In [1]:
```
## Anindya Prithvi (2020024)
## RL Assignment1

import numpy as np
import matplotlib.pyplot as plt
import time
from tqdm import trange
g_rand = np.random.default_rng(1337)
```

## Question 1

For the following sequence calculate the estimates of the expected reward for all arms.

1. Using Sample Mean
2. Using Exponential Weighted Average (alpha = 0.1)

*Initial Values*
Q1(arm1) = 5
Q1(arm2) = 8
Q1(arm3) = -6
Q1(arm4)= 0

*Sequence*

$$\begin{bmatrix} \text{Action} & 2 & 3 & 4 & 4 & 1 & 2 & 3 & 3 & 1 \\ \text{Reward} & -5 & 9 & 5 & 2 & -4 & 9 & 10 & 2 & 1 \end{bmatrix}$$

Is the sample mean affected by the choice of initial Q values?
Try to prove mathematically: The dependency of both 1 and 2 on the initial Q value.

## Answer 1

-- Note, $t$ has replaced the character $n$ here...otherwise everything follows convention. Do not confuse $t$ with global time, however it may be called the *arm's time*, $R_t$ being the arm's reward at $t$.

**Using Sample Mean**

| Time | Q(1) | Q(2) | Q(3) | Q(4) |
|---|---|---|---|---|
| 1 | 5 | 8 | -6 | 0 |
| 2 | 5 | **-5** | -6 | 0 |
| 3 | 5 | -5 | **9** | 0 |
| 4 | 5 | -5 | 9 | **5** |
| 5 | 5 | -5 | 9 | **3.5** |
| 6 | **-4** | -5 | 9 | 3.5 |
| 7 | -4 | **2** | 9 | 3.5 |
| 8 | -4 | 2 | **9.5** | 3.5 |
| 9 | -4 | 2 | **7** | 3.5 |
| 10 | **-1.5** | 2 | 7 | 3.5 |

The sample mean is unaffected by the initial choice. By definition, sample mean of an arm is the mean of the rewards obtained from that arm "so far".
To check the dependence:

$$\begin{aligned} Q_{t+1} &= Q_t + \frac{1}{t}(R_t - Q_t) \\ &= Q_t\left(1 - \frac{1}{t}\right) + \frac{1}{t}R_t \\ &= \left(Q_{t-1}\left(1 - \frac{1}{t-1}\right) + \frac{1}{t-1}R_{t-1}\right)\left(1 - \frac{1}{t}\right) + \frac{1}{t}R_t \\ &= Q_{t-1}\frac{t-2}{t} + \frac{R_{t-1} + R_t}{t} \\ &\dots \\ &= Q_1\frac{0}{t} + \frac{\sum_{i=1}^{t} R_i}{t} \end{aligned}$$

**Exponential Weighted Average**

$Q_{t+1} = Q_t + \alpha[R_t - Q_t]$

| Time | Q(1) | Q(2) | Q(3) | Q(4) |
|---|---|---|---|---|
| 1 | 5 | 8 | -6 | 0 |
| 2 | 5 | **6.7** | -6 | 0 |
| 3 | 5 | 6.7 | **-4.5** | 0 |
| 4 | 5 | 6.7 | -4.5 | **0.5** |
| 5 | 5 | 6.7 | -4.5 | **0.65** |
| 6 | **4.1** | 6.7 | -4.5 | 0.65 |
| 7 | -4 | **6.93** | -4.5 | 0.65 |
| 8 | -4 | 6.93 | **-3.05** | 0.65 |
| 9 | -4 | 6.93 | **-2.545** | 0.65 |
| 10 | **3.79** | 6.93 | -2.545 | 0.65 |

To check the dependence:

$$\begin{aligned} Q_{t+1} &= Q_t + \alpha[R_t - Q_t] \\ &= Q_t(1-\alpha) + \alpha R_t \\ &= (Q_{t-1}(1-\alpha) + \alpha R_{t-1})(1-\alpha) + \alpha R_t \\ &= (Q_{t-1}(1-\alpha)^2) + (1-\alpha)\alpha R_{t-1} + \alpha R_t \\ &\dots \\ &= Q_1(1-\alpha)^t + \sum_{i=1}^{t}(1-\alpha)^{t-i}\alpha R_t \end{aligned}$$

So, when $\alpha = 0$ it is directly proportional (better say "equal") to $Q_1$, but when $0 < \alpha < 1$ the value starts fading away. The closer to 0 $\alpha$ is, the faster it forgets (as time progresses).

## Question 2

Using epsilon-greedy, generate an episode for 1000 time steps.

Action Space: $\{1, 2, 3, 4\}$
Distribution of Rewards Associated with each Arm: $\{\mathcal{N}(0,1), \mathcal{N}(0,0.7), \mathcal{N}(0,0.2), \mathcal{N}(0.2,0.5)\}$

Use Sample mean as the estimate for e-greedy selection.
i) epsilon= 0.2
ii) epsilon= 0.8
iii) epsilon= 0
iv) epsilon= 1
v) Take epsilon to be a function of time, such that it decreases as t increases.

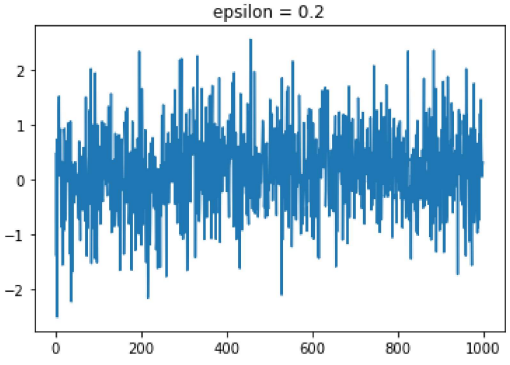
Plot the Rewards that you get at every time step in all 5 cases. What is the average reward for each epsilon?

Disambiguation: The term episode has been adopted across this assignment solely from the above reference. The real "episode" notion does not hold as we do not[acc book], in a sense, have a state.

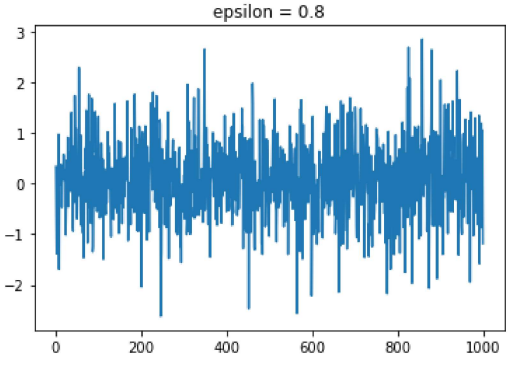
In [2]:
```
class Arm:
    def __init__(self, mu, sigma2, qval=0): #qval initial
        self.mu = mu
        self.sigma = np.sqrt(sigma2)
        self.used_n = 0
        self.qval = qval
#         self.rgen = np.random.default_rng(g_rand.integers(low = 0, high = 16637))
#         self.rgen = np.random.default_rng(int(mu*10+sigma2*10000))
        self.rgen = g_rand

    def get_reward(self) -> int:
        current_reward = self.rgen.standard_normal()*self.sigma + self.mu
#         print(f"I: {(self.mu, self.sigma)} have {self.qval} rewarded {current_reward}")
        self.update_qval(current_reward)
        return current_reward

    def update_qval(self, current_reward):
        self.used_n += 1
        n = self.used_n
        self.qval = self.qval*(n-1)/n + current_reward/n
```

In [3]:
```
class Bandit:
    def __init__(self, epsilon, arms, timesteps=1000, mode = "Normal"):
        self.epsilon = epsilon
        self.timesteps = timesteps
        self.arms = arms
        self.mode = mode

    def play_episode(self):
        reward_t = []
        for t in range(1, self.timesteps+1):
            arm = self.choose_arm(t)
            reward_t.append(self.play_step(arm))
        return reward_t

    def play_step(self, chosen_arm):
        reward = chosen_arm.get_reward()
        return reward

    def choose_arm(self, t):
        if self.mode=="UCB":
            c=2
            for arm in self.arms:
                if arm.used_n == 0:
                    return arm
            else:
                best_arm = np.argmax([a.qval+c*np.sqrt(np.log(t)/a.used_n) for a in self.arms])
                return self.arms[best_arm]

        get_rand = g_rand.uniform()
        chosen_arm = None
        if get_rand < self.epsilon(t):
            chosen_arm = g_rand.choice(self.arms)
        else:
            best_arm = np.argmax([a.qval for a in self.arms])
            chosen_arm = self.arms[best_arm]
        return chosen_arm
```

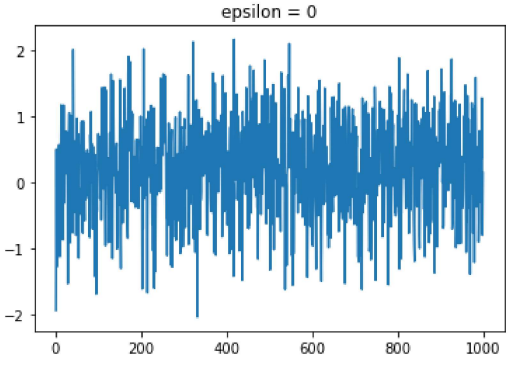
In [4]:
```
def reset_arms():
    return [Arm(0, 1), Arm(0, 0.7), Arm(0, 0.2), Arm(0.2, 0.5)]

timesteps = 1000
epsilons = [lambda a: 0.2, lambda a: 0.8, lambda a:0, lambda a: 1, lambda a: np.exp(0-a/70)]
for epsilon in epsilons:
    rew = Bandit(epsilon, reset_arms()).play_episode()
    plt.plot(np.arange(1,timesteps+1), rew)
    plt.title(f"epsilon = {epsilon(-5) if epsilon(-5)<=1 else 'dynamic'}")
    plt.show()
    print(f'The average is {np.average(rew)}')
```



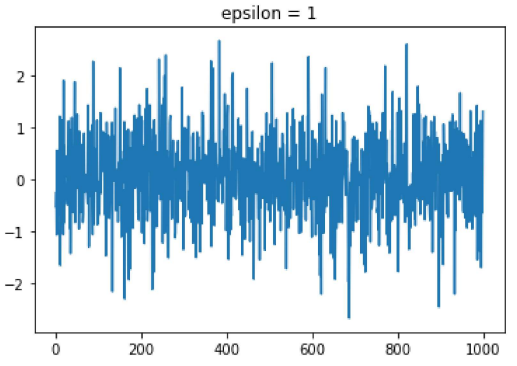

```
The average is 0.16166226257423655
```



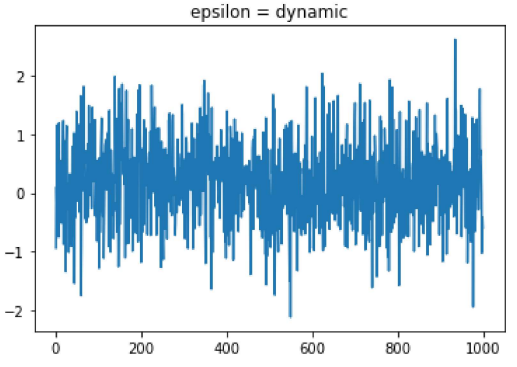

```
The average is 0.06367392467882071
```




```
The average is 0.20906456386735056
```




```
The average is 0.06540233894512829
```




```
The average is 0.212974834667418
```

Observation:

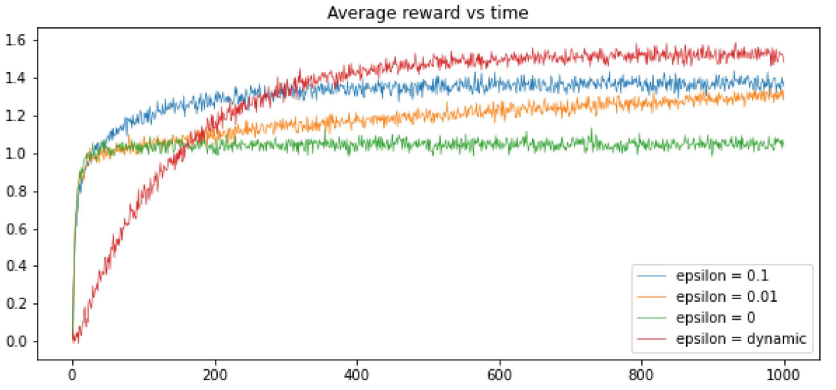
1. If timestamps are increased/episodes are played again and again, the decaying epsilon seems to win most of the times
2. If the starting few rewards are 'far' from the mean...the whole episode suffers

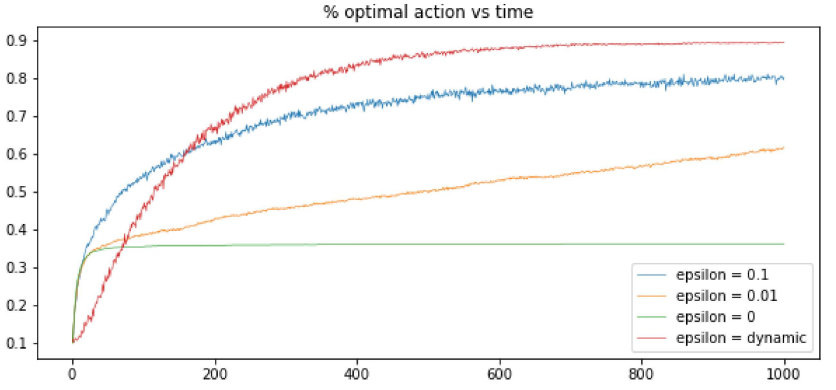
## Question 3

Read section 2.3 (The 10- armed Testbed) and generate figure 2.2 (Both the plots).

In [5]:
```
### Testbed

def testbed(variance = 1):
    runs = 2000
    arms_n = 10
    timesteps = 1000
    # epsilons = [lambda a: 1, lambda a: 0, lambda a: 0.1, lambda a: 0.01, lambda a: np.exp(0-a/100)
    epsilons = [lambda a: 0.1, lambda a: 0.01, lambda a: 0, lambda a: np.exp(-a/140)]
    # arm_reward_true = g_rand.normal(0,1,arms_n)
    # print(f"arm_rewards: {arm_reward_true}")

    plt.figure(figsize=(10,10))
    ax1 = plt.subplot(2,1,1)
    ax2 = plt.subplot(2,1,2)

    for epsilon in epsilons:
        all_rewards_avg = np.array([0 for i in range(timesteps)])
        all_optimalarm_avg = np.array([0 for i in range(timesteps)])
        for run in trange(1,runs+1):
            arm_reward_true = g_rand.standard_normal(arms_n)
            best_action_true = np.argmax(arm_reward_true)

            arms = [Arm(arm_reward_true[i], variance) for i in range(arms_n)]
            bandit = Bandit(epsilon, arms, timesteps)

            reward_t = []
            for t in range(1, timesteps+1):
                arm = bandit.choose_arm(t)
                if arm==arms[best_action_true]:
                    all_optimalarm_avg[t-1] += 1
                reward_t.append(bandit.play_step(arm))

            all_rewards_avg = all_rewards_avg*(run-1)/run + np.array(reward_t)/run
        all_optimalarm_avg = all_optimalarm_avg/runs
        elabel = f"epsilon = {epsilon(-5) if epsilon(-5)<=1 else 'dynamic'}"
        ax1.plot(np.arange(1,timesteps+1), all_rewards_avg, label=elabel, linewidth=0.6)
        ax2.plot(np.arange(1,timesteps+1), all_optimalarm_avg, label=elabel, linewidth=0.6)

    ax1.legend()
    ax1.set_title("Average reward vs time")
    ax2.legend()
    ax2.set_title("% optimal action vs time")

    plt.subplots_adjust(hspace=0.3)
#     plt.savefig("plot_2_2.png", dpi=3000)
    plt.show()

testbed()
```

```
100%|██████████| 2000/2000 [0
0:31<00:00, 64.44it/s]
100%|██████████| 2000/2000 [0
0:26<00:00, 76.15it/s]
100%|██████████| 2000/2000 [0
0:25<00:00, 79.32it/s]
100%|██████████| 2000/2000 [0
0:32<00:00, 61.07it/s]
```



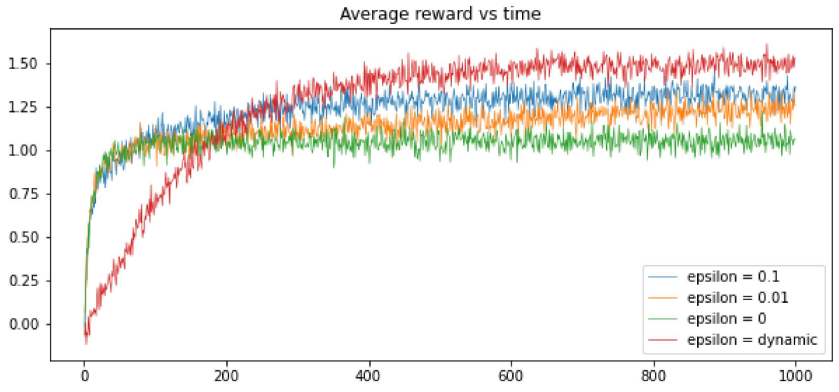



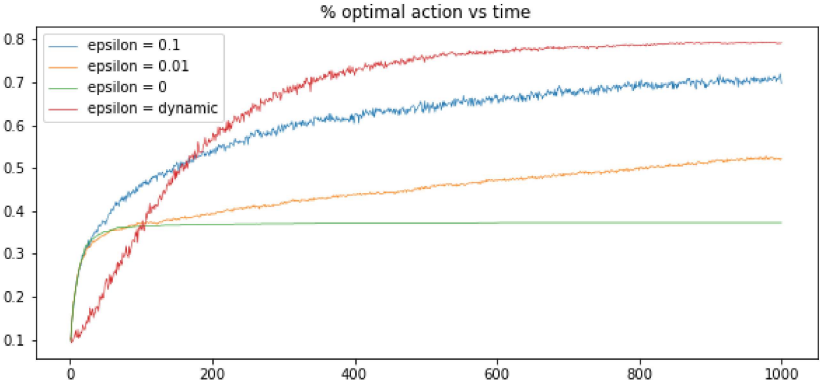

## Question 4

Testbed with variance 4

In [6]:
```
testbed(4)
```

```
100%|██████████| 2000/2000 [0
0:30<00:00, 66.45it/s]
100%|██████████| 2000/2000 [0
0:25<00:00, 78.62it/s]
100%|██████████| 2000/2000 [0
0:25<00:00, 77.98it/s]
100%|██████████| 2000/2000 [0
0:33<00:00, 59.20it/s]
```



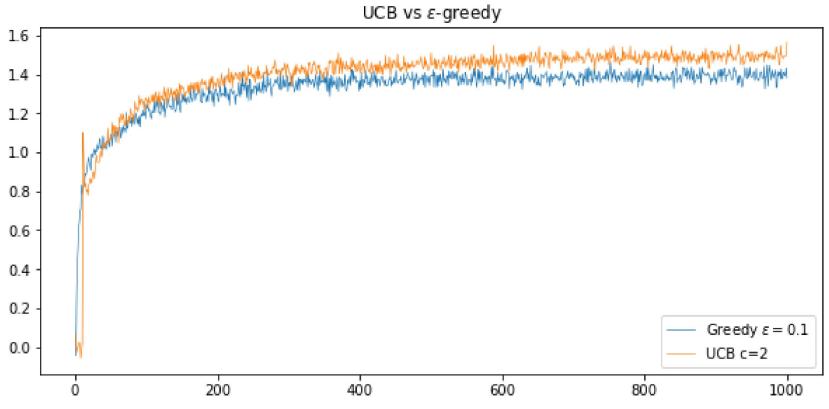




## Question 5

Upper confidence bound action selection

In [7]:
```
### Testbed

def testbed_UCB():
    runs = 2000
    arms_n = 10
    timesteps = 1000
    # epsilons = [lambda a: 1, lambda a: 0, lambda a: 0.1, lambda a: 0.01, lambda a: np.exp(0-a/100)
    # arm_reward_true = g_rand.normal(0,1,arms_n)
    # print(f"arm_rewards: {arm_reward_true}")

    plt.figure(figsize=(10,10))
    ax1 = plt.subplot(2,1,1)

    epsilon = lambda x: 0.1

    all_rewards_avg = np.array([0 for i in range(timesteps)])
    for run in trange(1, runs+1):
        arm_reward_true = g_rand.standard_normal(arms_n)
        best_action_true = np.argmax(arm_reward_true)

        arms = [Arm(arm_reward_true[i], 1) for i in range(arms_n)]
        bandit = Bandit(epsilon, arms, timesteps)

        reward_t = bandit.play_episode()
        all_rewards_avg = all_rewards_avg*(run-1)/run + np.array(reward_t)/run

    ax1.plot(np.arange(1,timesteps+1), all_rewards_avg, label="Greedy $\epsilon=0.1$", linewidth=0.6

    all_rewards_avg = np.array([0 for i in range(timesteps)])
    for run in trange(1, runs+1):
        arm_reward_true = g_rand.standard_normal(arms_n)
        best_action_true = np.argmax(arm_reward_true)

        arms = [Arm(arm_reward_true[i], 1) for i in range(arms_n)]
        bandit = Bandit(epsilon, arms, timesteps, "UCB")

        reward_t = bandit.play_episode()
        all_rewards_avg = all_rewards_avg*(run-1)/run + np.array(reward_t)/run

    ax1.plot(np.arange(1,timesteps+1), all_rewards_avg, label="UCB c=2", linewidth=0.6)

    ax1.legend()
    ax1.set_title("UCB vs $\epsilon$-greedy")

#     plt.subplots_adjust(hspace=0.3)
#     plt.savefig("plot_2_2.png", dpi=3000)
    plt.show()

testbed_UCB()
```

```
100%|| 2000/2000 [0
0:30<00:00, 66.11it/s]
100%|| 2000/2000 [0
1:08<00:00, 29.20it/s]
```




[Ex2.8] The sudden spike observed can be attributed to the fact that at the 11th step all arms have been "definitely" explored once, also, note that *the term* added after Q_value is constant for each arm, this straight away means that the

best arm will now be chosen greedily only based on the one time that each arm had been explored. Now since we have 2000 runs, the chances of the reward from the chosen arm is dense near the mean (Gaussian distro).
Now, right after this step, we observe a fall. This is because the best arm has now been used twice (observing/averaging over 2000 runs), this significantly impacts the term because of $N_t(a)$ in the denominator. When $c = 1$ this is less pronounced because it is just trying to reduce the effect of *the term*.

QED?

-- Note: All references to "The Term" are $\left(c\sqrt{\frac{ln(t)}{N_t(a)}}\right)$

## Question 6

Bandit Gradient Algorithm

In [8]:
```
class BanditGrad:
    def __init__(self, alpha, mean_shift=4, arms_n=10, based=True):
        self.H_array = np.zeros(arms_n)
        arm_reward_true = g_rand.standard_normal(arms_n)
        self.arms = [Arm(arm_reward_true[i]+mean_shift, 1) for i in range(arms_n)]
        self.alpha = alpha
        self.Rtbar = 0
        self.time = 1
        self.based = based
        self.arms_n = arms_n
        self.best_arm = np.argmax(arm_reward_true)

    def pivec(self):
        den = np.sum(np.e**self.H_array)
        return np.e**self.H_array/den

    def choose_arm(self):
        p=self.pivec()
        return np.random.choice(self.arms_n, p=p)

    def update_H(self):
        arm_i = self.choose_arm()
        Rt = self.arms[arm_i].get_reward()
        if self.based:
            n = self.time
            self.Rtbar = ((n-1)/n)*self.Rtbar + (1/n)*Rt
        self.time+=1
        iden = np.zeros(self.arms_n)
        iden[arm_i]=1
        self.H_array = self.H_array+self.alpha*(Rt-self.Rtbar)*(iden-self.pivec())
        if arm_i == self.best_arm:
            return 1
        else:
            return 0

    def play_episode(self,timesteps = 1000):
        while self.time<=timesteps:
            self.update_H()
```

In [9]:
```
runs = 2000
timesteps = 1000

for based in [True, False]:
    for alpha in [0.1, 0.4]:
        optim = np.zeros(timesteps+1)
        for _ in trange(runs):
            bandit = BanditGrad(alpha, based=based)
            for tt in range(1,timesteps+1):
                optim[tt] += bandit.update_H()
        optim/=runs
        plt.plot(np.arange(0, timesteps+1), optim, label = f"$a = {alpha}${', baselined' if based el

plt.legend()
```

```
100%|| 2000/2000 [0
1:33<00:00, 21.33it/s]
100%|| 2000/2000 [0
1:33<00:00, 21.31it/s]
100%|| 2000/2000 [0
1:32<00:00, 21.61it/s]
100%|| 2000/2000 [0
1:49<00:00, 18.31it/s]
```

Out[9]: <matplotlib.legend.Legend at 0x1d83771f6a0>