i) (4)

-> import multiprocessing
def mapper(data):

    key, value = data
    result = []

    for word in value.split():
        result.append((word, 1))
    return result

def reducer(data):

    key, value = data
    return (key, sum(values))

def mapreduce(data, mapper, reducer, num_process=2):

    pool = multiprocessing.pool(processes=num_proce
sss

    mapped_data = pool.map(mapper, data)

    flattend_data = [item for sublist in mapped
        _data for item in sublist]

    grouped_data = {}
        for key, value in flattened_data:
            grouped_data.setdefault(key, []).appen
(value)

```
grouped_data_items = list(grouped_data.items())

reduced_data = pool.map(reducer, grouped_data_items)

pool.close()
pool.join()

return reduced_data

if __name__ == "__main__":

input_data = [(1, "apple banana"), (2, "banana, orange"), (3, "orange, apple")]

result = mapreduce(input_data, mapper, reducer)

for item in result:

    print(item)
```

(4) (ii)

```
-> import multiprocessing

class MapReduce:

    def __init__(self, num_processes = 2):
        self.num_processes = num_processes

    def mapper(self, data):
        raise NotImplementedError("Subclasses must
                                   implement mapper method")

    def reducer(self, data):
        raise NotImplementedError("Subclasses must
                                   implement reducer method")

    def map(self, data):
        result = []
        for item in data:
            result.extend(
                self.mapper(item)
            )
        return result

    def reduce(self, data):
        reduced_data = {}

        for key, value in data:
```

```
        reduced_data.setdefault(key, []).append
                                        (value)
        return [(key, sum(values)) for key, values
                in reduced_data.items()]

    def run(self, data):
        pool = multiprocessing.Pool(processes = self.
                                        num_processes)
        chunk_size = len(data) // self.num_processes
        chunks = [data[i:i + chunk_size]
            for i in range(0, len(data), chunk_
                                        size)]
        Fluttened_data = [
            item for sublist in mapped_dat
                for item in sublist
        ]
        result = self._reduce(fluttened_data)
        pool.close()
        pool.join()
        return result

class WordCount(MapReduce):
    def mapper(self, data):
        key, value = data
        result = []
        for word in value.split():
            result.append((word, 1))
        return result

    def reducer(self, data):
        key, value = data
        return (key, sum(values))
```

```
if __name__ == "__main__":
    input_data = [(1, "apple banana"), (2, "banana orange"), (3, "orange apple")]
    word_count_job = WordCount()
    result = word_count_job.run(input_data)
    for item in result:
        print(item)
```

(4) (iii)

-> from mrjob.job import MRJob
from mrjob.step import MRStep

class WordCount(MRJob):

def steps(self):
return [MRStep(mapper=self.mapper, reducer=
self.reduce

def mapper(self, _, line):
"""Tokenizes each line and emits (word,
"""
for word in line.split():
yield word.lower(), 1

if __name__ == '__main__':
WordCount.run()

- input.txt

apple banana
banana orange
orange apple.