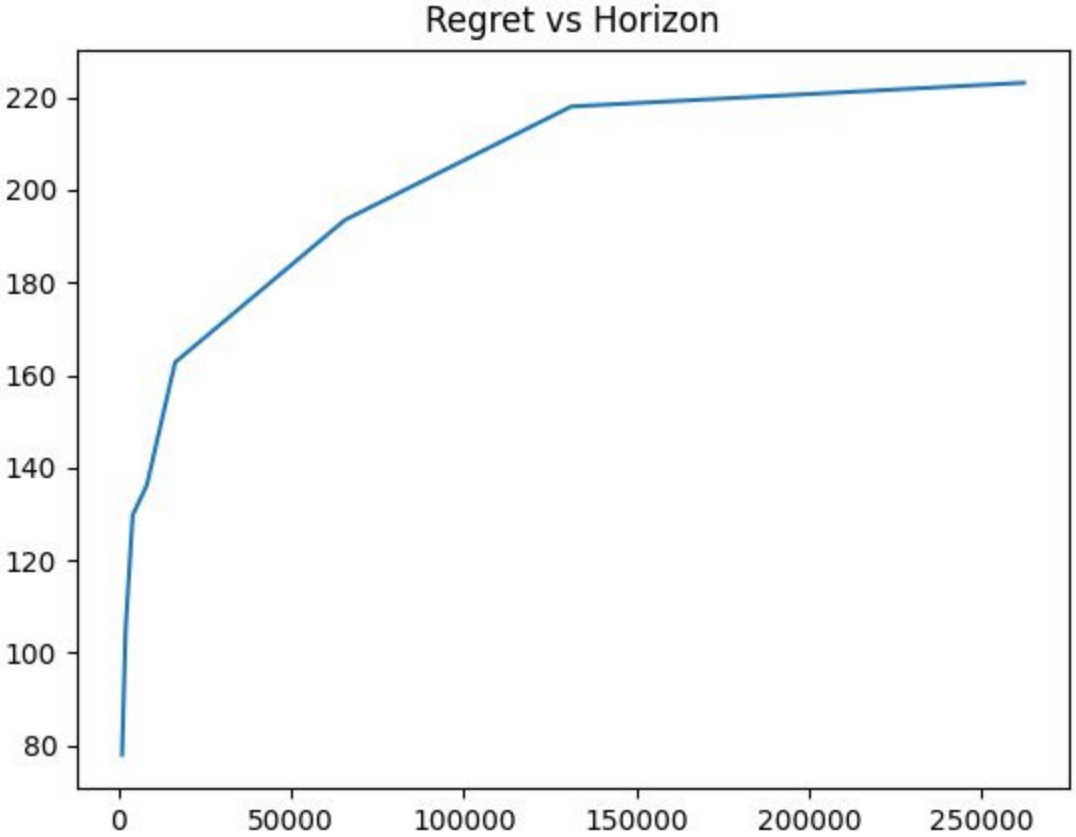
CS747: Assignment_1

Rupansh Parth Kaushik: 200260043

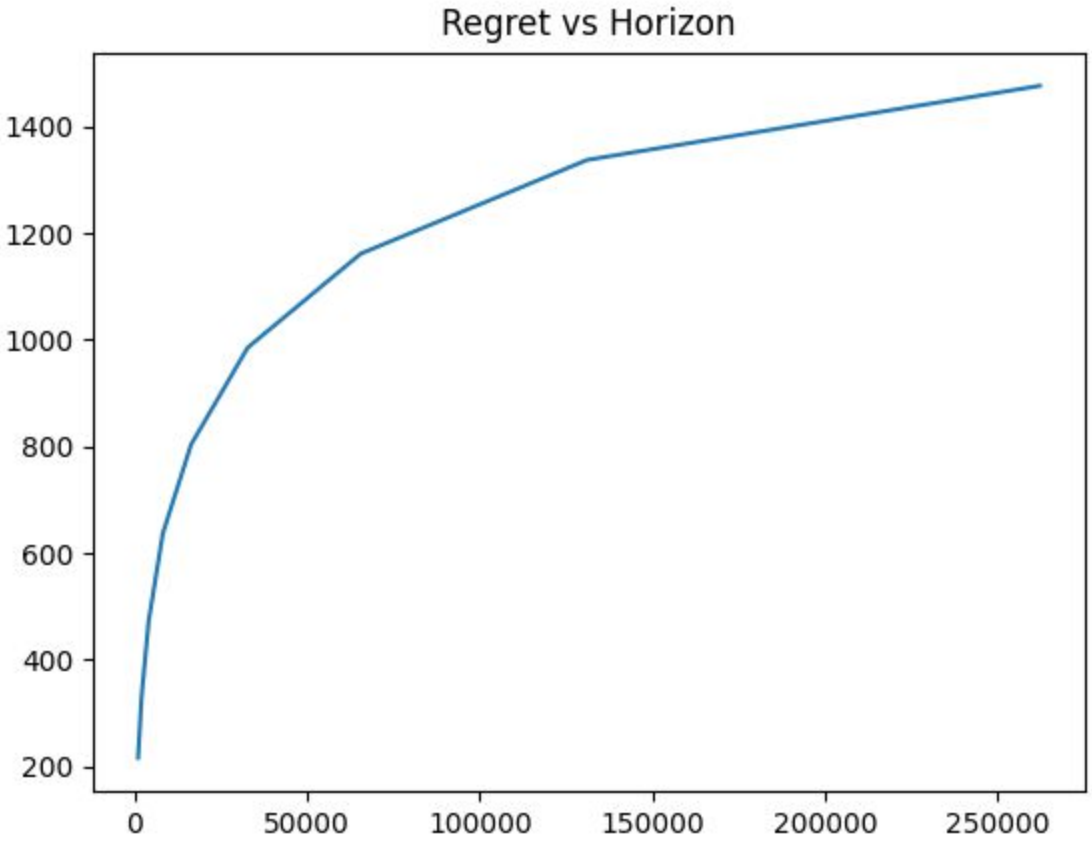
Task 1: KL_UCB

- Def KL(p,q): Calculates `p*np.log(p/q)+(1-p)*np.log((1-p)/(1-q))`
- Def get_kl_ucb(): Uses bisection method to calculate the solution to the equation `p*np.log(p/q)+(1-p)*np.log((1-p)/(1-q)) = (np.log(t) + 3*np.log(np.log(t)))/t`
- Class KL_UCB: Firstly we define the initial variables self.values, self.counts, self.ucb, self.n and self.arms which stores the empirical mean for each arm, the number of times each arm has been pulled, the value of ucb_kl calculated according to the formula given in class slides and which iteration is going on respectively.
- give_pull() : gives the index of the arm which has max value in self.ucb
- get_reward(): updates the values of all the variables.
- The next slide contains the plot from the simulator.py file


Regret vs Horizon
220
200
180
160
140
120
100
80
0
50000
100000
150000
200000
250000

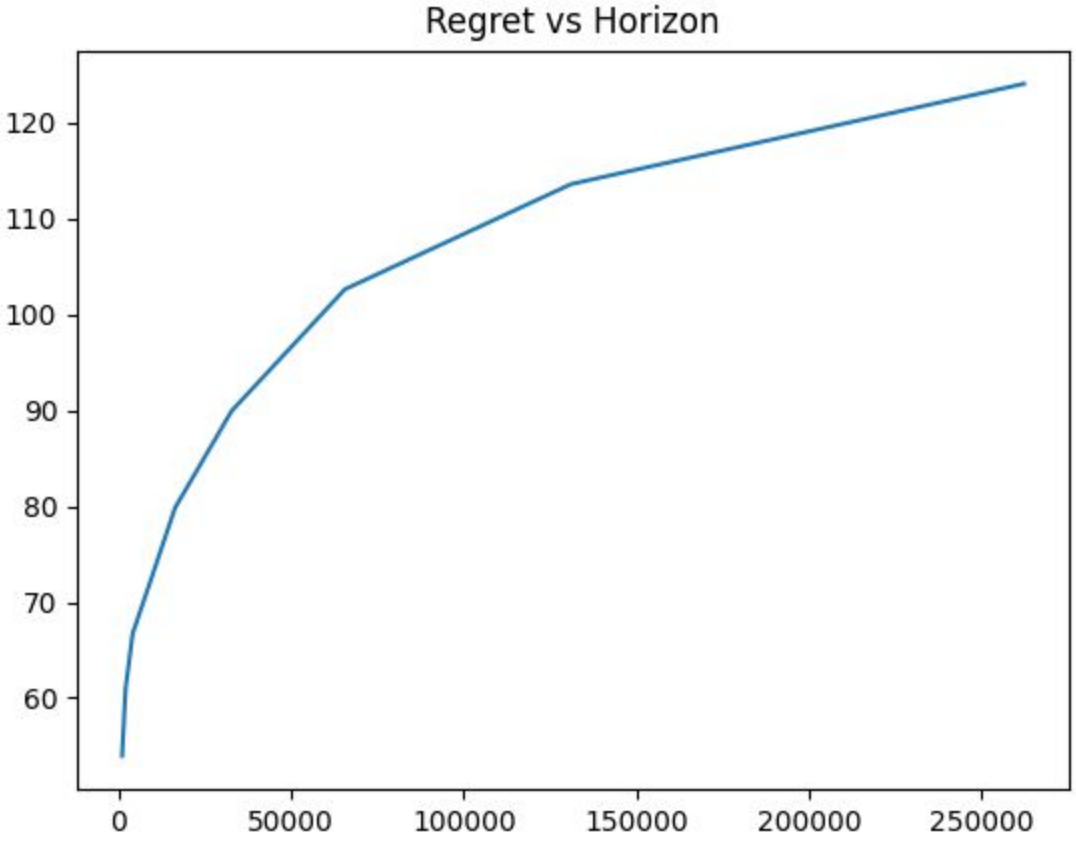
Task 1: UCB

- class UCB(): Firstly we define the initial variables self.values, self.counts, self.ucb, self.n and self.arms which stores the empirical mean for each arm, the number of times each arm has been pulled, the value of ucb calculated according to the formula given in class slides and which iteration is going on respectively.
- give_pull() : gives the index of the arm which has max value in self.ucb
- get_reward(): updates the values of all the variables.
- The next slide contains the plot from the simulator.py file


Regret vs Horizon
1400
1200
1000
800
600
400
200
0
50000
100000
150000
200000
250000

Task 2: Thompson sampling

- class Thompson_Sampling(): Firstly we define the initial variables self.success, self.counts, self.beta, self.n and self.failure which stores the number of success for each arm, the number of times each arm has been pulled, the value of beta calculated using a numpy beta distribution function, which iteration is going on and number of failures of each arm respectively.
- give_pull() : gives the index of the arm which has max value in self.beta
- get_reward(): updates the values of all the variables.
- The next slide contains the plot from the simulator.py file


Regret vs Horizon
120
110
100
90
80
70
60
0
50000
100000
150000
200000
250000

Task 2

My idea for this task was to explore the arms for the first int(eps*Horizon/batch-size) times for some eps and keep updating the beta values for the exploration and then pull a specific number of arms for different batch sizes.

- The pulls will be based on the values of self.beta which is same as for thompson_sampling.
- And the frequencies depend upon the beta values of each arm. Actually it has been scaled throughout the batch size using the powers of corresponding beta values as reference.
- I was able to get a regret of around 350 for each test case

Task 3:

- First part was exploring the outcomes of the arms for 1/4th of the total iterations (Horizon value)
- Second part consist of selecting arms based on the beta values of different arms.