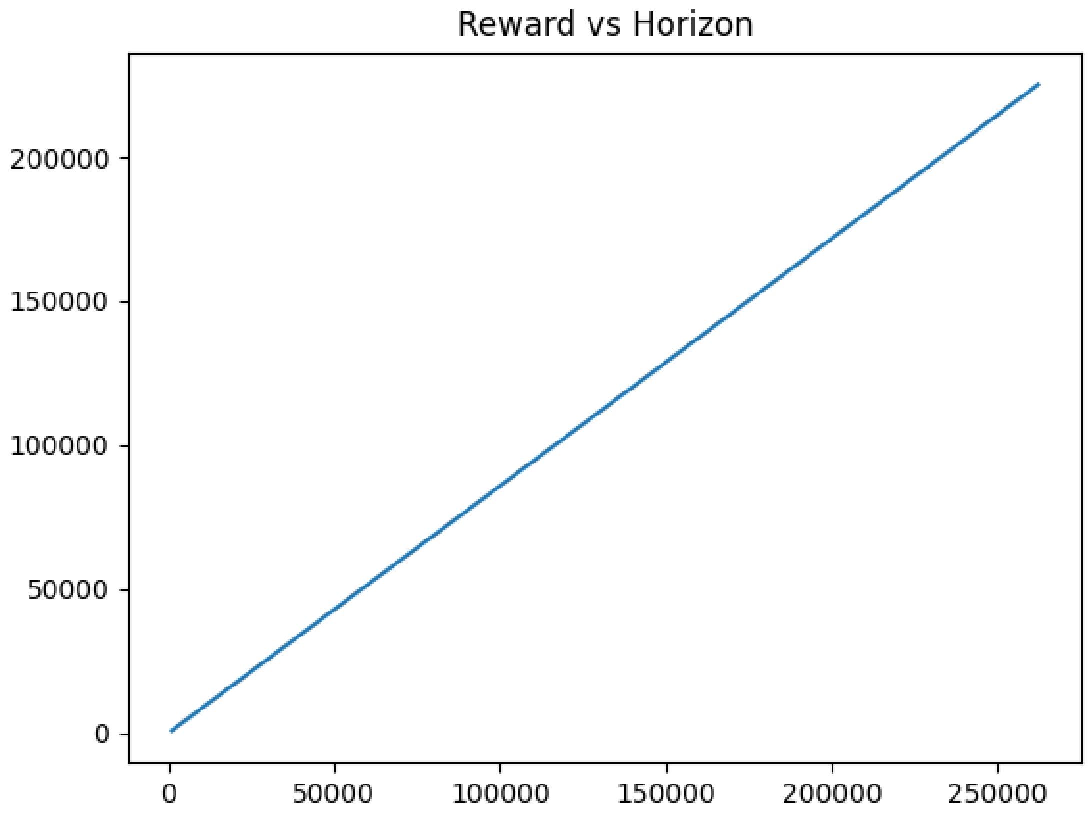

Figure 7: Variation in reward vs horizon for my algorithm

This plot was obtained by adding `plt.plot(horizons,rewards)` to the `task3` function in `simulator.py`

4 Task 4

The file `task4.py` contains the entire implementation. Here, I again used Thompson Sampling in a multi-bandit setting as my main goal is to maximize my reward. I have no control over which bandit is being chosen as it is sampled randomly. Performing Thompson Sampling over the chosen bandit will maximize my reward. I preferred Thompson Sampling over UCB, KL-UCB as it has a tighter regret bound as seen in task 1.

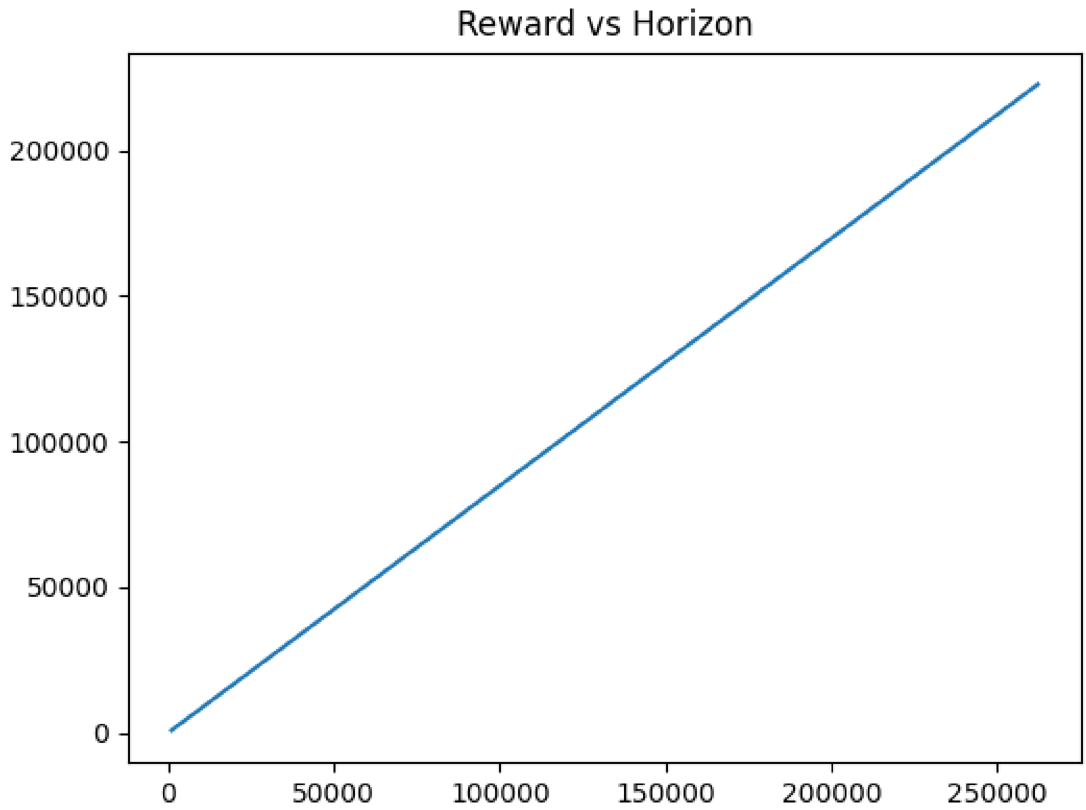

Figure 8: Variation in reward vs horizon for my algorithm

This plot was obtained by adding `plt.plot(horizons,rewards)` to the `task4` function in `simulator.py`