#1

Input $G = (V, E)$

max degree $= \Delta$

sub-linear regime

$\Theta(1)$-approx maximum matching of G $\quad O(\log \Delta)$ rounds

1. $M = \phi$

2. For $i = 1 \rightarrow \log \Delta$ do

3. Let A be the set of edges of G s.t. each edge is sampled with probability $\frac{2^i}{2\Delta}$ and independently of other edges.

4. Let $\bar{A}$ be the subset of edges of A that do not share endpoints with any other edge in A

5. $M = M \cup \bar{A}$

6. Remove from G all verticies with endpoints in $\bar{A}$ and all verticies with degree at least $\frac{\Delta}{2^i}$

7. return M

Correctness :- Algorithm above outputs a matching as the set $\bar{A}$ added to M on line 5 consists of edges which are matching themselves, and after an edge is added to M its endpoints are removed from G on line 6.

Round Complexity :- The algorithm runs in $O(\log \Delta)$ rounds as in every round, we only sample edges, construct the set $\bar{A}$ and then remove some edges, all that can be done in parallel

Approximation :-

Each round $\quad e = (u, v)$

$$e \text{ sample chance } = \frac{2^i}{2\Delta}$$

Let X_e be an indicator random variable $= 1$ if edge e is added to M in iteration i

e will be in $\bar{A}$ if none of its verticies u and v have other edges in A

(u, v) degree $\leq \frac{\Delta}{2^i}$

$$\text{Chance of edge to be in } \bar{A} = \frac{2^i}{2\Delta}\left(1 - \frac{2^i}{2\Delta}\right)^{\frac{2\Delta}{2^{i-1}}}$$

$$= \frac{2^i}{2\Delta}\left(1 - \frac{2^i}{2\Delta}\right)^{\frac{4\Delta}{2^i}}$$

$$\geq \frac{2^i}{2\Delta} \cdot e^{-4}$$

Let R_i be # of verticies removed in line 6 with degree atleast $\frac{\Delta}{2^i}$

Let $\bar{A}_i$ be edges in iteration i added to M.

$$\mathbb{E}\left[|\bar{A}_i|\right] = \mathbb{E}\left[\sum_e X_e\right]$$

$$\geq \sum_{v \in R_i} \left(\frac{1}{2} \sum_{e=\{u,v\}} \mathbb{E}[X_e]\right)$$

$$\geq \sum_{v \in R_i} \left(\frac{1}{2} \sum_{e=\{u,v\}} \frac{2^i}{2\Delta} \cdot e^{-4}\right)$$

$$= \sum_{v \in R_i} \left(\frac{1}{2} d(v) \frac{2^i}{2\Delta} \cdot e^{-4}\right)$$

$$\geq \sum_{v \in R_i} \left(\frac{1}{2} \cdot \frac{\cancel{\Delta}}{\cancel{2^i}} \cdot \frac{\cancel{2^i}}{2\cancel{\Delta}} \cdot e^{-4}\right)$$

$$= |R_i| \frac{e^{-4}}{4}$$

The final matching size is in expectation at least $e^{-4}/4$ function of number of verticies removed on line 6.

At end, the matching of size $\leq |M| + |\sum_i R_i|$ in G which is maximal

$|M| \leq 2(1+4e^{4})$ times smaller than a maximum matching in G

2.

stream : $e_1, \ldots, e_n$

$e_i = O(B)$ bits

e_i has weight $w_i \in [1, n]$

i sample prob $\sum_{j=1}^{n} \frac{w_i}{w_j}$

memory $= O(B + \text{poly} \log n)$

Input $G = (V, E, w)$

1. tempWeight $= w_1$
2. $e = e_1$
3. for $i = 2 \longrightarrow n$ do
4. tempWeight $+= w_i$
5. with probability $\frac{w_i}{\text{tempWeight}}$ replace e by e_i
6. return e

Memory Complexity :- The algorithm only maintains a single element from the stream and total weight upto the time, so memory used is $O(B + \log n)$ bits, where B is the size of element, and $\log n$ is used for weight as the max weight can go upto n^2, needing $2 * \log n$ bits.

Correctness :- Let $P[e = e_i]$ be the probability that i^{th} element is selected

need :- $P[e = e_i] = \frac{w_i}{\sum_{j=1}^{n} w_j}$

Using Induction

Base Case $n = 1$

$e_1 = x_1$

$P[e = e_1] = 1$ ✓

Induction

Assume $P[e = e_t] = \frac{w_i}{\sum_{i=1}^{t} w_i}$

for $e = t+1$

$P[e = e_{t+1}$ when e_{t+1} is selected$] = \frac{w_{t+1}}{\sum_{i=1}^{t+1} w_i}$

$$P\left[e = \text{some other edge rather than } e_{t+1}\right] = \left(1 - \frac{w_{t+1}}{\sum_{i=1}^{t+1} w_i}\right) \cdot \left(\frac{w_t}{\sum_{i=1}^{t} w_i}\right)$$

$$= \frac{\cancel{\sum_{i=1}^{t} w_i}}{\sum_{i=1}^{t+1} w_i} \cdot \frac{w_{t+1}}{\cancel{\sum_{i=1}^{t} w_i}}$$

$$= \frac{w_{t+1}}{\sum_{i=1}^{t+1} w_i}$$

3. generalization of majority element

stream size $= n$ elements   stream $= x_1, \ldots, x_n$

element $= O(B)$ bits

k distinct elements

each element $> \frac{n}{k+1}$ times

output k elements by using $O(K(B+\log n))$ memory

1. dict_imp = { } → or map in c++
2. for $i = 1 \longrightarrow n$ do
3. if x_i is in dict_imp, increment the value of that key by 1
4. elseif |dict_imp| < k, add x_i to dict_imp and have values as 1
5. else reduce all the values of elements in dict_imp by 1 and remove elements with value 0
6. return the keys in dict_imp

Memory Complexity :- Maintaining k elements with size of each element being B we need kB bits and then k log n for their values to be ~~stored~~. Putting them both together we need $O(kB + k\log n)$ bits of memory or $O(k(B+\log n))$ bits of memory.

Correctness :- When a new element is seen by the algorithm and all 'k' counters are filled, the algorithm assumes none of them are majority elements and reduces every counter by 1.
If we imagine having a subset of all data ~~of~~ size k+1, having k elements that appear more than $\frac{n}{k+1}$ times, and maybe 1 element that does not. The counter of all elements would be reduced only once due to the 1 different element. Doing that for every subset would end up with a result of a net value of 'k' required elements as 1 in dict_imp, and the keys would be the output.

#4.

Input : $G = (V, E, w)$

$\varepsilon \in (0, 1)$

algo using $O(\frac{n}{\varepsilon^2} \text{ poly} \cdot \log n)$ bits of memory

T_{chunk} size of chunk $= c \cdot \frac{n}{\varepsilon^2}$

1. sparcifiers $= \phi$
2. for $i = 1 \longrightarrow$ # of edges in stream do:
3. for every $c \cdot \frac{n}{\varepsilon^2}$ chunk, call it G_i
4. $H_i = S(G_i, \varepsilon)$
5. sparcifiers $=$ sparcifiers $\cup$ H_i
6. if |sparcifier| is power of 2
7. replace two entries in sparcifiers with the result of S(sparcifier entry1, sparcifier entry, ε)
8. if |sparcifier| > 1 redo :- 6, 7