1. Iteration Time Analysis:

A.) Profile time taken for 10 iterations :

```
[cs3401.13@rce problem2]$ cat profile_metrics.csv
iteration,total_time,map_time,shuffle_time,reduce_time,io_time
1,.699071653,.200142809,.009216478,.171740394,.021829070
2,1.093970894,.190360870,.009464200,.171110557,.022118983
3,1.103579650,.190807635,.009361694,.170378042,.022201892
4,1.101387144,.204288877,.009166210,.170009458,.022300698
5,1.106438217,.190528156,.009248777,.181801138,.021752108
6,1.090062649,.189172085,.009240302,.172048684,.022044166
7,.965525758,.190290005,.009393922,.159471015,.021947386
8,1.113359892,.201523542,.009110608,.170377814,.021737397
9,1.117379305,.203165720,.009208903,.170339300,.021674876
10,1.106991690,.189777854,.009084010,.183828016,.021778133
```

On the shared test case containing 1000 nodes, these are the profile times for the 10 iterations.

Average values (approximately):

- Total iteration time = **1.07 sec**
- Map time = **0.195 sec (18%)**
- Shuffle time = **0.009 sec (0.84%)**
- Reduce time = **0.172 sec (16%)**
- IO time = **0.022 sec (2%)**

B.) Overhead per iteration :

Now, Map + Shuffle + Reduce = **0.376 sec**

=> Overhead = 1.07 - 0.376 = **0.694 sec**

=> 65% of time is framework overhead.

C.) Time for I/O : 0.022 sec approx

2. Scalability Analysis:

We analyze scaling with:

- |V| (nodes)
- |E| (edges)

A.) Theoretical Complexity

Each iteration:

Map Phase:

For each node:

- emit adjacency list => O(|V|)
- emit one contribution per outgoing edge => O(|E|)

So:

Map complexity = O(V + E)

Shuffle + Sort:

Sort on intermediate data:

Intermediate size= ≈ O(V + E)

Sorting complexity: O((V+E) log(V+E))

This is the dominant cost.

Reduce:

One reduce per node: O(V + E)

Overall Per Iteration:

O((V+E) log(V+E))

Total:

O(I (V+E) log(V+E))

B.) Validation :

For 1000 nodes, with average degree of 5 :

```
iteration,total_time,map_time,shuffle_time,reduce_time,io_time
1,.688740260,.202591640,.008507463,.171395372,.020382741
2,1.101295390,.188100353,.008737501,.170695784,.021165567
3,1.077677635,.188216075,.008594518,.171568174,.020948184
4,1.110663099,.200595707,.008847840,.181602799,.021574595
5,1.099566657,.188788630,.008724266,.172272852,.021507376
6,1.087750103,.200932390,.008890018,.170618654,.021392625
7,1.117459323,.201171549,.009114308,.170955311,.021599475
8,1.093666239,.189473568,.008794205,.172403472,.021263592
9,1.103729576,.188454798,.008712652,.170588673,.021076925
10,1.109676198,.199504188,.008930582,.182233105,.021393708
```

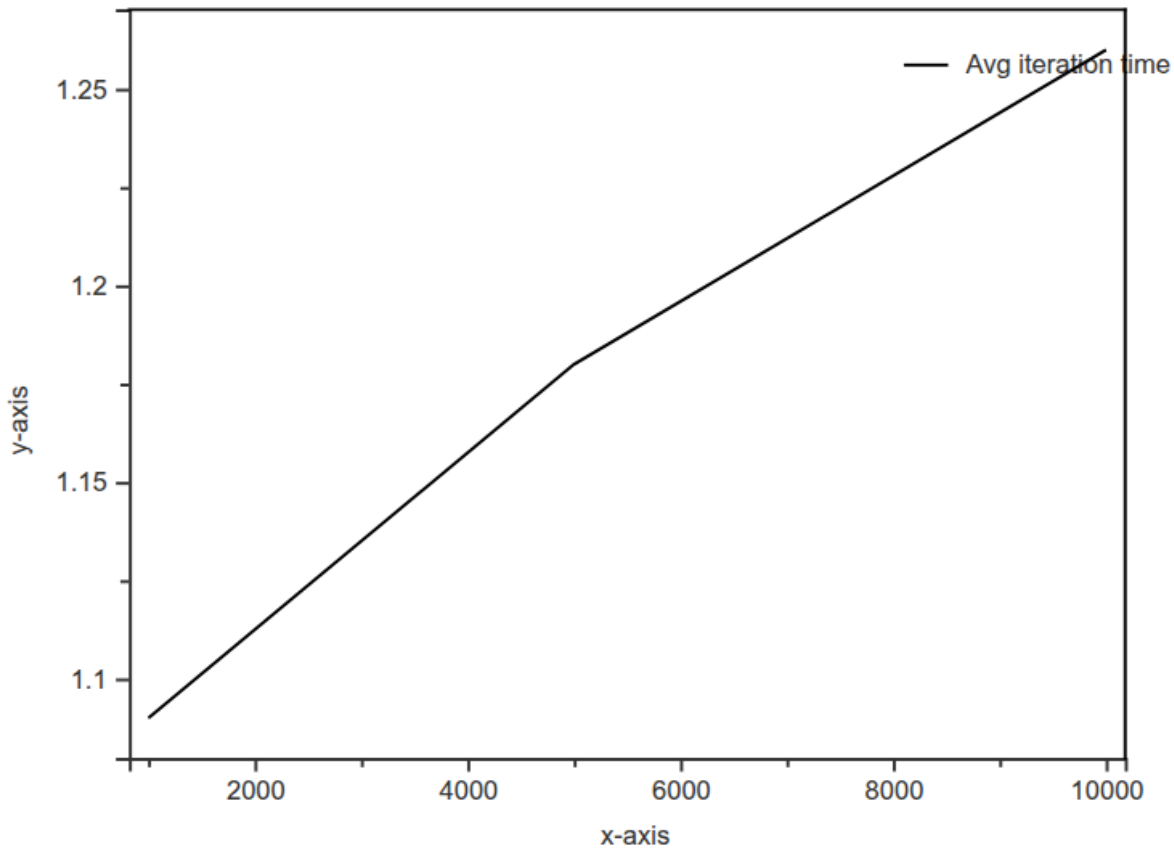
For 5000 nodes, with average degree of 5 :

```
iteration,total_time,map_time,shuffle_time,reduce_time,io_time
1,.764524009,.227314698,.027739465,.191205850,.053911685
2,1.169696276,.217224450,.028478715,.190365537,.054745309
3,1.174209831,.231204841,.028893959,.165428836,.055579227
4,1.188148406,.231383409,.029101086,.191728096,.056074182
5,1.171361402,.229056765,.028930411,.165941346,.055475953
6,1.198880681,.230986456,.029274296,.189240915,.056487459
7,1.197189838,.218973481,.029331380,.189933848,.056544421
8,1.173918462,.219147739,.029193774,.178035636,.056452980
9,1.181059952,.229563133,.029350930,.177887033,.056416459
10,1.171091921,.229834127,.029173217,.166798828,.056133805
```

For 1000 nodes, with average degree of 5 :

```
iteration,total_time,map_time,shuffle_time,reduce_time,io_time
1,.843810128,.258593768,.052345043,.187158462,.094314926
2,1.254217691,.261099525,.052965135,.186568546,.097090864
3,1.250770931,.242995369,.055371262,.187252483,.100542760
4,1.262319208,.253769091,.054683296,.186205052,.099619054
5,1.274638793,.267745201,.055344687,.186525796,.100595531
6,1.250058071,.254448698,.054815529,.186108002,.099862502
7,1.279074868,.266273789,.054950604,.198304371,.099555814
8,1.251817054,.242003903,.055140104,.187460984,.100205700
9,1.257227240,.267139489,.055148441,.199163939,.099988343
10,1.260479813,.266429353,.055234602,.174910938,.099978620
```

=> Average iteration time
for 1000 nodes is **1.09 seconds**
for 5000 nodes is **1.18 seconds**
for 10000 nodes is **1.26 seconds**



C.) Observations:

From 1k to 10k:

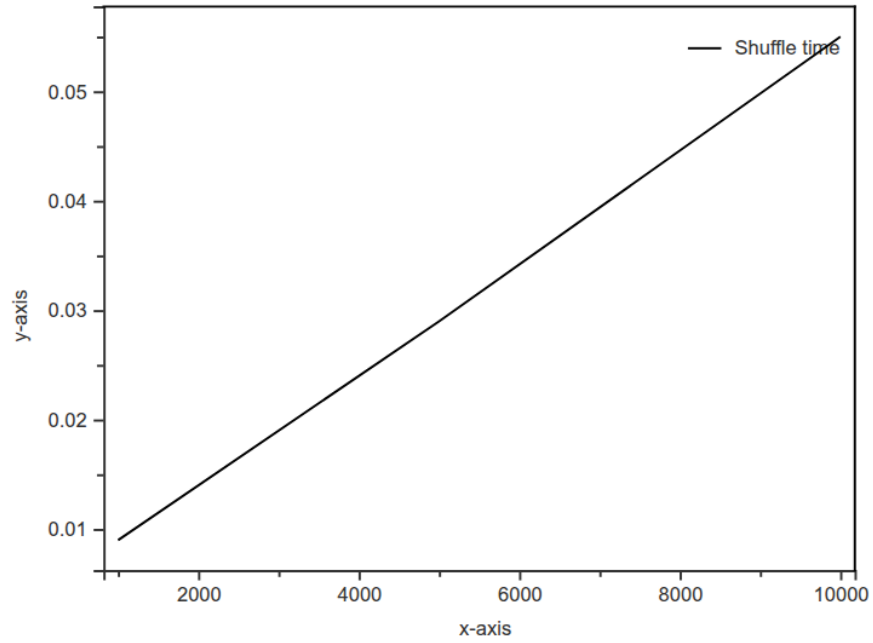
- Nodes increased **10×**
- Edges increased **10×**
- Time increased only **~15%**

That means: runtime is not dominated by graph size yet. It is dominated by fixed overhead.

Shuffle time :

Nodes	Shuffle
1k	~0.009s
5k	~0.029s
10k	~0.055s

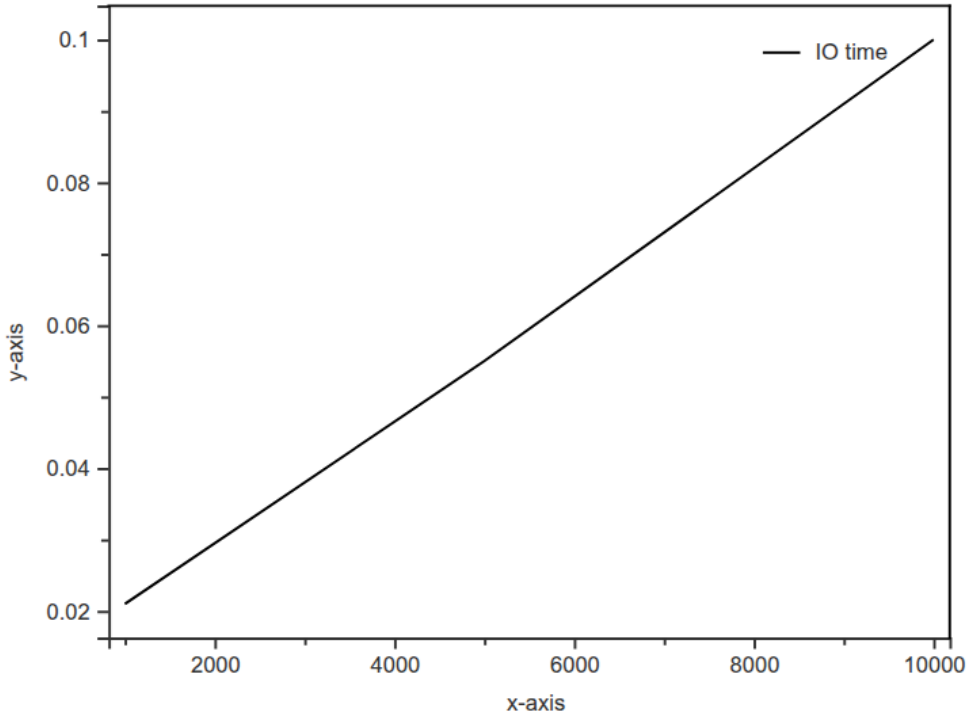
This scales almost perfectly linearly with edges. That is expected because shuffle size = O(E)



IO Time:

Nodes	IO
1k	0.021s
5k	0.055s
10k	0.10s

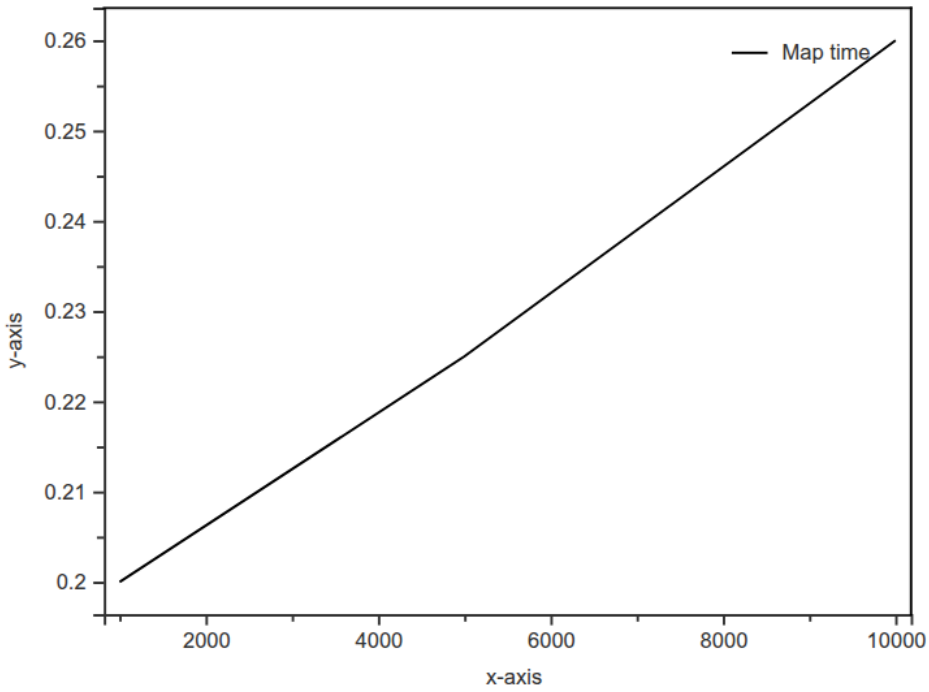
Also nearly linear with E.



Map Time:

Nodes	Map
1k	0.19–0.20
5k	0.22–0.23

Nodes	Map
10k	0.25–0.27



Reduce Time:

Almost constant (~0.18–0.19). That means reducer cost is small relative to overhead.

D.) Compare With Theoretical Complexity

Theoretical per iteration: O((V+E)log(V+E))

Since avg degree is constant (5): E=O(V)

So: T(V) ≈ O(VlogV)

Now check growth: If pure V log V -

For 1k: 1000log(1000)

For 10k: 10000log(10000)

Ratio ≈ 100000log100000/1000log1000

≈ **13.3×**

But the observed ratio:

1.26/1.09 ≈ **1.15×**

This huge difference might be due to the dominance of overhead at small scale, i.e – the true compute time is being hidden by SLURM startup, Barrier sync, Process creation, etc.

Validation to this claim : Extract Real Compute Scaling -

Subtract fixed overhead.
From 1k:
Total ≈ 1.09
Map + Shuffle + Reduce + IO ≈ 0.20 + 0.009 + 0.17 + 0.02 ≈ 0.40

So overhead ≈ 0.69 sec.

Subtract that overhead from others:
For 5k:
1.18 – 0.69 ≈ 0.49 sec compute

For 10k:
1.26 – 0.69 ≈ 0.57 sec compute

Now compare compute only:
0.40 to 0.57 ≈ 1.4×

Much closer to theoretical scaling.

E.) Estimating Maximum Graph Size for 10 Minutes

10 iterations of 10k:
10 × 1.26 ≈ 12.6 sec

Time limit = 600 sec

Scale factor:

600 / 12.6 ≈ 47×

So estimated maximum:
Nodes ≈ 10k × 47 ≈ **470k nodes**
Edges ≈ 50k × 47 ≈ **2.35 million edges**

Before hitting 10 min limit.

*But, sorting will eventually dominate, so actual max slightly lower.

F.) Conclusions

Observation 1

Execution time increases sub-linearly with graph size for small graphs because distributed framework overhead dominates computation.

Observation 2

Shuffle and I/O times scale linearly with number of edges, confirming O(E) behavior.

Observation 3

Sorting cost grows but remains small relative to SLURM startup overhead for graphs <= 10k nodes.

Observation 4

True asymptotic behavior will approach O(V log V) once graph size becomes large enough for compute to dominate overhead.

3. I/O Performance

```
iteration,total_time,map_time,shuffle_time,reduce_time,io_time,data_written_bytes
1,.857885915,.259474532,.051899915,.185965274,.094195851,3801431
2,1.258600327,.263630842,.052998031,.186289373,.096936120,4079377
3,1.263931891,.253699119,.054985757,.187341370,.099586610,4456782
4,1.245088458,.255197069,.054247927,.197665426,.099512785,4413192
5,1.265259412,.255682555,.055603319,.199030260,.101227070,4424312
6,1.287056976,.255926970,.055633717,.197778187,.100360434,4434942
7,1.266993689,.255162341,.056403029,.186366156,.101572283,4422517
8,1.228603684,.255073014,.054922233,.175278143,.100828233,4422412
9,1.261404791,.254992419,.055884247,.198851325,.100782001,4426207
10,1.150590617,.253871461,.055553086,.187626833,.100811295,4423792
```

A.) Average data volume per iteration = 4.4MB
(this is for graph input with 10k nodes, 5 edges on average)

B.) Disk throughput = data_written/io_time = 4.4 MB/0.1 s = 44 MB/s

C.) Network overhead - In real distributed Hadoop, shuffle causes network transfer of intermediate key-value pairs. Our approximation can be - Network cost ≈ size of merged.txt. Because in real cluster, that would be transferred across nodes.