**G30HW1 Analysis**

The goal of this short analysis is to estimate **M\_L** of the function *MRPrintStatistics* implemented in our program, and to show that it respects the constraints of the MapReduce model.

The *MRPrintStatistics* function goes through the following steps:

1. It assigns each point in the dataset to its closest centroid using squared Euclidean distance.
2. Then it pairs each point with its cluster index and group (either A or B): *((cluster\_idx, group), 1).*
3. It uses *reduceByKey* to count the number of points from each group per cluster.
4. Then it collects the result locally into a dictionary that stores the stats for each cluster.

Each entry in the final dictionary includes:

* The cluster index *i*,
* The coordinates of the centroid (2 float values),
* The number of points from group A *(NA\_i),*
* The number of points from group B *(NB\_i).*

So, for every cluster, we store 4 numeric values in total.

We have number of clusters K. Each cluster needs:

2 coordinates (floats) + NA + NB = 4 numbers. Each number takes 8 bytes.

So for each cluster: *4 × 8 bytes = 32 bytes*  
And for all clusters: *M\_L = K × 32 bytes -> small*

All heavy computations involving the full dataset (of size |U|) are done using RDD operations. Only a small summary of size O(K) is collected locally via collect(). That means we are respecting the MapReduce constraint — only a small amount of local memory is used, and the big data is handled in a distributed way.

**Conclusion**

The function *MRPrintStatistics* follows the memory limitations required by the MapReduce model.  
Its local memory usage depends only on K, which is small, so M\_L stays within acceptable bounds. All large data is processed using RDDs, and only a small, final result is processing by the driver.