**Approach :**

# ***data*** is a collection of 10 billion vectors each of which 100-dimensional

# ***point*** is a 100-dimensional vector

* We’ll create a **Max\_heap** *mx\_heap* where each entry in *mx\_heap* is a tuple of distance and index.
* Initially insert the distances between first 10 vectors in *data* and *point* and their corresponding indices into *mx\_heap*.
* Traverse through each vector *data[i]* (*i*>10) and compare the distance between vectors *data[i]* and *point*
* If the distance is greater than the maximum distance in *mx\_heap*, then it indicates there exist at least 10 vectors whose distance from *point* is less than distance between *data[i]* and *point*.
* Otherwise, we’ll remove the maximum distance tuple in *mx\_heap* and insert the distance between *data[i]* and *point* into *mx\_heap.*
* Finally *mx\_heap* contains **top-10 nearest neighbor distances and their indices**.

# Method **top10()** returns the indices of **top-10** nearest vectors of ***point*** in ***data.***

**Pseudocode :**

***top10****(data,point)*

*{*

*N = 10^7*

*Max\_heap mx\_heap;*

*for i = 1:10*

*{*

*distance = distance between vectors data[i](ith vector in data) and point*

*max\_heap.push(distance , i)*

*}*

*for i = 10:N*

*{*

*distance = distance between vectors data[i](ith vector in data) and point*

*If distance < max\_heap.top():*

*max\_heap.pop()*

*Max\_heap.push(distance , i)*

*}*

*Array indices;*

*For i in mx\_heap*

*{*

*indices.insert(max\_heap.top()[1])*

*mx\_heap.pop()*

*}*

*return indices*

*}*