Task 1

a)

We shall use proof by induction

- **Base**

Since: $\mu_i^{(new)} = \mu_i + a(x_j - \mu_i)$ is the update rule, then

for $n=1$: $\mu_i^{(1)} = \mu_i^{(0)} + a(x_1' - \mu_i^{(0)}) \Rightarrow \mu_i^{(1)} = \mu_i^{(0)} + ax_1' - a\mu_i^{(0)} \Rightarrow$

$\Rightarrow \mu_i^{(1)} = (1-a)^1 \mu_i^{(0)} + a(1-a)^0 x_1'$ //

- **Inductive step**

Let's assume that $\mu_i^{(n)} = (1-a)^n \mu_i^{(0)} + a\sum_{k=1}^{n} (1-a)^{n-k} x_k'$ (1)

holds, we need to prove that the above holds for $n+1$ too

~~$\mu_i^{(n+1)} = \mu_i^{(n)} + a(x_{n+1}' - \mu_i^{(n)}) = (1-a)\mu_i^{(n)} + a\sum_{k=1}^{n} x_k'$~~

- for $n+1$ the update: $\mu_i^{(n+1)} = \mu_i^{(n)} + a(x_{n+1}' - \mu_i^{(n)}) =$

$= (1-a)\mu_i^{(n)} + ax_{n+1}' \overset{(1)}{=}$

$\Rightarrow \mu_i^{(n+1)} = (1-a)\left[(1-a)^n \mu_i^{(0)} + a\sum_{k=1}^{n} (1-a)^{n-k} x_k'\right] + ax_{n+1}' \Rightarrow$

$\Rightarrow \mu_i^{(n+1)} = (1-a)^{n+1} \mu_i^{(0)} + a\sum_{k=1}^{n} (1-a)^{n+1-k} x_k' + ax_{n+1}' \Rightarrow$

$\Rightarrow \mu_i^{(n+1)} = (1-a)^{n+1} \mu_i^{(0)} + a\sum_{k=1}^{n+1} (1-a)^{n+1-k} x_k'$ // (proved.)

b)

$\mu_i^{(n)} = \frac{1}{n}\sum_{k=1}^{n} x_k' = \frac{1}{n}\left(\sum_{k=1}^{n-1} x_k' + x_n'\right) = \frac{n-1}{n}\mu_i^{(n-1)} + \frac{1}{n}x_n' =$

$= \mu_i^{(n-1)} + \frac{1}{n}(x_n' - \mu_i^{(n-1)})$, so the update rule would be $\mu_i^{new} = \mu_i + \frac{1}{n}(x_n' - \mu_i)$, where n the number of points assigned to the cluster up to this point

c) · In the first rule, α stays the same. What this does is in essense overide (slowly) the contribution of the previous points, which sounds great for online learning.

· In the second rule, the step size slowly decreases, effectively making it so that we don't simply forget the contribution of the previous points but rather consider them all equally. The mean (μ) will converge to the average of the points, making it perfect for any scenario where we know all the points beforehand

Task 2

a) Let's assume that we have already queried k points (where $k < n$). We now know the true labels of those points, however since we know nothing about the intervals $[a,b]$ and $[c,d]$ or about the distribution of the n points, there is no way we can be sure that our guess for the rest of the $n-k$ points will be correct. (think of this as a 2 man game where you make the predictions and your opponent chooses a, b, c, d. In this case, you will always lose, unless you query all n points).

b) The idea is the following:

1) Sort the samples

2) Starting from left, query all the samples until you find the first "1", then skip $m-1$ points by querying like in the beginning

3) Search for the next "-1" and repeat the skipping process

4) Do the same for the next "1"

5) If you find a "-1" after the last "-1" area then stop, since you know those will all be -1

· Overall we perform at most $n - 3(m-1)$ queries

Task 3

a) We know that: $E[R_T] = T\mu^* - E[\sum_{t=1}^{T} y_t]$ ①

Rewrite:

$$\sum_{t=1}^{T} y_t = \sum_{i=1}^{k} \sum_{j=1}^{n_i^T} y_{i,j} \xrightarrow{\text{then}} E[\sum_{i=1}^{k} \sum_{j=1}^{n_i^T} y_{i,j}] = \sum_{i=1}^{k} E[\sum_{j=1}^{n_i^T} y_{i,j}]$$

$$\xrightarrow[\text{rule}]{\text{tower}} \sum_{i=1}^{k} E[E[\sum_{j=1}^{n_i^T} y_{i,j} \mid n_T i]] = \sum_{i=1}^{k} E[\sum_{j=1}^{n_i^T} \mu_i] = \sum_{i=1}^{k} E[n_i^T \mu_i] =$$

$$= \sum_{i=1}^{k} \mu_i \cdot E[n_i^T]$$

$$① = T\mu^* - \sum_{i=1}^{k} \mu_i E[n_i^T] \xrightarrow{T = \sum_{i=1}^{k} E[n_T i]} \sum_{i=1}^{k} E[n_i^T](\mu^* - \mu_i) =$$

$$= \sum_{i=1}^{k} E[n_i^T] \Delta_i \quad //$$

b)

Hoeffding's inequality: $P(|\mu_i - \hat{\mu}_i^t| \geq \epsilon) \leq 2\exp(-2\epsilon^2 n_i^t) \Rightarrow$

$\epsilon = \sqrt{\frac{\log(T)}{n_i^t}} \Rightarrow P\left(|\mu_i - \hat{\mu}_i^t| \geq \sqrt{\frac{\log(T)}{n_i^t}}\right) \leq 2\exp(-2\log(T)) \Rightarrow$

$\Rightarrow P\left(|\mu_i - \hat{\mu}_i^t| \geq \sqrt{\frac{\log(T)}{n_i^t}}\right) \leq \frac{2}{T^2} \Rightarrow$

$P(\ldots = \sqrt{\frac{\log(T)}{n_i^t}}) = 0$

$\Rightarrow P\left(|\mu_i - \hat{\mu}_i^t| > \sqrt{\frac{\log(T)}{n_i^t}}\right) \leq \frac{2}{T^2} \Rightarrow P(\mu_i \notin C_i^t) \leq \frac{2}{T^2}$

So we ignore the $=$

c)

When $\mu_i \in C_i^t$: $\hat{\mu}_i \in \left[\hat{\mu}_i^t - \sqrt{\frac{\log(T)}{n_i^t}}, \hat{\mu}_i^t + \sqrt{\frac{\log(T)}{n_i^t}}\right]$

- Since $\mu_i \in C_i^t$ and $\mu_i^* \in C_i^t$: (the above, but also the inverse!)

So: $\hat{\mu}_i^t \leq \mu_i + \sqrt{\frac{\log(T)}{n_i^t}}$ ① and $\hat{\mu}_i^t \geq \mu_i - \sqrt{\frac{\log(T)}{n_i^t}}$ (rather than $\mu_i \sim \hat{\mu}_i^t$)

$\hat{\mu}_{i^*}^t \leq \mu^* + \sqrt{\frac{\log(T)}{n_{i^*}^t}}$ and $\hat{\mu}_{i^*}^t \geq \mu^* - \sqrt{\frac{\log(T)}{n_{i^*}^t}}$ ②

also we know that in UCB i would be chosen only if:

$$\hat{\mu}_i^t + \sqrt{\frac{\log(T)}{n_i^t}} \geq \hat{\mu}_{i^*}^t + \sqrt{\frac{\log(T)}{n_{i^*}^t}} \quad (3)$$

we can substitute (2) on (3) and (2) on (3) (we will have an even bigger value left with (3) and an even smaller on the right with (2), so it's fine. Then:

$$\mu_i + 2\sqrt{\frac{\log(T)}{n_i^t}} \geq \mu^* \Rightarrow 2\sqrt{\frac{\log(T)}{n_i^t}} \geq \Delta_i \Rightarrow \frac{4\log(T)}{\Delta_i^2} \geq n_i^t$$

but! accounting for the fact that we try all arms atleast once at the start of the algorithm: $n_i^t \leq \frac{4\log(T)}{\Delta_i^2} + 1$

d)

Task 4

a)

a) Streaming model:

In the streaming model, data can arrive through multiple streams, continously. The speed at which we intake data is not something we can control, however this distinguises this model from a database management system, which doesn't need to worry about data passing by too fast and thus being lost. The incoming data can be stored in memory and thus become accessable later on for queries, however the memory is greatly limited and thus can't hold all the incoming info. A solution to this exists in the form of a huge storage called the archive, however while we can save way more data in the archive, access to it for reads is extremely slow and thus rarely attempted. The last important part of this model are the queries. There are 2 types, standing queries and ad-hoc queries. The former are comprised of queries that run constantly and can produce answers whenever it's needed. The latter are queries that are asked once about the current state of a stream / streams. The idea

is that you can expect what ad-hoc queries you will receive, thus being able to save summaries of the incoming data, or use sliding windows for all streams, which will hold the k most recent data, or data that arrived after a certain point t, so the ad-hoc queries can use this data to answer the needed requests.

b) ~~Counting 1s~~ * The bucket size starts from 1 always!

The idea is as follows:

You want to count how many 1s approximately showed in the last k bits, $k \leq N$ (N = window size). In that case we will use buckets which will hold 2 pieces of info, the time the last 1 bit was added into the bucket and the size of the bucket. It's important to note that we can only have up to 2 buckets of the same size at any one time, if the number goes to 3 we merge the two oldest buckets and get a bucket with double the size and a time stamp of the newest 1 bit of the two. (in this case we use a time stamp system where the larger the number, the newer it is!)

Lastly we calculate the 0s like this:

- if timestamp − size $\geq$ current_time − k then count its size as all 0s
- else cant half its size

sum for all the buckets and you have a good result!

(also you only end up using O(log N) space! rather than O(N))

Example say we are at t=7

k=5 , 1 0 1 1 0 1 1 0

t=1 → [[1, 1]]

t=2 → [[0, 1]]

t=3 → [[3, 1], [0, 1]]

t=4 → [[4, 1], [3, 2]]

t=5 → [[4, 1], [3, 2]]

t=6 → [[6, 1], [4, 1], [3, 2]]

t=7 → [[7, 1], [6, 2], [3, 2]]

full full half

So we estimate 4 1's, when in reality we have 3, not too bad!

b)

a) BDMO algorithm

This is a stream-clustering algorithm that builds upon the counting-1s method we explained and is thus able to maintain clusters efficiently by using summaries (like the number of points in a cluster, the centroid and other info needed to perform merging of clusters). Here the bucket size doesn't need to start from 1.

Algo

- Just like before you create the buckets and merge them when 3 buckets of the same size appear.
- Inside each bucket you follow a specific clustering strategy (so you can choose to cluster the points that enter) and each time a merge happens you must consider whether or not a merge of clusters should happen, in which case the summary would have to be updated quite a bit.

Finally if a request about the clustering of the last m points is made, then you can be sure that by choosing the smallest covering set of buckets for the last m points, we can be sure that we will use $\leq 2m$ points, sacrificing some accuracy for the efficiency we presented in the previous question.

b) Parallelism

Given a large set of points for which we want to compute the centroids, parallelism could be exploited through map-reduce! Each mapper gets a subset of points and yields a key-value pair with key = 1 always and value = description of a cluster. Since the keys are all the same, only one reducer

can be used. This reducer simply gets the descriptions of the clusters and tries to merge them, outputting the final clusters.