TASK 1

TASK 1.1: IMPLEMENT A PYTHON PROGRAM (EXTRACTION)

```python
from pymongo import MongoClient

# Making database connection
client = MongoClient('localhost', 27017)
db = client["assignment1"]
col = db["movies"]
cur = col.find()

# Write a txt file with <year, company> pairs of all movies
file = open("year_and_company.txt", "w")
# Loop through each movie document in the cur
for movie in cur:
    for i in range(min(len(movie['companies']), 3)):
        year_company = movie['date'][-4:] + " , " + movie['companies'][i]['name']+ '\n'
    file.write(year_company)
file.close()
```

 # Making database connection
 # Open a txt file for sorting year and company data
 # Loop through each movie document in the cur
For each I in the range from 0 to min(3, the length of movie[‘companies’]):
    Extract the last four characters (year) of movie[‘date’]
    Extract the company name from the ‘companies’ field
    Construct a string containing the year and company name using a delimiter “,”
    Split the line into year and company
 # Write the year_company file
 # Close the txt file

TASK 1.2: IMPLEMENT THE MAPREDUCE PROGRAM (COUNT)

```python
from mrjob.job import MRJob

class MRWordCount(MRJob):
    def mapper(self, _, line):
        yield line, 1

    def reducer(self, year_company, count):
        yield year_company, sum(count)

if __name__ == "__main__":
    MRWordCount.run()
```

Input the text file

 # Mapper function

Input: Each line contains a record in the format "year_company, count"

For each line in the input:

Split the line into year_company and count using a delimiter ","

Emit key-value pair (line, 1)

 # Reducer function

Input: List of key-value pairs

// key: year and company

//value: a list of counts

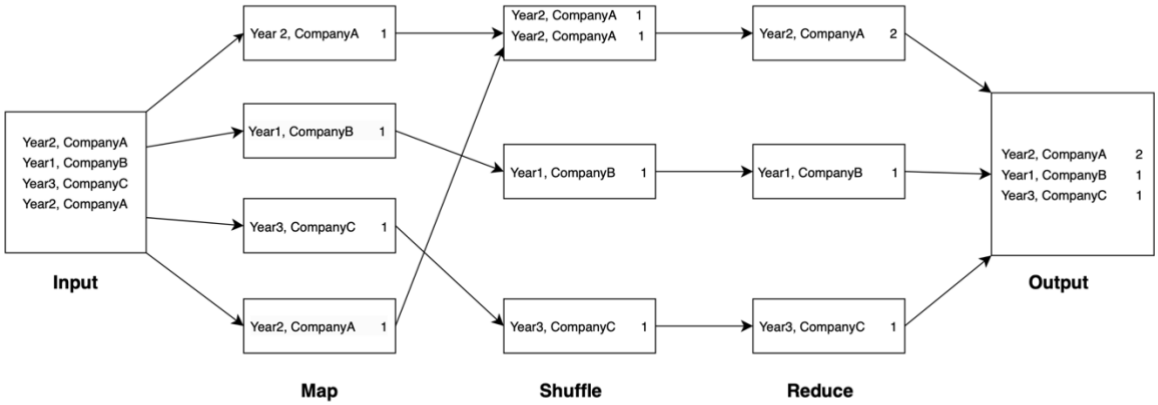
For each key-value pair in the input:

Initialize a variable sum(count) to 0

For each count in the list of counts:

Emit key-value pair (year_company, sum(count))

FLOWCHART FOR TASK 1



TASK 2

TASK 2.1: MERGE SORT

```python
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRMergeSort(MRJob):

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

    def mapper(self, _, line):
        year_company, count = line.split('\t')
        count = int(count)
        # "key" for combining year_company and count
        yield "key", (year_company[1:-1], count)

    def merge_sort(self, arr):
        if len(arr) > 1:
            mid = len(arr) // 2
            left_half = arr[:mid]
            right_half = arr[mid:]

            self.merge_sort(left_half)
            self.merge_sort(right_half)

            i = j = k = 0
            while i < len(left_half) and j < len(right_half):
                # Sort in ascending order
                if left_half[i][1] < right_half[j][1]:
                    arr[k] = left_half[i]
                    i += 1
                else:
                    arr[k] = right_half[j]
                    j += 1
                k += 1

            while i < len(left_half):
                arr[k] = left_half[i]
                i += 1
                k += 1

            while j < len(right_half):
                arr[k] = right_half[j]
                j += 1
                k += 1

        # Return the sorted array
        return arr

    def reducer(self, _, tuples):
        sorted_tuples = self.merge_sort(list(tuples))
        for tuple in sorted_tuples:
            yield tuple[0], tuple[1]

if __name__ == '__main__':
    MRMergeSort.run()
```

Input the txt file

 # Steps function

 # Mapper function

Input: Each line contains a record in the format "year_company, count"

For each line in the input

Split the line into year_company and count using a delimiter “\t”

Combine year_company and count by using “key”

Emit key-value pair (“key” (year_company, count))

yield "key", (year_company[1:-1], count)

 # Merge_sort function

If the length of array >1:

Calculate the middle index mid

Split the array into left_half and right_half

Call merge_sort on left_half and right_half

Initialize variables i, j, and k to 0

While i < length of left_half and j < length of right_half:

Compare elements at indices i and j based on the second element of each tuple

Update array at index k with element from left_half

Increment i

Else:

Update array at index k with the element from right_half

Increment j

Increment k

Copy any remaining elements from left_half

Copy any remaining elements from right_half

 # Return the sorted array

 # Reducer function

Input: List of key-value pairs

// key: year_company

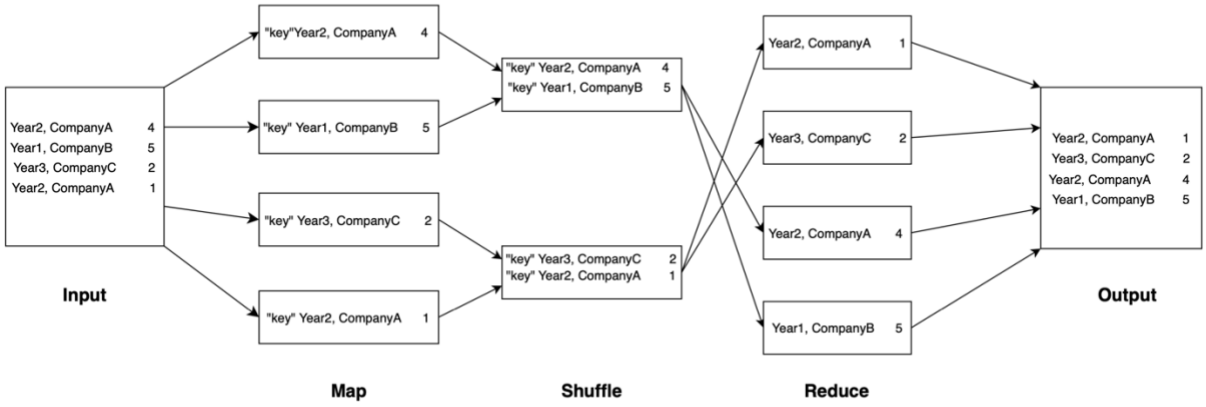
// value: count

Sort the list of tuples using merge-sort

For each tuple in the sorted list:

Emit key-value pair with the first element of the tuple (year_company) and the second element (count)

FLOWCHART FOR MERGE SORT



TASK 2.2: BUCKET SORT

```python
from mrjob.job import MRJob
from mrjob.step import MRStep

class MRBucketSort(MRJob):

    def configure_args(self):
        super(MRBucketSort, self).configure_args()
        self.add_passthru_arg('--num_buckets', type=int, default=250)
        self.add_passthru_arg('--bucket_size', type=int, default=3)

    def steps(self):
        return [
            MRStep(
                mapper=self.bucket_assignment_mapper
            ),
            MRStep(
                reducer=self.bucket_sort_reducer
            ),
            MRStep(
                reducer=self.bucketid_sort_reducer
            )
        ]

    def bucket_assignment_mapper(self, _, line):
        year_company, count = line.split('\t')
        count = int(count)
        bucket_id = count // self.options.bucket_size
        yield bucket_id, (year_company[1:-1], count)


    def bucket_sort_reducer(self, bucket_id, records):
        # Sort in descending order
        sorted_records = sorted(records, key=lambda x: (-x[1], x[0]))
        for record in sorted_records:
            yield "key",(bucket_id, record)

    def bucketid_sort_reducer(self, key, bucketid_records):
        for value in sorted(bucketid_records, key=lambda x:x[0], reverse=True):
            yield value[1]

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

```

Input the txt file

 # Configure command-line arguments function

 # Steps function

 # Mapper function

Split the input line into year_company and count using a delimiter “\t”

Convert count to an integer

Calculate the bucket_id based on count and bucket size

Emit key-value pair
// key: bucket_id
// value: count

 # Reducer function for records (count) sorting

Sort records in descending order based on the second element of each tuple (records)
Emit sorted key-value pairs
Combine bucket_id and record by using “key”
// key: “key”
// tuple: bucket_id and record

 # Reducer function for bucket_id sorting

Sort records in descending order based on the first element of each tuple (bucket_id)
Emit sorted sorted values from bucketid_records

FLOWCHART FOR BUCKET SORT