Q1

(1) The advantage is that data warehouse has fast Query Response which could save users' time. In addition, the data warehouse is usually requires less storage space that database.

The disadvantage is that selective Materialization Decision Problem is NP-hard, the main objective is either the minization of a cost function or a constraint, a constraint can be user oriented or system oriented.

(2) According to the "Selection of views to Materialize in a Data warehouse" Himanshu Gupta, Inner-Level Greedy Algorithm (the paper I searched online)

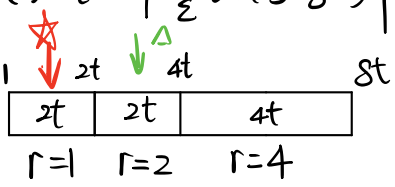
Given G, a view graph with indexes, and S, the space constraint.

```
1  Begin
2    M = ϕ        /* M = Set of structures selected so far */
3    while ( S(M) < S)
4       C = ϕ ;  /* Best set containing a view and some of its indexes
5       for each view Vi not in M
6          IG = {Vi} ; /* IG = set of Vi and some of its indexes in a greedy manner
7          while(S(IG)<S)    /* Construct IG */
8             Let Iic be the index of Vi whose benefit per unit space
9             IG = IG U Iic
10         end while
11         if ( B(IG,M)/ S(IG) > B(C,M)/|C| /or C = ϕ /* B(IG,M) = Gain( M U IG ,S), which is the same meaning in lecture*/
12            C = IG;
13      end for
14      M = M U C
15   end while
16   return M
```

Each stage can be thought of as consisting of two phrases. In the first phase, for each view V_i we can construct a set IG_i which initially contains only the view. Then one by one its indexes are added to IG_i in the order of their incremental benefits until the benefit per unit space of IG_i with respect to M, the set of structures selected till this stage, reach its maximum. That IG_i having the maximum benefit per unit space with respect to M is chosen as C. In the second phase, an index whose benefit per unit space is the maximum with respect to M is selected. The benefit per unit space of the selected index is compared with that of C, and the better one is selected for addition to M. (After done Q1, my mindset changed a little bit, just feel like I am Lagranged)

Q2

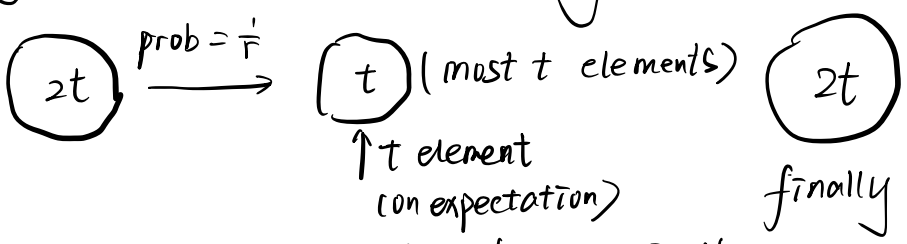
(a) $t = \lceil \frac{1}{\varepsilon} \ln(s^{-1}\delta^{-1}) \rceil$



① The major rational behind Sticky Sampling Algorithm is that suppose we are in the stage of ~~the~~ red arrow point, principle is to keep the number of entries in the memory at most $2t$ (on average)

$r=8$

② when we cross the boundary to ↓Δ



it will keep the number of entries in the main memory and most $2t$ on average at any time.

③ Then to prove, the number of elements with frequency at least s in no more $\frac{1}{s}$, $\frac{1}{s} \cdot e^{-\varepsilon t} = \frac{e^{-\varepsilon t}}{s}$

So the probility of failure should be at most δ.

After make sure of $2t$, when sampling rate $r \geq 2$. $N = rt + rt'$ $t \in [1, t)$, after make sure of N, we need to know the length of sequence of unsuccessful coin tosses the probility exceeds εN is at most $(1-\frac{1}{r})^{\varepsilon N} \leq (1-\frac{t}{N})^{-\varepsilon N} < e^{-\varepsilon t}$

(b) Differences: $(1-\delta)$

1. sticky: probabilistic algorithm storage is $\lceil 2/\varepsilon \ln(s^{-1}\delta^{-1}) \rceil$
2. Lossy: deterministic algorithm. storge is $\lceil 1/\varepsilon \log(\varepsilon N) \rceil$

(C) Lossy needs an error parameter ε to deduce the memory storage

space-saving needs the memory parament M to deduce error. and with the same size of memory space-saving algorithm has better performance to Lossy Counting algorithm.

Q3

(a) $\varepsilon N = 2 \quad (s-\varepsilon)N = 2$

Algo 1: No, condition 3 not satisfied

Algo 2: Yes, all conditions satisfied

Algo 3: Yes all conditions satisfied

(b)

(i)

$$\text{memory} = m \cdot \frac{1}{\varepsilon} \cdot \log \varepsilon N$$

(ii) nicely bounded

Q4

(a)(i) Set memory size = M initially.

$$M = \underbrace{\text{previous data}}_{①} + \underbrace{\text{incoming data}}_{②}$$

size $= M - B$ size $= B$

Then we divide ① into four parts of same size. Each part is associate with a summary. In each part, we do what we do in the original space-saving algorithm.

Given a set of data points. S. Spacesaving(S) = the stored summary.

Set X = spacesaving(S). X contains two components. ① X.E = a list of entries each in form of (e, f, Δ). e is the item number, f is the frequency of the item, Δ is the maximum possible error inf. ② X.p = variable p_e used in the original spacesaving algorithm.

And the output is

$f(e, B_i)$ = the count of element e stored in the summary of part B_i

if the entry (e, f, Δ) exists in the summary X for B_i, $f(e, B_i) = f$, otherwise $f(e, B_i) = 0$

$\Delta(e, B_i)$ = the Δ value of element e stored in part B_i

In the same case

if the entry (e,f,Δ) exists in the summary X for B_i, $\Delta(e,B_i)=\Delta$, otherwise $\Delta(e,B_i)=0$

For each item e, calculate $f_e=\sum_{i=1}^{4} f(e,B_i)$ $\Delta_e=\sum_{i=1}^{4}\Delta(e,B_i)$

Finally, we can get a list of items where $f_e+\Delta_e \geq s\cdot 4\cdot B$

(ii) In (i) the size of ① is $M-B$, after set it into 4 parts of the same size, some the size of each part is $\frac{(M-B)}{4}$.

Consider a part contains X, we can assume that X.p or X.E occupies 1 byte so the remain is $\frac{M-B}{4}-1$

assuming m = the size of a single entry in form of (e,f,Δ)

So the numer of the entries $=(\frac{M-B}{4}-1)/m$

In each part, the greatest error $=\frac{1}{(\frac{M-B}{4}-1)/m}=\frac{4m}{M-B-4}$ every fraction

overall greatest error in count $=\frac{4m}{M-B-4}\times 4B=\frac{16mB}{M-B-4}$

overall greatest error in fraction $=\frac{16mB}{M-B-4}\times\frac{1}{4B}=\frac{4m}{M-B-4}$

(b) (i) Set memory size = M initially.

M = previous data + incoming data

① size $=M-B$ ② size $=B$

Then we divide ① into four parts of same size. Each part is associate with a summary. In each part, we do what we do in the original space-saving algorithm.

Given a set of data points. S. Spacesaving(S) = the stored summary.

Set X = spacesaving(S). X contains two components. ① X.E = a list of entries each in form of (e,f,Δ). e is the item number, f is the frequency of the item, Δ is the maximum possible error in f. ② X.p = variable p_e used in the original spacesaving algorithm.

And the output is

$f(e,B_i)$ = the count of element e stored in the summary of part B_i

if the entry (e,f,Δ) exists in the summary X for B_i, $f(e,B_i)=f$, otherwise $f(e,B_i)=0$

assume I to be the incomplete batch. $f(e,I)$ = count of element e in the imcomplete batch I

for B_1, for each item e, $Ye=\min\{f(e,B_1)+\Delta(e,B_1), 0.5B\}$

for B_2, B_3, B_4, I

$$f_e=\sum_{i=2}^{4} f(e,B_i)+f(e,I);\quad \Delta_e=\sum_{i=2}^{4}\Delta(e,B_i)+0=\sum_{i=2}^{4}\Delta(e,B_i)$$

Finally, we can get a list of items where $f_e + \Delta e + Y_e \geq 5.4 \cdot B$

(ii) for B_2, B_3, B_4 the greatest error $= \frac{4m}{m-B-4}$ in fraction

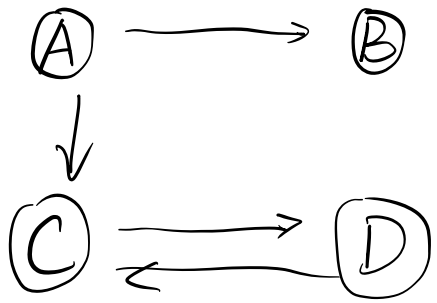
for I, the greatest error $= 0$

for B_1, the greatest error $= 0.5$ in fraction

So the overall greatest error in fraction $= \frac{1}{4}\left(\frac{4m}{m-B-4} \times 3 + 0 + 0.5\right)$

$= \frac{3m}{m-B-4} + \frac{1}{8}$

Q5



(1) $Auth(A) = 0$

$Auth(B) = Hub(A)$

$Auth(C) = Hub(A) + Hub(D)$

$Auth(D) = Hub(C)$

$Hub(A) = Auth(B) + Auth(C)$

$Hub(B) = 0$

$Hub(C) = Auth(D)$

$Hub(D) = Auth(C)$

$Auth(A)\_normalized = 0$

$Auth(B)\_normalized \approx 0.382$

$Auth(C)\_normalized \approx 0.618$

$Auth(D)\_normalized \approx 0$

$Hub(A)\_normalized \approx 0.618$

$Hub(B)\_normalized = 0$

$Hub(C)\_normalized \approx 0$

$Hub(D)\_normalized \approx 0.382$

(b) A C B D