NAME: Kevin Motia
DATE: 2024-02-26
Homework: Assignment 3
Data Science II

Problem 1

The multi-armed bandit problem is essentially an optimization problem where there are multiple choices with unknown rewards. You are attempting to determine which choice yields the highest reward in order to choose that option to maximize your rewards over multiple rounds of choosing. The problem requires a tradeoff between exploring new choices and exploiting the one that has the highest reward so far.

My MAB setup utilizes the HTML file written by Dr. Bagrow as seen in (Question 1, cell 1:2). Probability distributions were chosen to be supported on the interval [0,1]. Five beta distributions were chosen where there are two parameters to tune the mean reward value. The distributions were chosen to be separated evenly on the [0,1] interval, and to have some overlap in their probability densities, but not so much as to make the MAB problem extremely easy or hard.

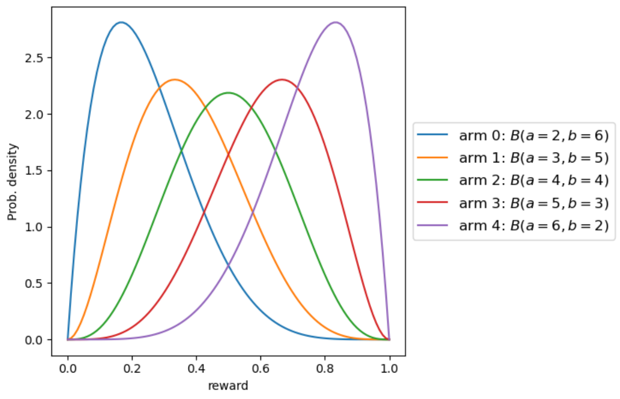


Figure 1: Probability density plot of rewards for 5 armed bandit

Problem 2

Here, we modified the MAB setup to use 6 arms (Question 2, cell 1). The setup follows the same thought process as done in Problem 1.

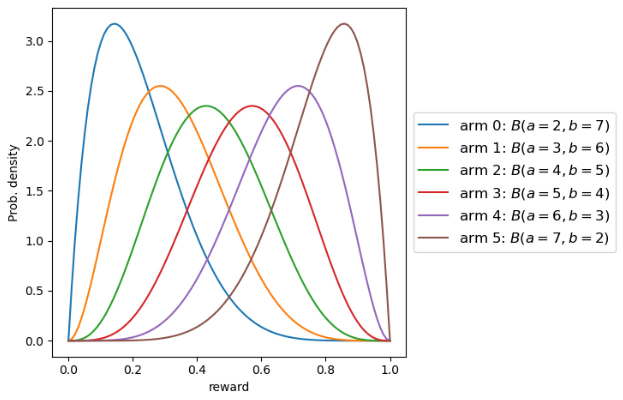


Figure 2: Probability density plot of rewards for 6 armed bandit

Next, I implemented four bandit algorithms: Random, Naive Greedy, ϵ-First Greedy, and ϵ-Greedy. These can be seen in cells labelled as such under Question 2. Each algorithm utilizes 6 arms. All algorithms that rely on initial reward estimates assume an initial estimate of 0.5 for each arm.

The random algorithm simply selects an arm at random for each timestep. The Naive Greedy algorithm chooses the arm with the greatest estimated reward and updates the estimated rewards at each successive timestep. The ϵ-First Greedy algorithm is separated into two parts: an exploration phase and an exploitation phase. The exploration phase lasts for m timesteps, where m is set to 10. During this phase, arms are randomly chosen and their reward estimates are updated. During the exploitation phase, we have the option of exploiting the arm with the current highest reward estimate with probability $1 - \epsilon$ or going back to the exploration phase with probability ϵ. The ϵ-Greedy algorithm functions similarly by having an exploration and exploitation phase, where the difference is in that m steps are not used for an initial exploration phase. Instead we go straight into exploring with probability ϵ or exploiting with probability $1 - \epsilon$.

These strategies are not cheating because they are unable to utilize the true reward distribution for their estimates. Instead, they rely on initial estimates of 0.5 for each arm, and can update their reward estimates for an arm by playing it.

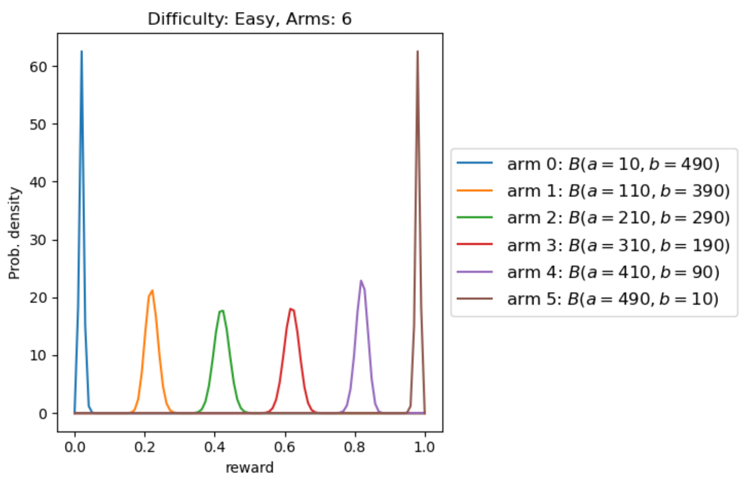
Problem 3

Two scenarios were plotted in this section. An "Easy" bandit and a "Hard" bandit. The four algorithms discussed in Problem 2 are compared for these cases.

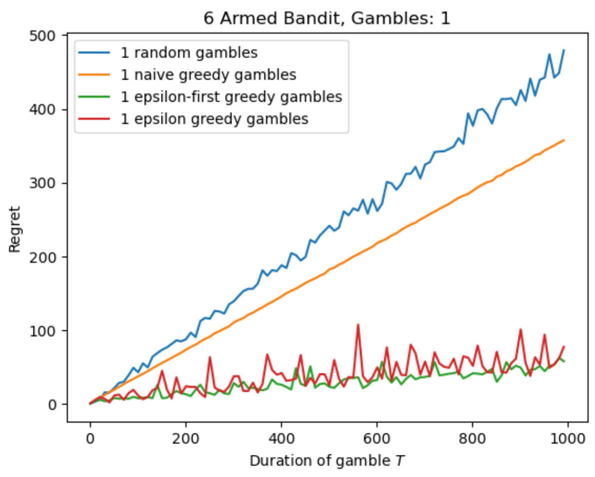
The "Easy" bandit was made easy by choosing a reward distribution that are sufficiently spaced out without overlap. In contrast, the "Hard" bandit uses a reward distribution where there is significant overlap in the reward distributions. This means that given a certain arm, it is difficult for the algorithms to reliably determine which arm yields the best reward. It then requires more gambles to gain a more accurate estimate of the best arm.

For the second metric used to visualize algorithm performance, I decided to use the fraction of optimal arm pulls. It is a direct representation of how well the algorithm at choosing the arm with the highest estimated reward. It is also pretty intuitive in that 1.0 indicates optimal performance.

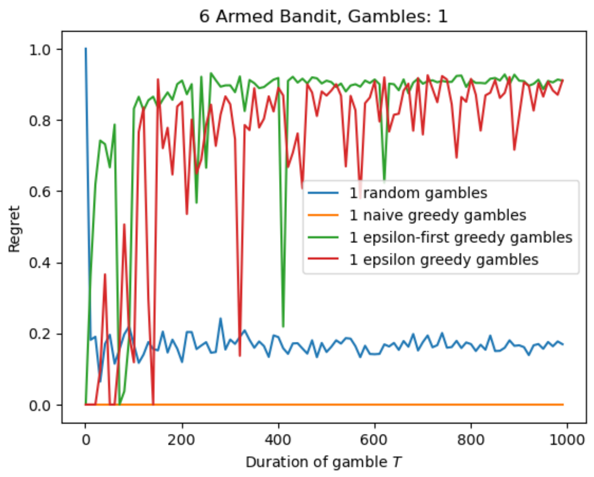
From the performance metrics, it appears that either ϵ greedy performs the best, but it is still relatively close in performance with ϵ-first greedy so I plotted that algorithm as well in Problem 4. This is most likely because ϵ-first greedy does not separate its exploration and exploitation phases so explicitly. It instead combines them for the duration of the gamble time.



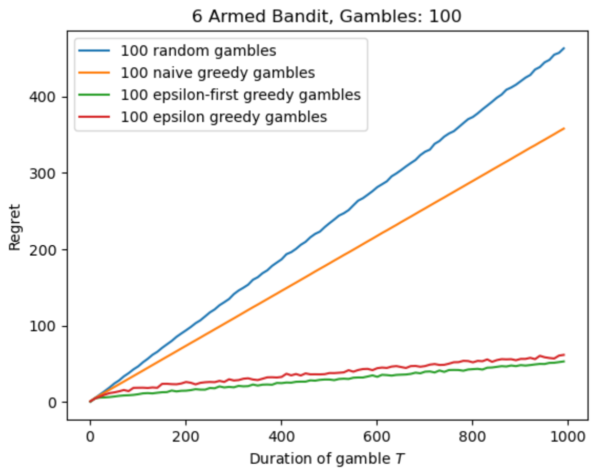
(a)



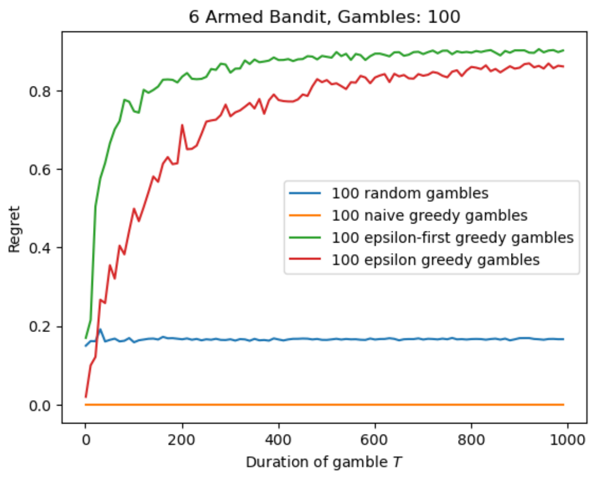
(b)



(c)

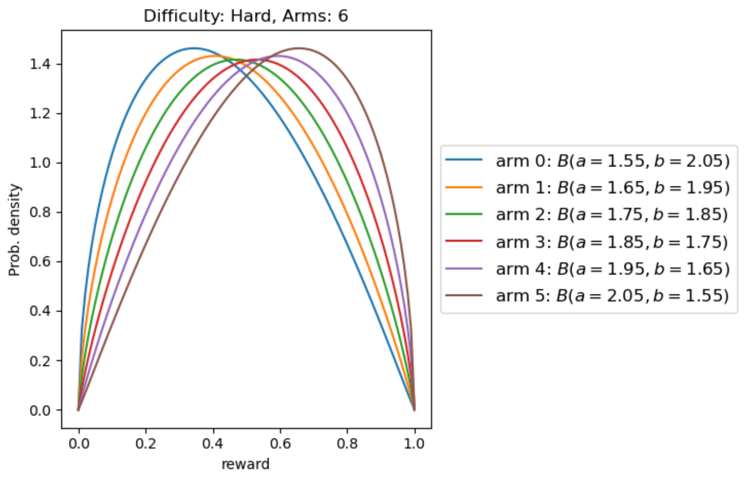


(d)

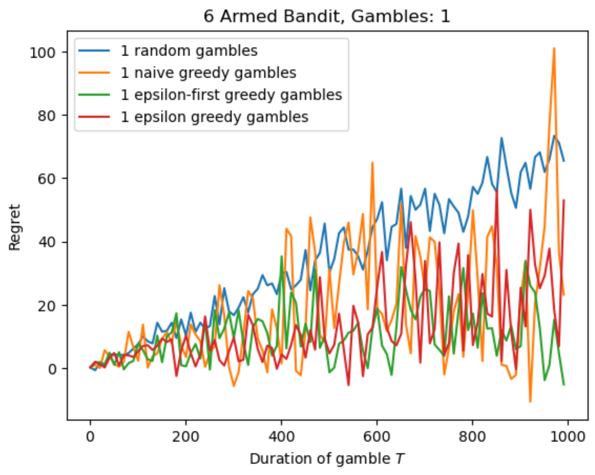


(e)

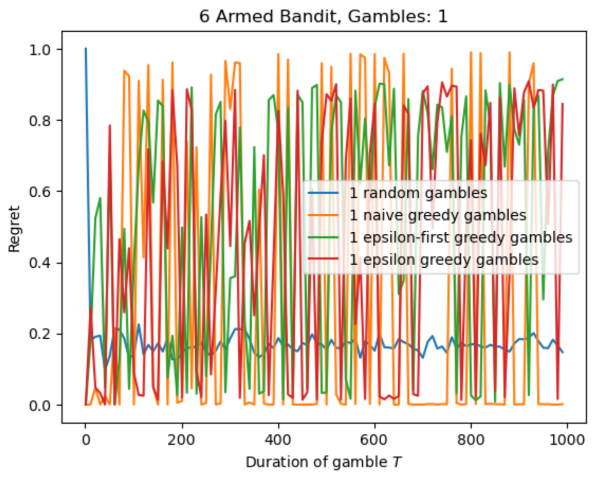
Figure 3: **(a)** shows the probability distribution of each arm and their corresponding rewards. This is for the easy MAB. **(b)** and **(c)** show the plots for the mean regret and fraction of optimal pulls averaged over 1 gamble respectively. **(d)** and **(e)** show these plots again when averaged over 100 gambles.



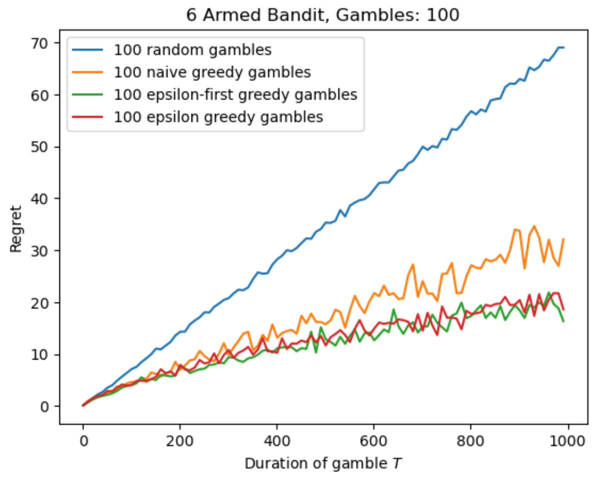
(a)



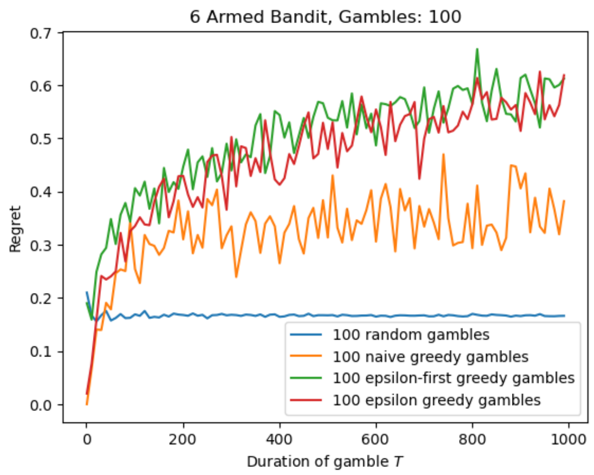
(b)



(c)



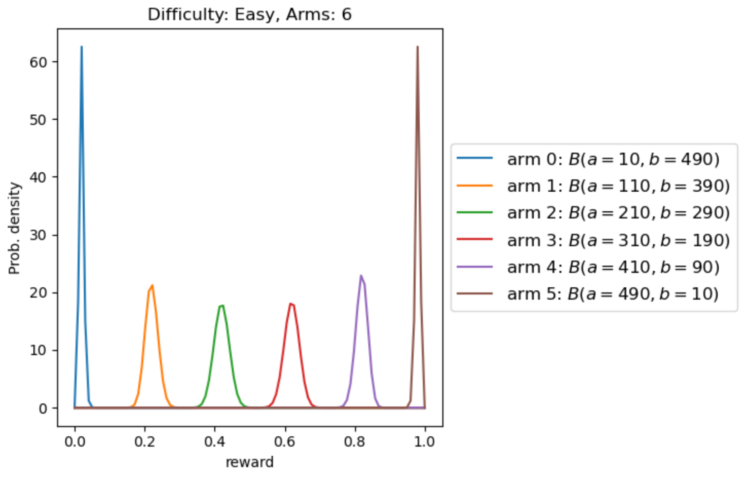
(d)



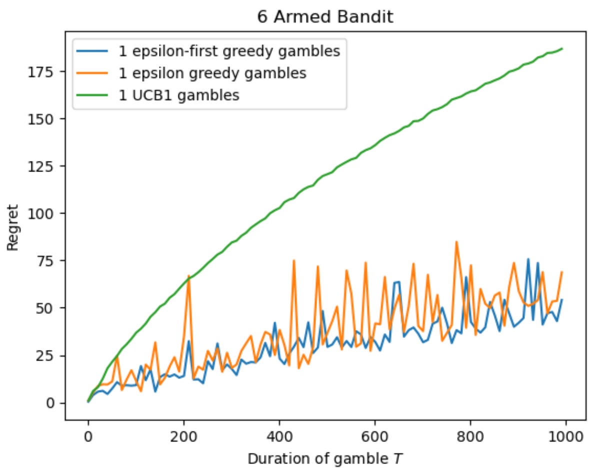
(e)

Figure 4: **(a)** shows the probability distribution of each arm and their corresponding rewards. This is for the hard MAB. **(b)** and **(c)** show the plots for the mean regret and fraction of optimal pulls averaged over 1 gamble respectively. **(d)** and **(e)** show these plots again when averaged over 100 gambles.

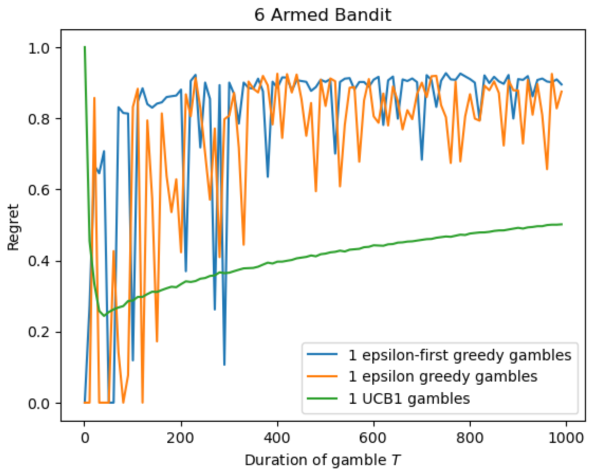
Problem 4



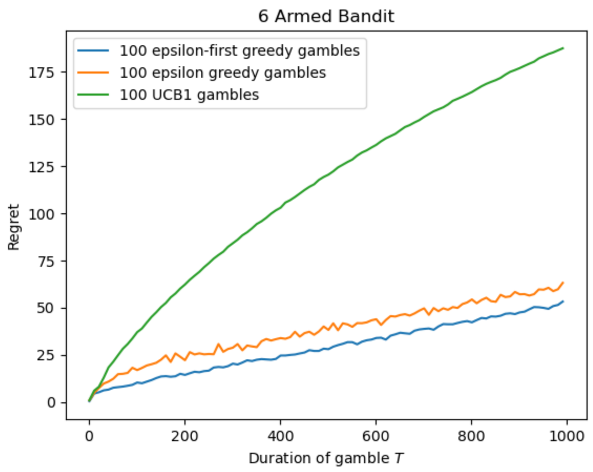
(a)



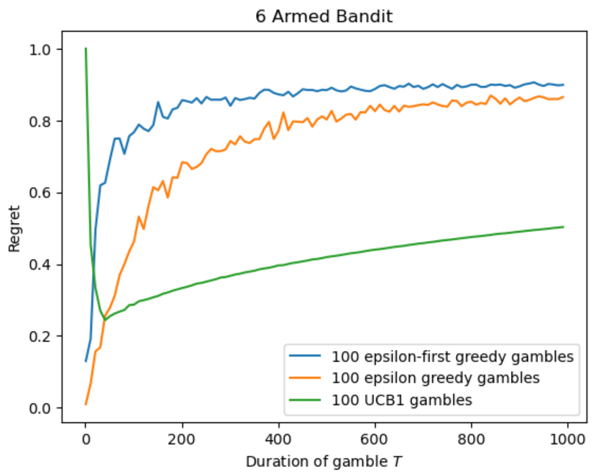
(b)



(c)

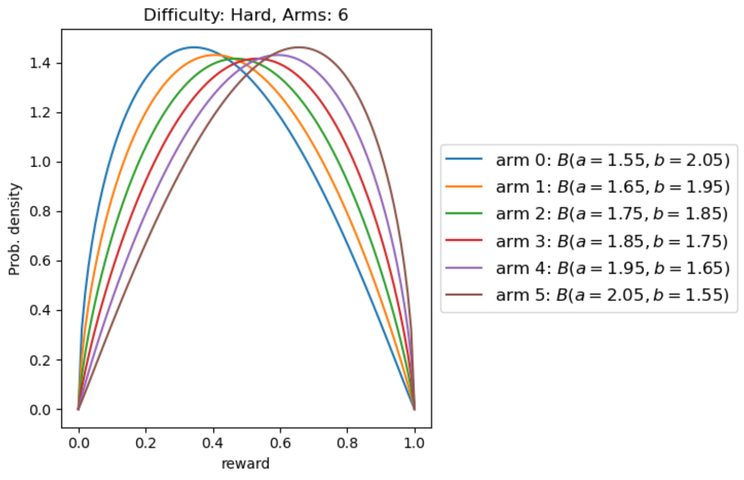


(d)

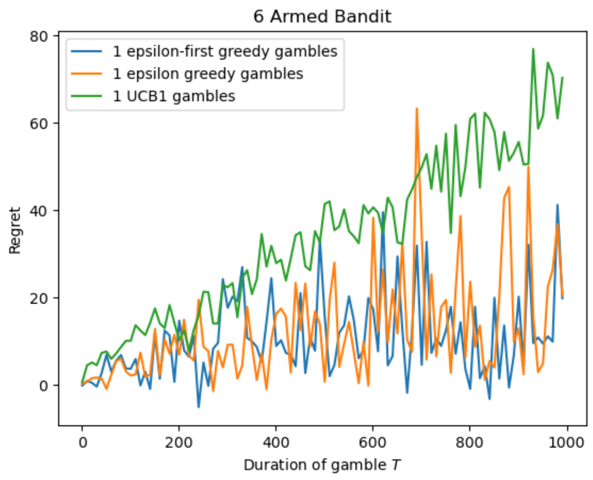


(e)

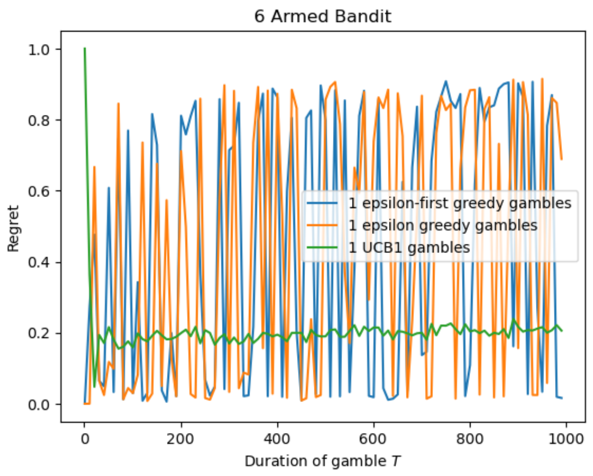
Figure 5: **(a)** shows the probability distribution of each arm and their corresponding rewards. This is for the easy MAB. **(b)** and **(c)** show the plots for the mean regret and fraction of optimal pulls averaged over 1 gamble respectively. **(d)** and **(e)** show these plots again when averaged over 100 gambles. These plots are specifically meant to compare UCB1 to the best performing algorithms from the previous section.



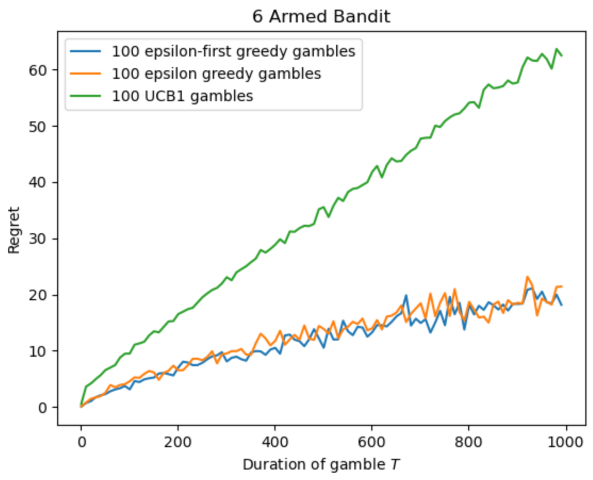
(a)



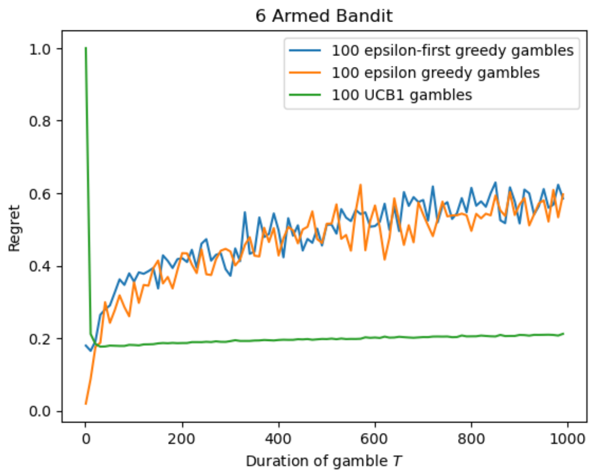
(b)



(c)



(d)



(e)

Figure 6: **(a)** shows the probability distribution of each arm and their corresponding rewards. This is for the hard MAB. **(b)** and **(c)** show the plots for the mean regret and fraction of optimal pulls averaged over 1 gamble respectively. **(d)** and **(e)** show these plots again when averaged over 100 gambles. These plots are specifically meant to compare UCB1 to the best performing algorithms from the previous section.

In the above plots it appears that my attempted implementation of the UCB1 algorithm performs better relative to the other two algorithms. As discussed in the paper Kuleshov reading, the UCB1 algorithm "greedily picks the arm after exploring each one". It picks an arm according to the function stored as ucb_values in (Question 3: UCB1 Implementation). Here, ucb_values stores the upper confidence bound values for an expected reward from a given arm after playing it. Calculating this value appears to be superior than simply using the expected reward.