main

May 5, 2023

1 Assignment 1

In this assignment I will implement an algorithm with Q value, greediness and policy in order to learn the k-bandit algorithm and find a policy that can lead to choosing the best arm.

1.1 Libraries

[2]:
```
import numpy as np
import random
import matplotlib.pyplot as plt

import bandit
```

First of all we added the bandit.py and some other libraries in the main file. main() is the main coding part: - First of all we have timesteps variable that can be set for number of iterations for learning. - Regrets is a set of regrets for showing them at the end - Epsilon is the probability of being greedy. In other words if epsilon = 0.1, it means that 10 percent of the time we are exploiting and 90% of the time we are exploring. - For the learning we are using Policy updating and Q Val algorithm and in order to use them we first initializing QVal and policy. - Delta is the learning rate

For the for loop, first of all we are choosing an arm based on the epsilon, after that we updating the policy array and the we save the policy for having a graph in order to have an illustration.

After that we pull the arm and obtaining the reward and updating QVal.

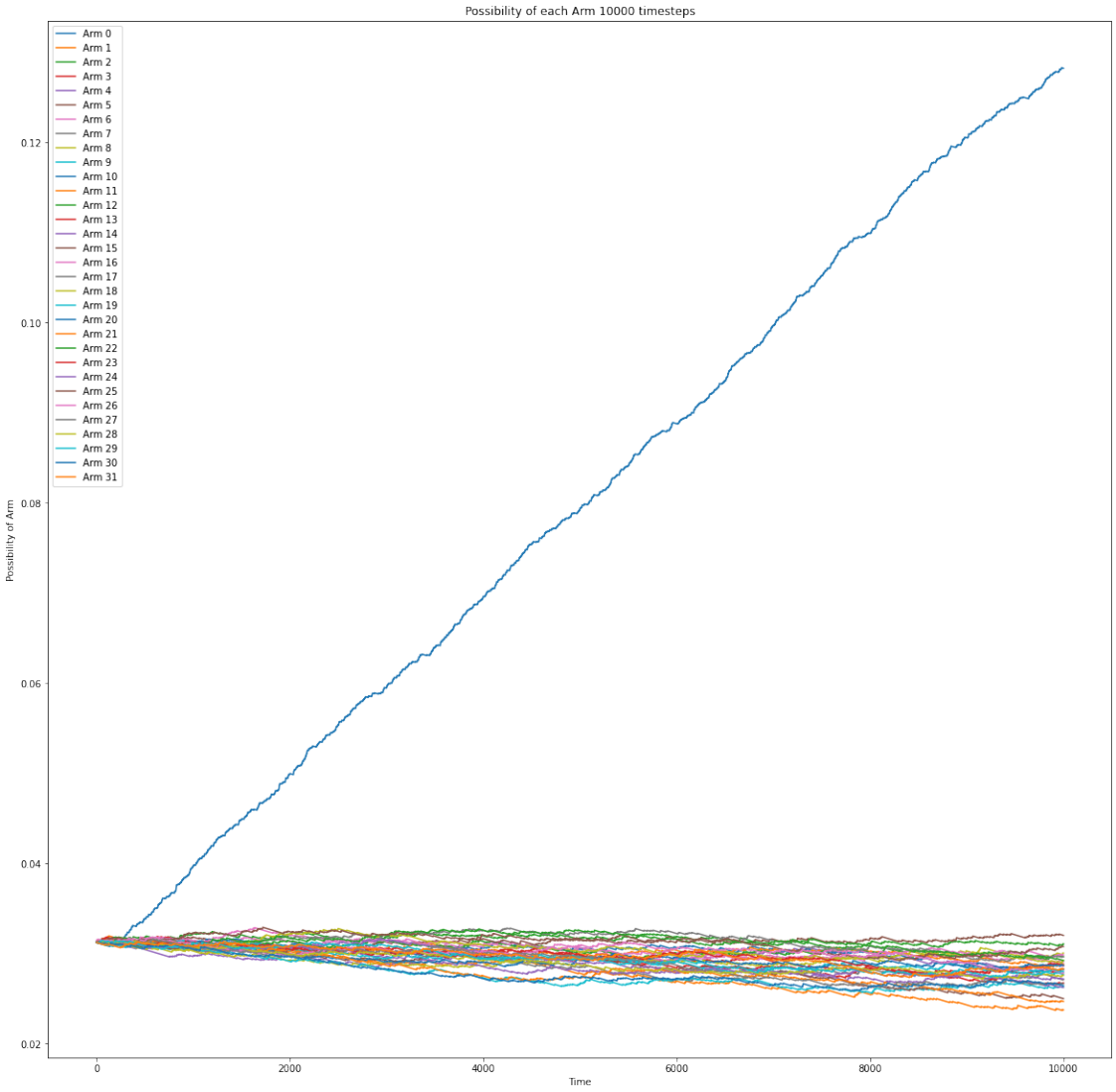
At the end we illustrate the data.

[3]:
```
def main():
    # Number of steps
    timesteps = 10000
    # Import the bendit code
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
    # exploration probability
    # Greediness
    epsilon = 0.1
    # Q Value initialization
    QVal = [0] * b.num_arms()
```

```
    # Counting each move or usage of arm (This can also be use as possibility)
    ActionCount = [0] * b.num_arms()
    # Delto for updating possibility
    delta = 0.0001
    # possibility of each arm
    bandits_p = []
    # Policy initialization
    policy = []
    for i in range(b.num_arms()):
        policy.append(1/b.num_arms())
        bandits_p.append([])

    # for time steps
    for t in range(timesteps):
        # Choose an arm
        a = 0
        if np.random.random() >= epsilon:
            a = random.randrange(b.num_arms())
        else:
            a = QVal.index(max(QVal))

        # Update the policy
        if policy[a] <= (1 - delta):
            policy[a] = policy[a] + delta
            for i in range(b.num_arms()):
                if i != a:
                    if policy[i] >= (delta/(b.num_arms() - 1)):
                        policy[i] = policy[i] - (delta/(b.num_arms() - 1))

        # Save the policy for plotting
        for i in range(b.num_arms()):
            bandits_p[i].append(policy[i])

        # Pull the arm, obtain a reward
        rew = b.trigger(a)
        ActionCount[a] = ActionCount[a] + 1

        # Update the Q-Value
        QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))

        # Save the regret
        regret += b.opt() - rew
        regrets.append(regret)

        # print the data
        if (t % 1000) == 0:
```

```
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
    for i in range(b.num_arms()):
        plt.plot(bandits_p[i], label="Arm " + str(i))
    plt.xlabel('Time')
    plt.ylabel('Possibility of Arm')
    plt.title('Possibility of each Arm ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.248824 regret 0.02481728025669999 Action 11 epsilon 0.1
10.0 from 100.0 Reward 0.252946 regret 22.507630142256676 Action 5 epsilon 0.1
20.0 from 100.0 Reward 0.257093 regret 47.56135691845663 Action 19 epsilon 0.1
30.0 from 100.0 Reward 0.281424 regret 72.41469763495667 Action 1 epsilon 0.1
40.0 from 100.0 Reward 0.280207 regret 97.86137691765678 Action 1 epsilon 0.1
50.0 from 100.0 Reward 0.281244 regret 120.78325461475679 Action 1 epsilon 0.1
60.0 from 100.0 Reward 0.258228 regret 144.9784397752572 Action 3 epsilon 0.1
70.0 from 100.0 Reward 0.2415 regret 167.71302221505735 Action 30 epsilon 0.1
80.0 from 100.0 Reward 0.261181 regret 194.514342172427 Action 7 epsilon 0.1
90.0 from 100.0 Reward 0.261383 regret 220.62959706852706 Action 23 epsilon 0.1
[0.27426852298506377, 0.26341920951375003, 0.2596729776832797,
0.25082182575316886, 0.2582768302801392, 0.25816877581906256,
0.25877423543711364, 0.2560494436857142, 0.20499891210531568,
0.22916275772490555, 0.24599534971201412, 0.24479194441487886,
0.227659069846128, 0.238690171348264, 0.2433481111637361, 0.24620766808174605,
0.24959770297926434, 0.25624852151279476, 0.24974844028771312,
0.2557053308374099, 0.2530655063152777, 0.25289570515582355, 0.2560348866047779,
0.2542458085085503, 0.2079249932405303, 0.23524361623506498,
0.24118033982332154, 0.24337771954514936, 0.23022086787472923,
0.24075222963879017, 0.24151359177443615, 0.24871893438767603]
```



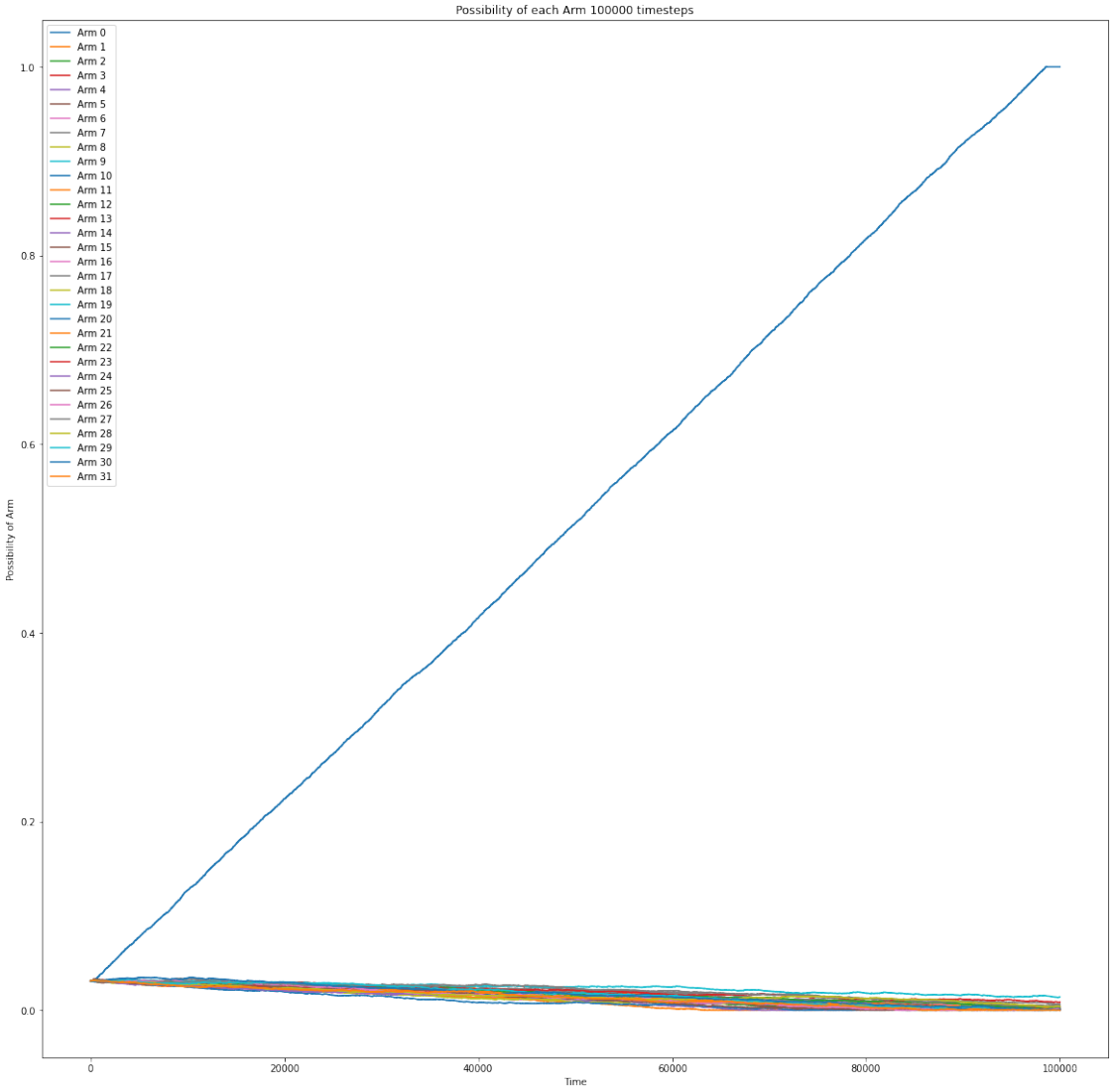
It is clear that after about timestep 1300, arm 0 is dominant. If we continue for 100000 we can see that its going to be 100%.

[4]:
```
def main():
    # Number of steps
    timesteps = 100000
    # Import the bendit code
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
    # exploration probability
    # Greediness
    epsilon = 0.1
```

```
    # Q Value initialization
    QVal = [0] * b.num_arms()
    # Counting each move or usage of arm (This can also be use as possibility)
    ActionCount = [0] * b.num_arms()
    # Delta for updating possibility
    delta = 0.0001
    # possibility of each arm
    bandits_p = []
    # Policy initialization
    policy = []
    for i in range(b.num_arms()):
        policy.append(1/b.num_arms())
        bandits_p.append([])

    # for time steps
    for t in range(timesteps):
        # Choose an arm
        a = 0
        if np.random.random() >= epsilon:
            a = random.randrange(b.num_arms())
        else:
            a = QVal.index(max(QVal))

        # Update the policy
        if policy[a] <= (1 - delta):
            policy[a] = policy[a] + delta
            for i in range(b.num_arms()):
                if i != a:
                    if policy[i] >= (delta/(b.num_arms() - 1)):
                        policy[i] = policy[i] - (delta/(b.num_arms() - 1))

        # Save the policy for plotting
        for i in range(b.num_arms()):
            bandits_p[i].append(policy[i])

        # Pull the arm, obtain a reward
        rew = b.trigger(a)
        ActionCount[a] = ActionCount[a] + 1

        # Update the Q-Value
        QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))

        # Save the regret
        regret += b.opt() - rew
        regrets.append(regret)

        # print the data
```

```
        if (t % 10000) == 0:
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
    for i in range(b.num_arms()):
        plt.plot(bandits_p[i], label="Arm " + str(i))
    plt.xlabel('Time')
    plt.ylabel('Possibility of Arm')
    plt.title('Possibility of each Arm ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.26339 regret 0.010251280256699968 Action 22 epsilon 0.1
10.0 from 100.0 Reward 0.260488 regret 253.68370024853616 Action 3 epsilon 0.1
20.0 from 100.0 Reward 0.255369 regret 502.9124283509357 Action 16 epsilon 0.1
30.0 from 100.0 Reward 0.259823 regret 752.4392231399514 Action 23 epsilon 0.1
40.0 from 100.0 Reward 0.250774 regret 1003.7249942327921 Action 31 epsilon 0.1
50.0 from 100.0 Reward 0.28086 regret 1247.0734059815748 Action 1 epsilon 0.1
60.0 from 100.0 Reward 0.261491 regret 1488.7842344183678 Action 3 epsilon 0.1
70.0 from 100.0 Reward 0.263561 regret 1739.604474407264 Action 4 epsilon 0.1
80.0 from 100.0 Reward 0.26052 regret 1988.8041991078287 Action 8 epsilon 0.1
90.0 from 100.0 Reward 0.251145 regret 2237.2037270904016 Action 28 epsilon 0.1
[0.273219338446338, 0.26263856269556507, 0.25197266790874506,
0.25670807185862005, 0.25550713735460423, 0.25832983706084983,
0.2537284158139202, 0.2565140884691658, 0.20361475467767032, 0.2294008338056526,
0.24263467392455088, 0.24496456137039263, 0.22556774549505434,
0.239453975316424, 0.24479629456832142, 0.24541769958306828,
0.24987900178111255, 0.2581679910804129, 0.2527395281666552,
0.25669415195532325, 0.25494881255167645, 0.25582526852501797,
0.25461587749795994, 0.25486129840282373, 0.21308349102513402,
0.2319191403961876, 0.2421256123321204, 0.2415826667924838, 0.23086824369580086,
0.23975066431540387, 0.2445804190419179, 0.24752194060237437]
```



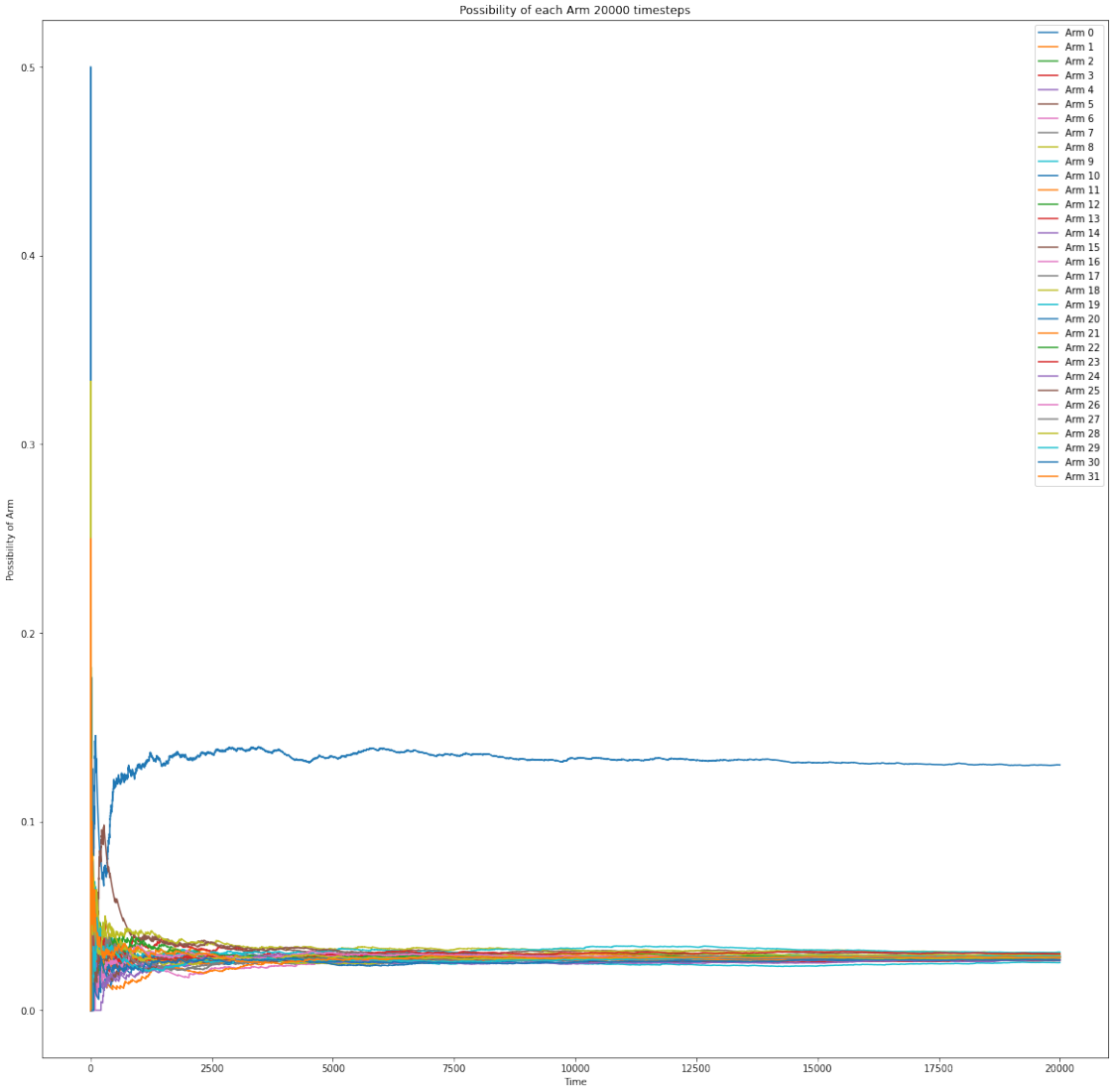
We can also try with the counting of the arms divided by current time step that are used:

[5]:
```
def main():
    # Number of steps
    timesteps = 20000
    # Import the bendit code
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
    # exploration probability
    # Greediness
    epsilon = 0.1
    # Q Value initialization
```

```
    QVal = [0] * b.num_arms()
    # Counting each move or usage of arm (This can also be use as possibility)
    ActionCount = [0] * b.num_arms()
    # possibility of each arm
    bandits_p = []

    # for time steps
    for t in range(timesteps):
        # Choose an arm
        a = 0
        if np.random.random() >= epsilon:
            a = random.randrange(b.num_arms())
        else:
            a = QVal.index(max(QVal))

        for i in range(b.num_arms()):
            bandits_p.append([0])

        for i in range(b.num_arms()):
            bandits_p[i].append(ActionCount[i]/(t+1))

        # Pull the arm, obtain a reward
        rew = b.trigger(a)
        ActionCount[a] = ActionCount[a] + 1

        # Update the Q-Value
        QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))

        # Save the regret
        regret += b.opt() - rew
        regrets.append(regret)

        # print the data
        if (t % 10000) == 0:
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
    for i in range(b.num_arms()):
        plt.plot(bandits_p[i], label="Arm " + str(i))
    plt.xlabel('Time')
    plt.ylabel('Possibility of Arm')
    plt.title('Possibility of each Arm ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
```

```
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.280077 regret -0.006435719743300039 Action 1 epsilon 0.1
50.0 from 100.0 Reward 0.269697 regret 240.31163919505607 Action 2 epsilon 0.1
[0.27368897396178454, 0.2654627642250003, 0.25343928302446783,
0.2574199745335677, 0.25682349545649685, 0.2598677776119729, 0.2561056342102968,
0.2577434985448631, 0.2037628842605913, 0.23129006551266215,
0.24432119024945262, 0.24447466560391046, 0.2248516433761484,
0.2373250183321904, 0.24546914082969934, 0.24878580607762235, 0.251195086649653,
0.26221129373820995, 0.2526503414939651, 0.2538736163500002,
0.25527317499151275, 0.2585787264441556, 0.25551238007300153, 0.25448711318798,
0.2168770533731205, 0.23484448349733117, 0.24240619231909916,
0.24699030237086322, 0.23105349016690402, 0.2408970852168126,
0.24514300342589093, 0.2466618740321492]
```



It's again visible that Arm 0 is the optimal choice and it's visible from timestep about 1500.

Now we are trying with epsilon decay.

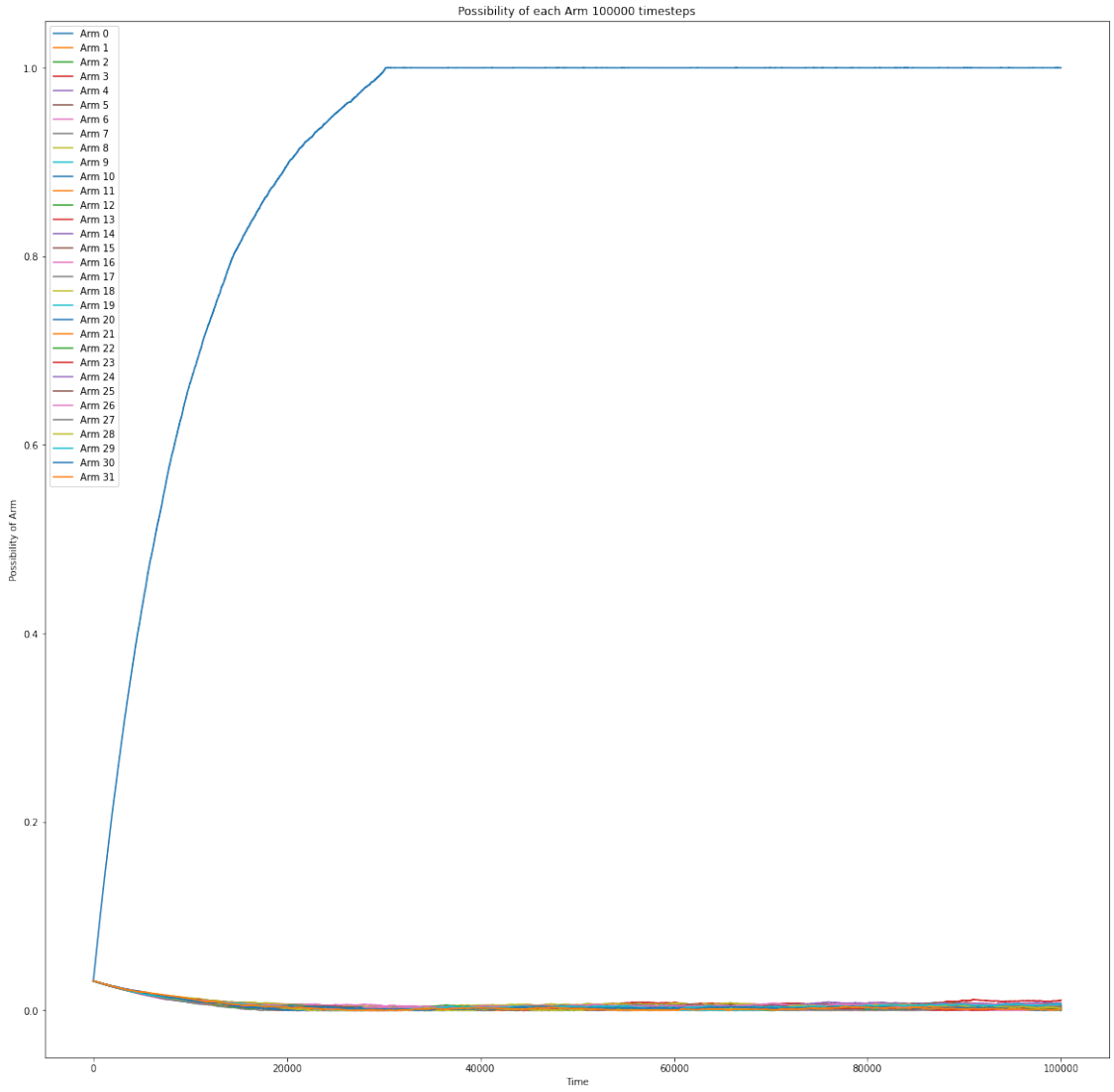
[6]:
```
def main():
    # Number of steps
    timesteps = 100000
    # Import the bendit code
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
    # exploration probability
    # Greediness
```

```
epsilon = 1
decay = 0.9999
min_epsilon = 0.1
# Q Value initialization
QVal = [0] * b.num_arms()
# Counting each move or usage of arm (This can also be use as possibility)
ActionCount = [0] * b.num_arms()
# Delta for updating possibility
delta = 0.0001
# possibility of each arm
bandits_p = []
# Policy initialization
policy = []
for i in range(b.num_arms()):
    policy.append(1/b.num_arms())
    bandits_p.append([])

# for time steps
for t in range(timesteps):
    # Choose an arm
    a = 0
    if np.random.random() >= epsilon:
        a = random.randrange(b.num_arms())
    else:
        a = QVal.index(max(QVal))

    # Update the policy
    if policy[a] <= (1 - delta):
        policy[a] = policy[a] + delta
        for i in range(b.num_arms()):
            if i != a:
                if policy[i] >= (delta/(b.num_arms() - 1)):
                    policy[i] = policy[i] - (delta/(b.num_arms() - 1))

    # Save the policy for plotting
    for i in range(b.num_arms()):
        bandits_p[i].append(policy[i])

    # Pull the arm, obtain a reward
    rew = b.trigger(a)
    ActionCount[a] = ActionCount[a] + 1

    # Update the Q-Value
    QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))

    # Save the regret
    regret += b.opt() - rew
```

```
        regrets.append(regret)

        # updating epsilon
        epsilon = max([min_epsilon, epsilon * decay])

        # print the data
        if (t % 10000) == 0:
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
    for i in range(b.num_arms()):
        plt.plot(bandits_p[i], label="Arm " + str(i))
    plt.xlabel('Time')
    plt.ylabel('Possibility of Arm')
    plt.title('Possibility of each Arm ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.278471 regret -0.004829719743300043 Action 1 epsilon
0.9999
10.0 from 100.0 Reward 0.252647 regret 107.08410424875639 Action 28 epsilon
0.3678242603283259
20.0 from 100.0 Reward 0.261613 regret 320.1042779017751 Action 8 epsilon
0.13530821730781062
30.0 from 100.0 Reward 0.261728 regret 568.2974569868555 Action 5 epsilon 0.1
40.0 from 100.0 Reward 0.258196 regret 810.7541650166212 Action 17 epsilon 0.1
50.0 from 100.0 Reward 0.24772 regret 1054.9784901964365 Action 14 epsilon 0.1
60.0 from 100.0 Reward 0.261417 regret 1304.2667900401536 Action 21 epsilon 0.1
70.0 from 100.0 Reward 0.282116 regret 1556.8616486884114 Action 1 epsilon 0.1
80.0 from 100.0 Reward 0.254105 regret 1805.1562234733815 Action 32 epsilon 0.1
90.0 from 100.0 Reward 0.257259 regret 2052.135655781079 Action 17 epsilon 0.1
[0.27364765792368656, 0.26352794405325186, 0.25388814962461415,
0.25542180499559936, 0.2565308467952936, 0.25886693125408944,
0.2545227708803909, 0.255898916122795, 0.20350002750916665, 0.2291367412693414,
0.2437149803123733, 0.24415433863716987, 0.2258903850528382,
0.23841863748331496, 0.2445359144600156, 0.24680048207873903,
0.24812796332095252, 0.2565310866153553, 0.2531813037696746,
0.25575135187281867, 0.2532876960153039, 0.2551408129700943,
0.25490067932870447, 0.25517702070432785, 0.21410984096406196,
0.23154547364986391, 0.24378101185997747, 0.2426612332717956,
```

```
0.22970299652565052, 0.23997892639108384, 0.24563525285687887,
0.2478010049957876]
```



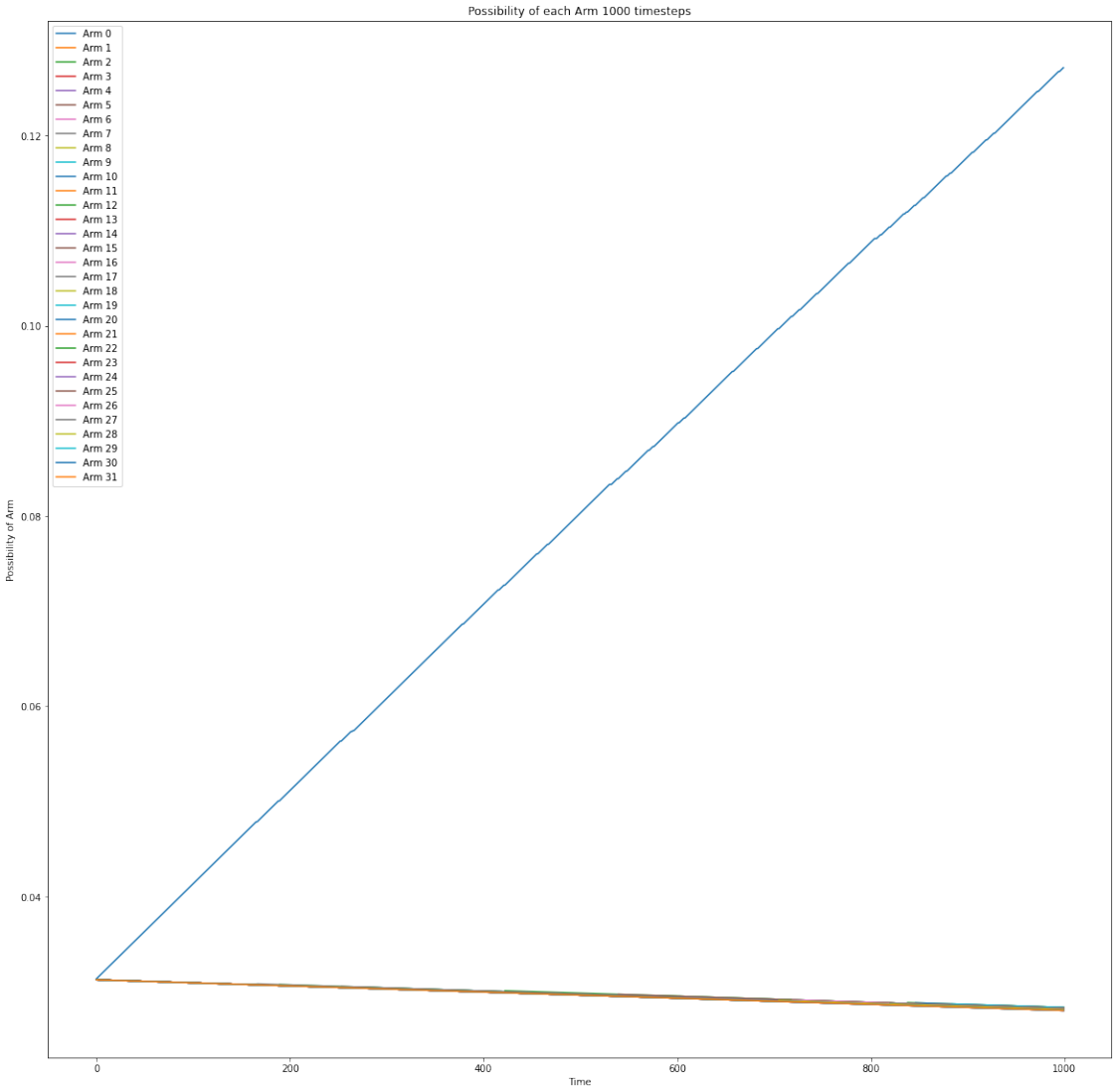
This technique is works extremly better and it will be obvious from almost 0. You can see more details in next part with 1000 timesteps.

[7]:
```
def main():
    # Number of steps
    timesteps = 1000
    # Import the bendit code
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
```

```
    # exploration probability
    # Greediness
    epsilon = 1
    decay = 0.9999
    min_epsilon = 0.1
    # Q Value initialization
    QVal = [0] * b.num_arms()
    # Counting each move or usage of arm (This can also be use as possibility)
    ActionCount = [0] * b.num_arms()
    # Delta for updating possibility
    delta = 0.0001
    # possibility of each arm
    bandits_p = []
    # Policy initialization
    policy = []
    for i in range(b.num_arms()):
        policy.append(1/b.num_arms())
        bandits_p.append([])

    # for time steps
    for t in range(timesteps):
        # Choose an arm
        a = 0
        if np.random.random() >= epsilon:
            a = random.randrange(b.num_arms())
        else:
            a = QVal.index(max(QVal))

        # Update the policy
        if policy[a] <= (1 - delta):
            policy[a] = policy[a] + delta
            for i in range(b.num_arms()):
                if i != a:
                    if policy[i] >= (delta/(b.num_arms() - 1)):
                        policy[i] = policy[i] - (delta/(b.num_arms() - 1))

        # Save the policy for plotting
        for i in range(b.num_arms()):
            bandits_p[i].append(policy[i])

        # Pull the arm, obtain a reward
        rew = b.trigger(a)
        ActionCount[a] = ActionCount[a] + 1

        # Update the Q-Value
        QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))
```

```
        # Save the regret
        regret += b.opt() - rew
        regrets.append(regret)

        # updating epsilon
        epsilon = max([min_epsilon, epsilon * decay])

        # print the data
        if (t % 10000) == 0:
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
    for i in range(b.num_arms()):
        plt.plot(bandits_p[i], label="Arm " + str(i))
    plt.xlabel('Time')
    plt.ylabel('Possibility of Arm')
    plt.title('Possibility of each Arm ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.279867 regret -0.0062257197432999956 Action 1 epsilon
0.9999
[0.2719012937967708, 0, 0.2619453333333333, 0, 0.260711, 0.2645485, 0.2629375,
0.26154333333333335, 0.14253768143333334, 0.238602, 0.24865633333333334, 0,
0.237646, 0, 0, 0.253421, 0, 0.26510449999999997, 0.264337, 0.26163633333333336,
0, 0.261581, 0.260492, 0.26067799999999997, 0.172243, 0.12099903585, 0.248375,
0.2527755, 0.235127, 0, 0, 0]
```



So it is visible even from start. And now we can see the counting of each arm divided by current time step.

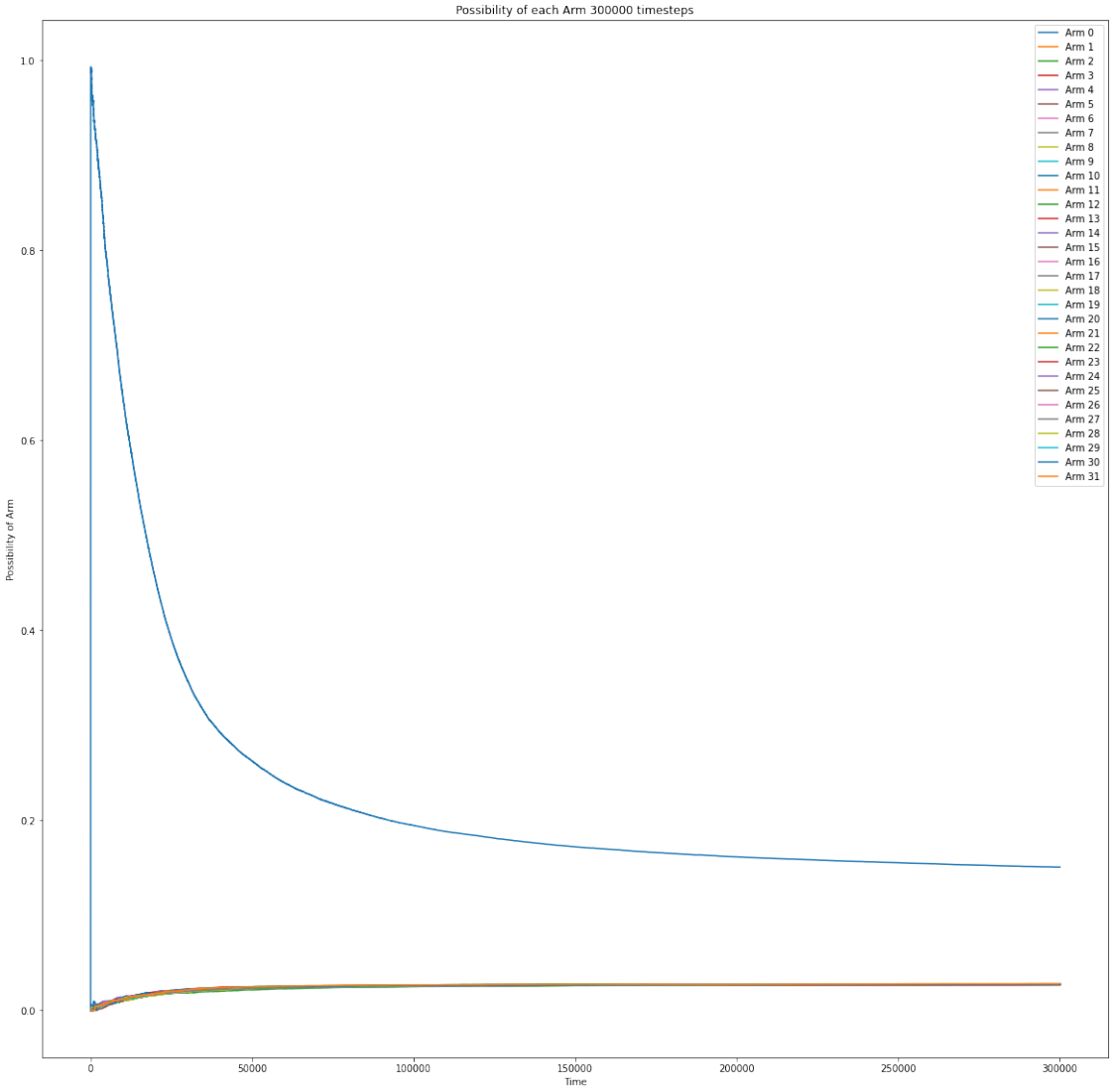
[8]:
```
def main():
    # Number of steps
    timesteps = 300000
    # Import the bendit code
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
    # exploration probability
    # Greediness
    epsilon = 1
```

```
    decay = 0.9999
    min_epsilon = 0.1
    # Q Value initialization
    QVal = [0] * b.num_arms()
    # Counting each move or usage of arm (This can also be use as possibility)
    ActionCount = [0] * b.num_arms()
    # possibility of each arm
    bandits_p = []

    # for time steps
    for t in range(timesteps):
        # Choose an arm
        a = 0
        if np.random.random() >= epsilon:
            a = random.randrange(b.num_arms())
        else:
            a = QVal.index(max(QVal))

        for i in range(b.num_arms()):
            bandits_p.append([0])

        for i in range(b.num_arms()):
            bandits_p[i].append(ActionCount[i]/(t+1))

        # Pull the arm, obtain a reward
        rew = b.trigger(a)
        ActionCount[a] = ActionCount[a] + 1

        # Update the Q-Value
        QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))

        # Save the regret
        regret += b.opt() - rew
        regrets.append(regret)

        # updating epsilon
        epsilon = max([min_epsilon, epsilon * decay])

        # print the data
        if (t % 10000) == 0:
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
```

```
    for i in range(b.num_arms()):
        plt.plot(bandits_p[i], label="Arm " + str(i))
    plt.xlabel('Time')
    plt.ylabel('Possibility of Arm')
    plt.title('Possibility of each Arm ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.282303 regret -0.008661719743300045 Action 1 epsilon
0.9999
3.3333333333333335 from 100.0 Reward 0.248929 regret 103.33248812017708 Action
12 epsilon 0.3678242603283259
6.666666666666667 from 100.0 Reward 0.259193 regret 315.80664676222665 Action 19
epsilon 0.13530821730781062
10.0 from 100.0 Reward 0.226668 regret 560.9549446525247 Action 25 epsilon 0.1
13.333333333333334 from 100.0 Reward 0.280662 regret 811.7493752142112 Action 1
epsilon 0.1
16.666666666666668 from 100.0 Reward 0.248771 regret 1050.0901707269304 Action
11 epsilon 0.1
20.0 from 100.0 Reward 0.24972 regret 1303.761727703833 Action 27 epsilon 0.1
23.333333333333332 from 100.0 Reward 0.212192 regret 1545.2542305945312 Action 9
epsilon 0.1
26.666666666666668 from 100.0 Reward 0.223954 regret 1798.5740464438304 Action
25 epsilon 0.1
30.0 from 100.0 Reward 0.264105 regret 2042.0594561522541 Action 20 epsilon 0.1
33.333333333333336 from 100.0 Reward 0.247352 regret 2288.202132278954 Action 30
epsilon 0.1
36.666666666666664 from 100.0 Reward 0.280741 regret 2544.5580668491516 Action 1
epsilon 0.1
40.0 from 100.0 Reward 0.262572 regret 2787.0167907132504 Action 7 epsilon 0.1
43.333333333333336 from 100.0 Reward 0.266131 regret 3037.346020613628 Action 6
epsilon 0.1
46.666666666666664 from 100.0 Reward 0.258145 regret 3274.864182585455 Action 21
epsilon 0.1
50.0 from 100.0 Reward 0.281109 regret 3522.564354320283 Action 1 epsilon 0.1
53.333333333333336 from 100.0 Reward 0.246851 regret 3763.887364586206 Action 15
epsilon 0.1
56.666666666666664 from 100.0 Reward 0.258593 regret 4008.7165688951063 Action
17 epsilon 0.1
60.0 from 100.0 Reward 0.277541 regret 4249.390327661332 Action 1 epsilon 0.1
63.333333333333336 from 100.0 Reward 0.269045 regret 4494.531648536371 Action 2
epsilon 0.1
66.66666666666667 from 100.0 Reward 0.261323 regret 4742.382115225272 Action 23
epsilon 0.1
```

```
70.0 from 100.0 Reward 0.260115 regret 4988.621083631183 Action 21 epsilon 0.1
73.33333333333333 from 100.0 Reward 0.2595 regret 5231.971382509589 Action 8
epsilon 0.1
76.66666666666667 from 100.0 Reward 0.242758 regret 5469.808577599503 Action 30
epsilon 0.1
80.0 from 100.0 Reward 0.261716 regret 5716.119692272007 Action 5 epsilon 0.1
83.33333333333333 from 100.0 Reward 0.261134 regret 5961.846205574525 Action 8
epsilon 0.1
86.66666666666667 from 100.0 Reward 0.262718 regret 6207.944658124452 Action 18
epsilon 0.1
90.0 from 100.0 Reward 0.201371 regret 6453.4097262335 Action 9 epsilon 0.1
93.33333333333333 from 100.0 Reward 0.277737 regret 6707.385597540402 Action 1
epsilon 0.1
96.66666666666667 from 100.0 Reward 3.727e-05 regret 6953.842956034754 Action 12
epsilon 0.1
[0.2733963189203498, 0.2636286308440487, 0.25360943410647047,
0.2560276056706766, 0.2562778984834178, 0.2595935671549928, 0.2555031720636662,
0.2561901682300273, 0.202910322318441, 0.23057119560749142, 0.24382203869963537,
0.245197024695031, 0.2240245940686403, 0.2382908035179843, 0.24492513383604408,
0.24752552998151045, 0.24949932987152829, 0.2586614950051182,
0.25235155295789696, 0.25644859227944106, 0.25346049347382543,
0.2562324313887477, 0.25470457281666753, 0.2557750443280993, 0.2136761469956916,
0.2312719913167876, 0.24249554324125885, 0.24284455024407794,
0.23083729568889003, 0.23997525095649602, 0.24433718364569523,
0.2472790130636415]
```



It's again obvious the Arm 0 is dominant from the begining and the prossibility of Arm 0 converges.

We can see the rewards and regrets as well.

[15]:
```
def main():
    # Number of steps
    timesteps = 1000000
    # Import the bendit code
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
    # exploration probability
    # Greediness
```

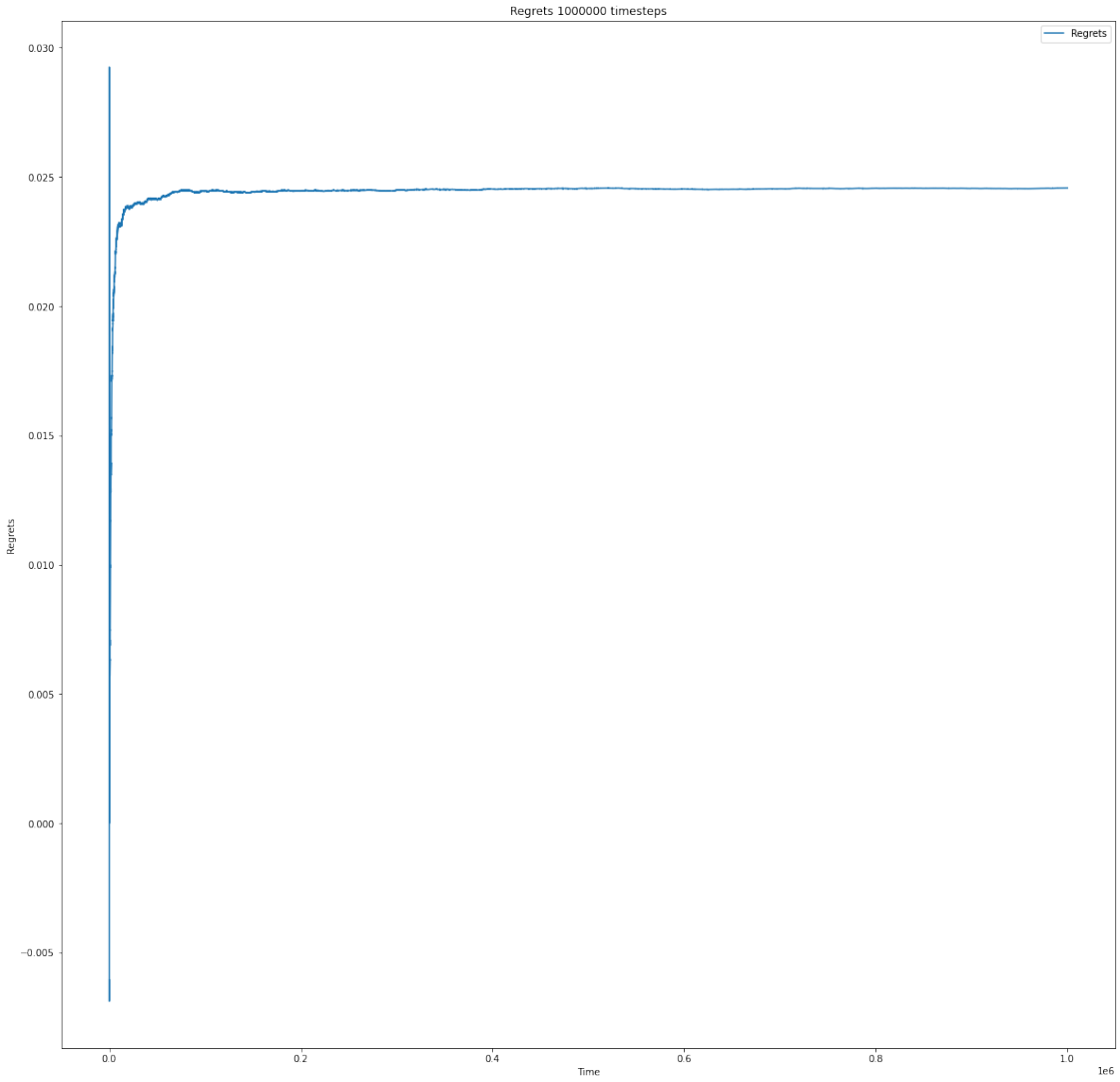
```
epsilon = 1
decay = 0.999
min_epsilon = 0.1
# Q Value initialization
QVal = [0] * b.num_arms()
# Counting each move or usage of arm (This can also be use as possibility)
ActionCount = [0] * b.num_arms()
# Delta for updating possibility
delta = 0.0001
# possibility of each arm
bandits_p = []
# Policy initialization
policy = []
for i in range(b.num_arms()):
    policy.append(1/b.num_arms())
    bandits_p.append([])

# for time steps
for t in range(timesteps):
    # Choose an arm
    a = 0
    if np.random.random() >= epsilon:
        a = random.randrange(b.num_arms())
    else:
        a = QVal.index(max(QVal))

    # Update the policy
    if policy[a] <= (1 - delta):
        policy[a] = policy[a] + delta
        for i in range(b.num_arms()):
            if i != a:
                if policy[i] >= (delta/(b.num_arms() - 1)):
                    policy[i] = policy[i] - (delta/(b.num_arms() - 1))

    # Save the policy for plotting
    for i in range(b.num_arms()):
        bandits_p[i].append(policy[i])

    # Pull the arm, obtain a reward
    rew = b.trigger(a)
    ActionCount[a] = ActionCount[a] + 1

    # Update the Q-Value
    QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))

    # Save the regret
    regret += b.opt() - rew
```

```
        regrets.append(regret/(t+1))

        # updating epsilon
        epsilon = max([min_epsilon, epsilon * decay])

        # print the data
        if (t % 10000) == 0:
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
    plt.plot(regrets, label="Regrets")
    plt.xlabel('Time')
    plt.ylabel('Regrets')
    plt.title('Regrets ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.279702 regret -0.006060719743300025 Action 1 epsilon
0.999
1.0 from 100.0 Reward 0.258653 regret 230.9971812439572 Action 19 epsilon 0.1
2.0 from 100.0 Reward 0.252069 regret 475.9052512262603 Action 28 epsilon 0.1
3.0 from 100.0 Reward 0.251176 regret 719.650557362236 Action 12 epsilon 0.1
4.0 from 100.0 Reward 0.232272 regret 965.0625926494015 Action 13 epsilon 0.1
5.0 from 100.0 Reward 0.243962 regret 1205.7699315467032 Action 14 epsilon 0.1
6.0 from 100.0 Reward 0.25188 regret 1456.0607771350296 Action 16 epsilon 0.1
7.0 from 100.0 Reward 0.250493 regret 1710.0168637595325 Action 27 epsilon 0.1
8.0 from 100.0 Reward 0.249268 regret 1957.7563063412804 Action 27 epsilon 0.1
9.0 from 100.0 Reward 0.279253 regret 2195.5832923480857 Action 1 epsilon 0.1
10.0 from 100.0 Reward 0.279931 regret 2446.1650218696914 Action 1 epsilon 0.1
11.0 from 100.0 Reward 3.01702e-05 regret 2692.1368115407804 Action 16 epsilon
0.1
12.0 from 100.0 Reward 0.281149 regret 2931.8637082043856 Action 1 epsilon 0.1
13.0 from 100.0 Reward 0.239553 regret 3172.7602996374967 Action 29 epsilon 0.1
14.0 from 100.0 Reward 0.24677 regret 3416.9416217378 Action 14 epsilon 0.1
15.0 from 100.0 Reward 0.268977 regret 3662.145387525494 Action 2 epsilon 0.1
16.0 from 100.0 Reward 0.263605 regret 3911.244452227387 Action 22 epsilon 0.1
17.0 from 100.0 Reward 0.262957 regret 4152.587247929377 Action 20 epsilon 0.1
18.0 from 100.0 Reward 0.282389 regret 4405.345515053661 Action 1 epsilon 0.1
19.0 from 100.0 Reward 0.24783 regret 4647.825065969698 Action 15 epsilon 0.1
20.0 from 100.0 Reward 1.77477e-05 regret 4891.640002527697 Action 24 epsilon
```

```
0.1
21.0 from 100.0 Reward 0.239732 regret 5136.710928267172 Action 29 epsilon 0.1
22.0 from 100.0 Reward 0.228348 regret 5380.941352865862 Action 25 epsilon 0.1
23.0 from 100.0 Reward 0.23694 regret 5625.328601918543 Action 10 epsilon 0.1
24.0 from 100.0 Reward 0.261554 regret 5873.269206945431 Action 24 epsilon 0.1
25.0 from 100.0 Reward 0.264567 regret 6117.328648888271 Action 18 epsilon 0.1
26.0 from 100.0 Reward 0.215334 regret 6361.575199214207 Action 9 epsilon 0.1
27.0 from 100.0 Reward 0.248246 regret 6614.01690464415 Action 15 epsilon 0.1
28.0 from 100.0 Reward 0.250968 regret 6852.468506348667 Action 32 epsilon 0.1
29.0 from 100.0 Reward 0.26149 regret 7091.5301333737325 Action 24 epsilon 0.1
30.0 from 100.0 Reward 0.262748 regret 7345.830996815389 Action 20 epsilon 0.1
31.0 from 100.0 Reward 0.25204 regret 7587.67223855796 Action 31 epsilon 0.1
32.0 from 100.0 Reward 0.281151 regret 7842.947952574552 Action 1 epsilon 0.1
33.0 from 100.0 Reward 0.263525 regret 8090.279404270189 Action 4 epsilon 0.1
34.0 from 100.0 Reward 0.254632 regret 8335.323006340592 Action 32 epsilon 0.1
35.0 from 100.0 Reward 0.252263 regret 8579.504887667415 Action 12 epsilon 0.1
36.0 from 100.0 Reward 0.244202 regret 8822.015404356229 Action 27 epsilon 0.1
37.0 from 100.0 Reward 0.262191 regret 9061.410917392941 Action 8 epsilon 0.1
38.0 from 100.0 Reward 3.19467e-05 regret 9307.287055013061 Action 6 epsilon 0.1
39.0 from 100.0 Reward 0.265516 regret 9559.629059415827 Action 6 epsilon 0.1
40.0 from 100.0 Reward 0.266604 regret 9810.100101588085 Action 6 epsilon 0.1
41.0 from 100.0 Reward 0.252753 regret 10052.138864651024 Action 28 epsilon 0.1
42.0 from 100.0 Reward 0.259936 regret 10302.63093358191 Action 3 epsilon 0.1
43.0 from 100.0 Reward 0.257122 regret 10551.487070952922 Action 21 epsilon 0.1
44.0 from 100.0 Reward 0.271697 regret 10795.51308658207 Action 2 epsilon 0.1
45.0 from 100.0 Reward 0.251262 regret 11043.983639892314 Action 15 epsilon 0.1
46.0 from 100.0 Reward 0.251703 regret 11290.788715585828 Action 31 epsilon 0.1
47.0 from 100.0 Reward 0.262102 regret 11539.79145371892 Action 5 epsilon 0.1
48.0 from 100.0 Reward 0.246383 regret 11782.14117982891 Action 11 epsilon 0.1
49.0 from 100.0 Reward 0.262943 regret 12026.84231551187 Action 6 epsilon 0.1
50.0 from 100.0 Reward 0.260859 regret 12273.891051873286 Action 19 epsilon 0.1
51.0 from 100.0 Reward 0.240578 regret 12523.024192209326 Action 29 epsilon 0.1
52.0 from 100.0 Reward 0.255749 regret 12774.556536730432 Action 17 epsilon 0.1
53.0 from 100.0 Reward 0.234108 regret 13019.020034577621 Action 29 epsilon 0.1
54.0 from 100.0 Reward 0.26193 regret 13253.376931271037 Action 24 epsilon 0.1
55.0 from 100.0 Reward 0.251424 regret 13498.853229085473 Action 16 epsilon 0.1
56.0 from 100.0 Reward 0.261246 regret 13740.21736715268 Action 23 epsilon 0.1
57.0 from 100.0 Reward 0.261216 regret 13980.493447866704 Action 5 epsilon 0.1
58.0 from 100.0 Reward 0.255296 regret 14223.70832378707 Action 32 epsilon 0.1
59.0 from 100.0 Reward 0.282771 regret 14466.415669452028 Action 1 epsilon 0.1
60.0 from 100.0 Reward 0.265517 regret 14718.625217064071 Action 18 epsilon 0.1
61.0 from 100.0 Reward 0.282514 regret 14956.176004038956 Action 1 epsilon 0.1
62.0 from 100.0 Reward 0.264563 regret 15196.53110610437 Action 20 epsilon 0.1
63.0 from 100.0 Reward 0.244905 regret 15443.1788033453 Action 30 epsilon 0.1
64.0 from 100.0 Reward 0.253141 regret 15689.561382324804 Action 28 epsilon 0.1
65.0 from 100.0 Reward 0.280907 regret 15936.33816671112 Action 1 epsilon 0.1
66.0 from 100.0 Reward 0.252963 regret 16184.997512211878 Action 12 epsilon 0.1
67.0 from 100.0 Reward 0.263593 regret 16430.009550903105 Action 4 epsilon 0.1
```

68.0 from 100.0 Reward 0.263151 regret 16682.067827140654 Action 4 epsilon 0.1
69.0 from 100.0 Reward 0.26011 regret 16931.850317902576 Action 20 epsilon 0.1
70.0 from 100.0 Reward 0.252091 regret 17176.63917301833 Action 31 epsilon 0.1
71.0 from 100.0 Reward 0.263453 regret 17422.3325088427 Action 20 epsilon 0.1
72.0 from 100.0 Reward 0.280566 regret 17681.97829673876 Action 1 epsilon 0.1
73.0 from 100.0 Reward 0.236546 regret 17924.269189572668 Action 10 epsilon 0.1
74.0 from 100.0 Reward 0.278461 regret 18167.388828847823 Action 1 epsilon 0.1
75.0 from 100.0 Reward 0.2648 regret 18412.549513169815 Action 18 epsilon 0.1
76.0 from 100.0 Reward 0.242264 regret 18653.512700391235 Action 29 epsilon 0.1
77.0 from 100.0 Reward 0.263616 regret 18904.520513247327 Action 22 epsilon 0.1
78.0 from 100.0 Reward 0.256334 regret 19152.515321751664 Action 17 epsilon 0.1
79.0 from 100.0 Reward 0.26335 regret 19393.157332747764 Action 22 epsilon 0.1
80.0 from 100.0 Reward 0.280312 regret 19646.599977527163 Action 1 epsilon 0.1
81.0 from 100.0 Reward 0.283506 regret 19888.939804959413 Action 1 epsilon 0.1
82.0 from 100.0 Reward 0.250401 regret 20138.450023181773 Action 31 epsilon 0.1
83.0 from 100.0 Reward 0.260423 regret 20383.16473588453 Action 3 epsilon 0.1
84.0 from 100.0 Reward 0.260759 regret 20628.84480889839 Action 7 epsilon 0.1
85.0 from 100.0 Reward 0.252782 regret 20871.145919129212 Action 28 epsilon 0.1
86.0 from 100.0 Reward 0.248244 regret 21119.31026395448 Action 27 epsilon 0.1
87.0 from 100.0 Reward 0.263058 regret 21366.669538574428 Action 8 epsilon 0.1
88.0 from 100.0 Reward 0.220687 regret 21609.173877950994 Action 25 epsilon 0.1
89.0 from 100.0 Reward 0.220273 regret 21854.629903463803 Action 9 epsilon 0.1
90.0 from 100.0 Reward 0.255888 regret 22097.5158900113 Action 32 epsilon 0.1
91.0 from 100.0 Reward 0.251238 regret 22338.71589808554 Action 32 epsilon 0.1
92.0 from 100.0 Reward 0.25807 regret 22585.355124176374 Action 17 epsilon 0.1
93.0 from 100.0 Reward 0.26057 regret 22829.87126334511 Action 23 epsilon 0.1
94.0 from 100.0 Reward 0.249234 regret 23070.710667441555 Action 14 epsilon 0.1
95.0 from 100.0 Reward 0.264451 regret 23315.772020019653 Action 4 epsilon 0.1
96.0 from 100.0 Reward 0.253941 regret 23561.242983846834 Action 32 epsilon 0.1
97.0 from 100.0 Reward 0.263231 regret 23815.514249908425 Action 20 epsilon 0.1
98.0 from 100.0 Reward 0.247011 regret 24066.883526699683 Action 30 epsilon 0.1
99.0 from 100.0 Reward 0.237857 regret 24318.94488108907 Action 29 epsilon 0.1
[0.2736068398716076, 0.2634570221067208, 0.25372013878308297,
0.2558847248715421, 0.2567502625039888, 0.2589363250792691, 0.25540743206435557,
0.2570235471923199, 0.20373490485614146, 0.23005943390467204,
0.24363719973448458, 0.24522836922494975, 0.22421872983785193,
0.23817757840655415, 0.2448460740882935, 0.24723680438787773,
0.2496503437521871, 0.2581421683257387, 0.25288925826277325,
0.25563302621967837, 0.2545303496252911, 0.25567066243236003,
0.2551873719460108, 0.254895018694694, 0.21463016300846374, 0.23120762607877388,
0.24220503604406307, 0.24339807245885267, 0.23003525983087897,
0.2396604134418471, 0.24455390373214256, 0.24788722901997687]



And again from 15000 to 20000 the groth of regret is linear and the average is not changing.

[7]:
```
def running_average(nums):
    result = []
    sum_so_far = 0
    for i, num in enumerate(nums):
        sum_so_far += num
        result.append(sum_so_far / (i + 1))
    return result

def main():
    # Number of steps
    timesteps = 100000
    # Import the bendit code
```

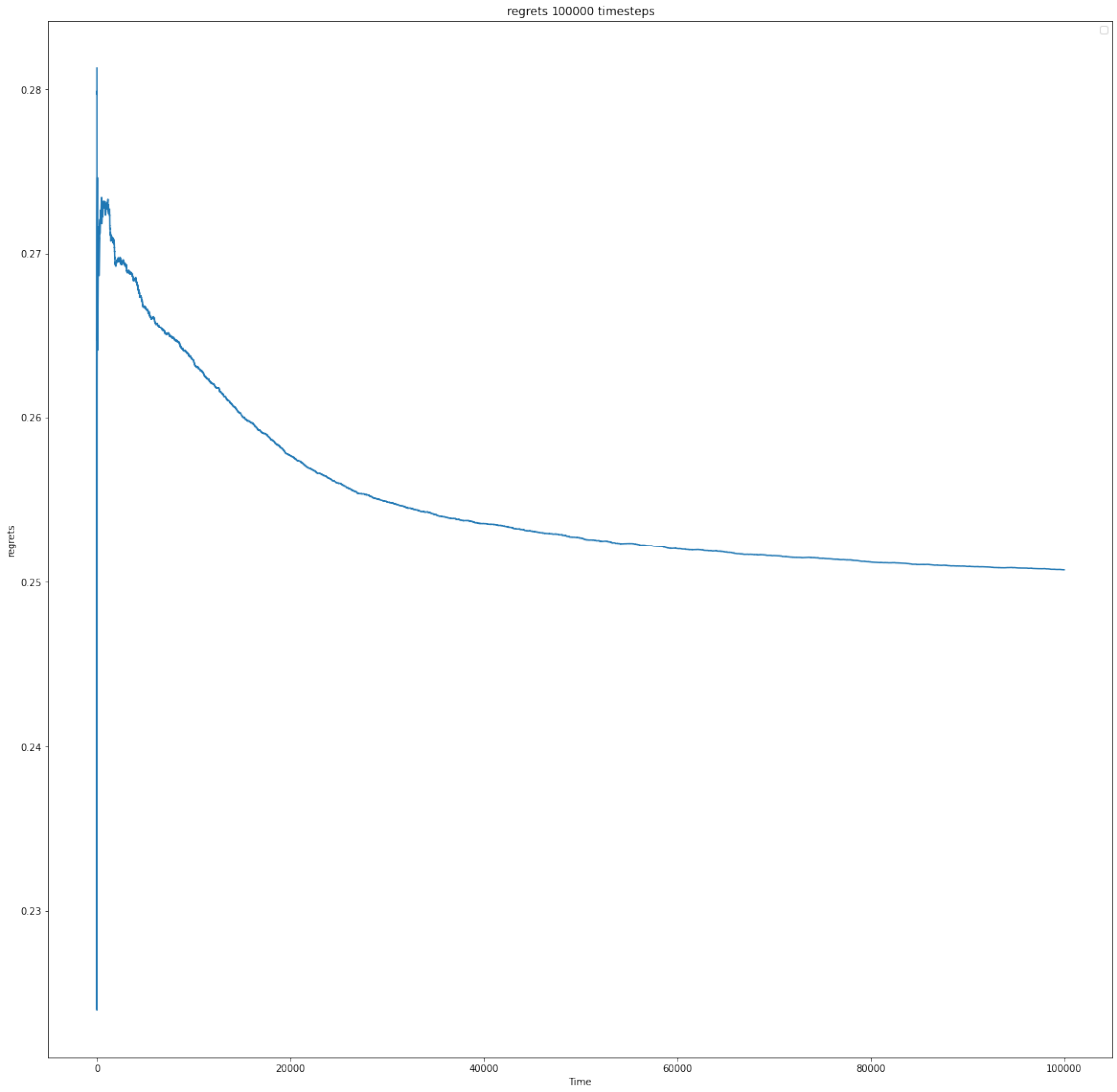
```
    b = bandit.Bandit()
    # Regrets
    regret = 0
    regrets = []
    rews = []
    # exploration probability
    # Greediness
    epsilon = 1
    decay = 0.9999
    min_epsilon = 0.1
    # Q Value initialization
    QVal = [0] * b.num_arms()
    # Counting each move or usage of arm (This can also be use as possibility)
    ActionCount = [0] * b.num_arms()
    # Delta for updating possibility
    delta = 0.0001
    # possibility of each arm
    bandits_p = []
    # Policy initialization
    policy = []
    for i in range(b.num_arms()):
        policy.append(1/b.num_arms())
        bandits_p.append([])

    # for time steps
    for t in range(timesteps):
        # Choose an arm
        a = 0
        if np.random.random() >= epsilon:
            a = random.randrange(b.num_arms())
        else:
            a = QVal.index(max(QVal))

        # Update the policy
        if policy[a] <= (1 - delta):
            policy[a] = policy[a] + delta
            for i in range(b.num_arms()):
                if i != a:
                    if policy[i] >= (delta/(b.num_arms() - 1)):
                        policy[i] = policy[i] - (delta/(b.num_arms() - 1))

        # Save the policy for plotting
        for i in range(b.num_arms()):
            bandits_p[i].append(policy[i])

        # Pull the arm, obtain a reward
        rew = b.trigger(a)
```

```
        ActionCount[a] = ActionCount[a] + 1

        # Update the Q-Value
        QVal[a] = QVal[a] = QVal[a] + ((1 / ActionCount[a]) * (rew - QVal[a]))

        # Save the regret
        regret += b.opt() - rew
        regrets.append(regret)
        rews.append(rew)

        # updating epsilon
        epsilon = max([min_epsilon, epsilon * decay])

        # print the data
        if (t % 10000) == 0:
            print(str(t / (timesteps/100)) + " from " + str(timesteps /␣
↪(timesteps/100)) + ' Reward', rew, 'regret', regret, "Action", a+1, "epsilon",␣
↪epsilon)


    print(QVal)
    # Plotting
    rews_avg = running_average(rews)
    plt.plot(rews_avg)
    plt.xlabel('Time')
    plt.ylabel('regrets')
    plt.title('regrets ' + str(timesteps) + " timesteps")
    plt.rcParams['figure.figsize'] = [20, 20]
    leg = plt.legend()
    plt.show()

if __name__ == '__main__':
    main()
```

```
0.0 from 100.0 Reward 0.281291 regret -0.007649719743300032 Action 1 epsilon
0.9999
10.0 from 100.0 Reward 0.24586 regret 101.62932072645638 Action 15 epsilon
0.3678242603283259
20.0 from 100.0 Reward 0.228262 regret 319.39506629662594 Action 25 epsilon
0.13530821730781062
30.0 from 100.0 Reward 0.269007 regret 562.100516251814 Action 2 epsilon 0.1
40.0 from 100.0 Reward 0.252338 regret 803.1652664936872 Action 28 epsilon 0.1
50.0 from 100.0 Reward 0.256907 regret 1047.8971712721877 Action 16 epsilon 0.1
60.0 from 100.0 Reward 0.241202 regret 1298.205689276885 Action 26 epsilon 0.1
70.0 from 100.0 Reward 0.26166 regret 1545.2900503339933 Action 20 epsilon 0.1
80.0 from 100.0 Reward 0.261084 regret 1796.4583081370718 Action 3 epsilon 0.1
90.0 from 100.0 Reward 0.249399 regret 2045.8080589642434 Action 11 epsilon 0.1
```

```
No artists with labels found to put in legend.  Note that artists whose label
start with an underscore are ignored when legend() is called with no argument.
```

```
[0.2729737367419295, 0.2647579782511348, 0.25298341077841485,
0.2565060382914768, 0.25604396150541037, 0.25833420976421934,
0.25560609725068956, 0.25619533003642364, 0.20363376414134535,
0.23123510481614232, 0.24314143364571783, 0.24511320584842272,
0.22550221759473868, 0.23722433116887406, 0.24437062276834903,
0.2479530839808754, 0.2502685964660381, 0.2588056644168999, 0.2521801401462407,
0.25725129002580094, 0.25354726069956157, 0.25511067308478863,
0.2551115826536298, 0.25480672917059033, 0.21453541670617057,
0.22985314255990721, 0.24137344278383807, 0.24295186536288244,
0.23133989644724356, 0.23922190580590072, 0.24495362808670842,
0.24642128543168615]
```



And the average reward graph.

[ ]: