| QUESTION 1|

1. The first ten 3-shingles are going to be the first 10 shingles of length 3 from the text at the start of section 3.2:

'The','he ','e m',' mo','mos','ost','st ','t e',' ef','eff'

| QUESTION 2|

1. To get the number of k-shingles in an n byte document, start by getting the shingle at the first byte, and continue until the final shingle which would be at the (n-k+1) byte, so in total you could get this many shingles:

n-k+1

| QUESTION 3|

3a. The permutation of {a, b, c, d, e} corresponding to h3(x): b, a, e, d, c The permutation of {a, b, c, d, e} corresponding to h4(x): b, c, a, e, d

3b. Table of final signatures after all four hash functions:

S1 S2 S3 S4 1 3 0 1 0 2 0 0 1 4 0 1 3 0 1 0

Hash Functions used
h1(x) h2(x) h3(x) h4(x)

3c. Jaccard Similarities, treating the sets as not having duplicates, doing the union / intersection of data points

S1,S2 = 2/5 = 0.4 S1,S3 = 2/3 = 0.67 S1,S4 = 2/3 = 0.67 S2,S3 = 1/5 = 0.2 S2,S4 = 1/5 = 0.2 S3,S4 = 2/2 = 1.0

3d. Similarity Table, comparison for each is found by doing # of equal rows / # of total rows:

S1,S2: 0/4 = 0.0 S1,S3: 1/4 = 0.25 S1,S4: 3/4 = 0.75 S2,S3: 0/4 = 0.0 S2,S4: 1/4 = 0.25 S3,S4: 1/4 = 0.25

| QUESTION 4|

For this question for I stored my data(answers) from question 3 in variables for the jaccard similarities and signatures, then used those in code to answer the parts of question 4. The code that does this is directly after this section.

4a.

Band 1 Buckets: {'S1': 1, 'S2': 0, 'S3': 0, 'S4': 1} Band 2 Buckets: {'S1': 4, 'S2': 4, 'S3': 1, 'S4': 1}

4b.

LSH Pairs Identified: {('S3', 'S4'), ('S1', 'S4'), ('S2', 'S3'), ('S1', 'S2')}

4c.

Avg JS (LSH): 0.5675 Avg JS (Non-LSH): 0.435

```
### QUESTION 4 CODE ###
import numpy as np

# Data I'm using based on my Question 3 answers for signatures and
jaccard similarities
signatures = {
    "S1": [1, 0, 1, 3],
    "S2": [3, 2, 4, 0],
    "S3": [0, 0, 0, 1],
    "S4": [1, 0, 1, 0],
}
jaccard_similarities = {
    ("S1", "S2"): 0.4,
    ("S1", "S3"): 0.67,
    ("S1", "S4"): 0.67,
    ("S2", "S3"): 0.2,
    ("S2", "S4"): 0.2,
    ("S3", "S4"): 1.0,
}

# Hash function + buckets for bands
def band_hash(x, y):
    return (x + y) % 5

b1Buckets = {key: band_hash(values[0], values[1]) for key, values in
signatures.items()}
b2Buckets = {key: band_hash(values[2], values[3]) for key, values in
signatures.items()}

# Find alike pairs in each band
def getAlikePairs(bucket):
    pairs = []
    for key1 in bucket:
        for key2 in bucket:
            if key1 != key2 and bucket[key1] == bucket[key2]:
                p = tuple(sorted((key1, key2)))
                if p not in pairs:
                    pairs.append(p)
    return pairs

b1Pairs = getAlikePairs(b1Buckets)
b2Pairs = getAlikePairs(b2Buckets)

# LSH identified pairs
lshPairs = set(b1Pairs + b2Pairs)

# Compute average Jaccard similarities
lshSim = [jaccard_similarities[pair] for pair in lshPairs]
notlshPairs = set(jaccard_similarities.keys()) - lshPairs
notlshSim = [jaccard_similarities[pair] for pair in notlshPairs]
```

```
avg_lshSim = round(np.mean(lshSim),4)
avg_notlshSim = round(np.mean(notlshSim),4)

# 4a
print("Band 1 Buckets:", b1Buckets)
print("Band 2 Buckets:", b2Buckets)
# 4b
print("LSH Pairs Identified:", lshPairs)
# 4c
print("Avg JS (LSH):", avg_lshSim)
print("Avg JS (Non-LSH):", avg_notlshSim)
```

| QUESTION 5|

Users with jaccard similarity > 0.5, found using the code below:

*I will also be submitting this as a .csv file

Pair # User1 User2 Jaccard Sim. 0 8 94 0.537313 1 8 347 0.508197 2 8 379 0.509091 3 38 235 0.521277 4 38 446 0.528302 5 46 126 0.509434 6 46 242 0.540000 7 46 468 0.595745 8 56 94 0.522388 9 56 126 0.527273 10 81 126 0.600000 11 81 340 0.500000 12 94 126 0.540984 13 94 347 0.603175 14 94 379 0.508197 15 94 470 0.511111 16 94 512 0.536232 17 107 130 0.550000 18 107 468 0.558140 19 126 130 0.609756 20 126 179 0.507042 21 126 242 0.520833 22 126 340 0.571429 23 126 347 0.509091 24 126 379 0.681818 25 126 468 0.510638 26 126 485 0.564103 27 126 498 0.586957 28 126 512 0.543860 29 126 574 0.564103 30 130 145 0.700000 31 130 242 0.575000 32 130 468 0.694444 33 130 485 0.593750 34 130 574 0.700000 35 145 468 0.513514 36 145 574 0.586207 37 150 270 0.609756 38 174 584 0.515152 39 242 468 0.658537 40 242 574 0.526316 41 270 389 0.574468 42 340 374 0.525000 43 347 379 0.557692 44 379 498 0.510638 45 379 512 0.535714 46 446 566 0.533981 47 455 512 0.528571 48 468 574 0.513514

```
### QUESTION 5 CODE ###
import pandas as pd
from itertools import combinations

ratings = pd.read_csv("ratings.csv")

# Grouping by userId
user_movies = ratings.groupby("userId")
["movieId"].apply(set).reset_index()

# Dictionary for quick access
umDict = dict(zip(user_movies["userId"], user_movies["movieId"]))

# Jaccard calculation
def jaccard_similarity(set1, set2):
    intersection = len(set1 & set2)
    union = len(set1 | set2)
    return intersection / union if union != 0 else 0
```

```
alikePairs = []

# Comparing all users
for user1, user2 in combinations(umDict.keys(), 2):
    js = jaccard_similarity(umDict[user1], umDict[user2])
    if js >= 0.5:
        alikePairs.append((user1, user2, js))

# Put in a df for easier reading of output
alikePairs_df = pd.DataFrame(alikePairs, columns=["User1", "User2",
"Similarity"])
print(alikePairs_df)
```

| QUESTION 6|

The code used to get all the results is directly after this section. Since there were many user pairs that met the 0.5 threshold, I stored the actual pairs and their score in a different .csv for each different number of hash functions and am submitting those separately. I did include the total number of 0.5+ similarity pairs found by each # of hash functions though.

6a.

Finished checking with 50 hash functions: There were 92 user pairs that met the 0.5 threshold

6b.

Finished checking with 100 hash functions: There were 46 user pairs that met the 0.5 threshold

6c.

Finished checking with 200 hash functions: There were 40 user pairs that met the 0.5 threshold

```
### QUESTION 6 CODE ###
import pandas as pd
import numpy as np
from itertools import combinations

ratings = pd.read_csv("ratings.csv")

# Grouping by userId
user_movies = ratings.groupby("userId")
["movieId"].apply(set).reset_index()

# Dictionary for quick access and set of all movies
umDict = dict(zip(user_movies["userId"], user_movies["movieId"]))
all_movies = set(ratings["movieId"].unique())

modVal = 11999  # Mod value given in footnote

# Create x amount of hash functions
```

```
def generate_hashes(numHash):
    hashes = []
    for i in range(1, numHash + 1):
        hashes.append(lambda x, i=i: ((2 * i + 1) * x + 100 * i) %
modVal)
    return hashes

# Min-Hashing the original signatures
def getSignatures(user_movies, hashes):
    signatures = {}
    for user, movies in user_movies.items():
        signature = []
        for h in hashes:
            signature.append(min(h(movie) for movie in movies))
        signatures[user] = signature
    return signatures

# Minhashing to get similarity
def getSimilarity(s1, s2):
    matches = sum(1 for a, b in zip(s1, s2) if a == b)
    return matches / len(s1)

# Find similar users
def getUserPairs(signatures):
    jaccard_threshold = 0.5
    alikePairs = []
    for user1, user2 in combinations(signatures.keys(), 2):
        sim = getSimilarity(signatures[user1], signatures[user2])
        if sim >= jaccard_threshold:
            alikePairs.append((user1, user2, sim))
    return alikePairs

# Calling all the above functions with different amounts of hash
functions
for numHash in [50, 100, 200]:
    hashes = generate_hashes(numHash)
    user_signatures = getSignatures(umDict, hashes)
    alikePairs = getUserPairs(user_signatures)

    # This time outputs are larger so I'm storing in a .csv file
instead
    alikePairs_df = pd.DataFrame(alikePairs, columns=["User 1", "User
2", "Sim. Score"])
    output_file = f"alikePairs_minhash_{numHash}.csv"
    alikePairs_df.to_csv(output_file, index=False)

    print(f"Finished checking with {numHash} hash functions:")
    print(f"There were {len(alikePairs)} user pairs that met the 0.5
threshold")
```

| QUESTION 7|

I started by rerunning the code for true jaccard similarities but this time storing it in a .csv file so that I could open and read it for this section. After running the code below I got the following results:

Accuracy when using 50 hash functions: False Positives: 66 True Positives: 26 False Negatives: 23 True Negatives: 185630

Accuracy when using 100 hash functions: False Positives: 17 True Positives: 29 False Negatives: 20 True Negatives: 185679

Accuracy when using 200 hash functions: False Positives: 9 True Positives: 31 False Negatives: 18 True Negatives: 185687

Based on these results we can see that while the minshaing was never 100% accurate, as the number of hash functions increased, so did the true accuracy, and minhashing with 200 hash functions produced the most accurate results when compared to the true jaccard similarities of each user pair.

```
### QUESTION 7 CODE ###
import pandas as pd

# True pairs from jaccard sim. scores
truePairs = pd.read_csv("alikePairs_jaccard.csv")
alikePairs_true = set(
    (row["User 1"], row["User 2"]) for _, row in truePairs.iterrows()
)

def comparePairs(mh_filepath, alikePairs_true, all_user_pairs):
    # Minhahsing results to compare against the 'true' results
    pairs_mh = pd.read_csv(mh_filepath)
    alikePairs_mh = set(
        (row["User 1"], row["User 2"]) for _, row in
pairs_mh.iterrows()
    )

    # t/f positives and negatives
    falsePos = len(alikePairs_mh - alikePairs_true)
    truePos = len(alikePairs_mh & alikePairs_true)
    falseNeg = len(alikePairs_true - alikePairs_mh)
    trueNeg = len(all_user_pairs - alikePairs_mh - alikePairs_true)

    return falsePos, truePos, falseNeg, trueNeg

# Info to get the total possible pairs of users (for t/f negatives)
ratings = pd.read_csv("ratings.csv")
user_movies = ratings.groupby("userId")
["movieId"].apply(set).reset_index()
umDict = dict(zip(user_movies["userId"], user_movies["movieId"]))
ids = list(umDict.keys())
```

```
all_user_pairs = set(
    (u1, u2) for u1 in ids for u2 in ids if u1 < u2
)

# Check accuracy of x hash functions
for num_hashes in [50, 100, 200]:
    mh_filepath = f"alikePairs_minhash_{num_hashes}.csv"
    falsePos, truePos, falseNeg, trueNeg = comparePairs(mh_filepath,
alikePairs_true, all_user_pairs)

    print(f"Accuracy when using {num_hashes} hash functions:")
    print(f"False Positives: {falsePos}")
    print(f"True Positives: {truePos}")
    print(f"False Negatives: {falseNeg}")
    print(f"True Negatives: {trueNeg}")
    print()
```