```
from google.colab import drive
drive.mount('/content/drive')
```

```
# Function to read the data
def ReadData(filepath):
  a = []
  with open(filepath, "r") as file:
    for t in file:
      t = t.strip().split()
      l = list(map(int,t))
      a.extend(l)
  return a

  # read data
data1 = ReadData('T10I4D100K.dat')
data2 = ReadData('T40I10D100K.dat')
data3 = ReadData('kosarak.dat')
```

```
# imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

Q1

Implement the Tidemark algorithm for estimating the number of distinct elements. Test it for the stream consisting of all the numbers in the file, windows of 50000 numbers each, compare it with the ground truth and plot this information.

```
# Vibhanshu Jain
# CS19B1027
```

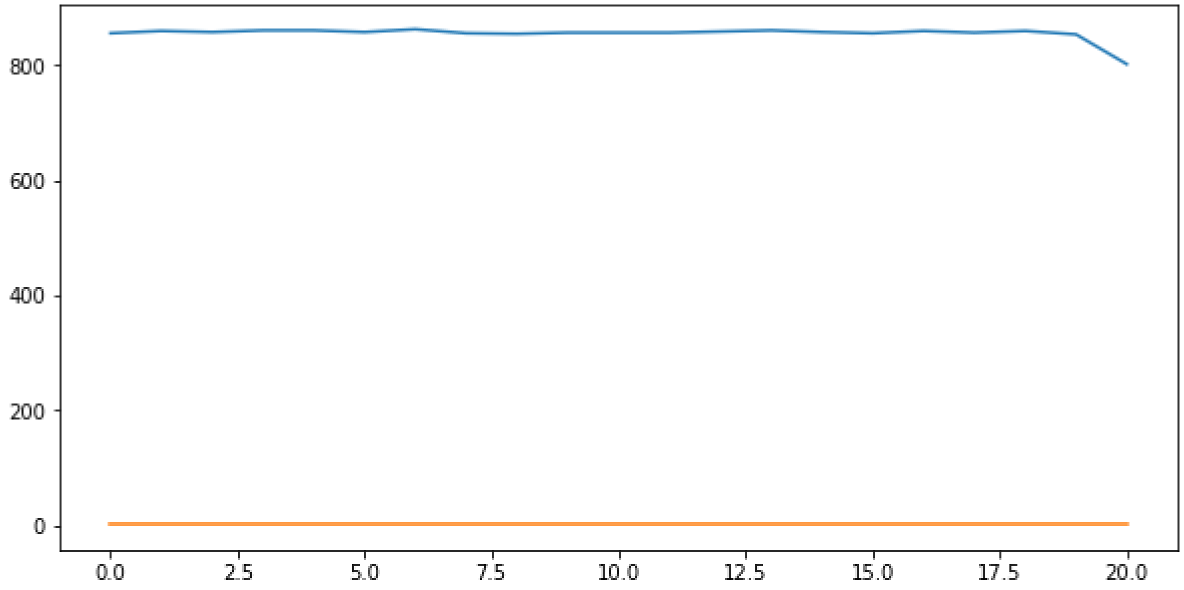
```
# Q1:
# tidemark algorithm implementation python

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Function to read the data
def ReadData(filepath):
  a = []
  with open(filepath, "r") as file:
    for t in file:
      t = t.strip().split()
      l = list(map(int,t))
      a.extend(l)
  return a

# tidemark algorithm implementation python

# Pair wise independent hash functions
# K > N
N = 20
K = 30
# random matrix of size N*K
A = np.random.randint(0, 100)

# random matrix of size N
B = np.random.randint(0, 100)

def hash_function(x):
     # hash function h(x) = (ax + b) mod 2
    return (np.dot(A,x) + B) % 2

# For an integer p > 0, zeros(p) is the number of zeros that the binary representation of p ends with.
def zeroes(p):
    if p > 0:
        # print(bin(p))
```

```
        temp = [i for i in str(bin(p))[2:]]
        # print(temp)
        zero = 0
        for j in range(len(temp)-1,0,-1):
            if temp[j] == '0':
                zero+=1
            else:
                break
        return zero

# read data
data1 = ReadData('/content/drive/MyDrive/scalable/T10I4D100K.dat')
data2 = ReadData('/content/drive/MyDrive/scalable/T40I10D100K.dat')
data3 = ReadData('/content/drive/MyDrive/scalable/kosarak.dat')

# function to calculate the tidemark

def tidemark(data):
    z = 0
    for i in data:
        z = max(zeroes(hash_function(i)), z)
    return 2**(z+0.5)

def tidemark_implemention(data,k):
    # find the length and divide the data 50000 each and then call time mark function on each of them
    data_len = len(data)
    data_tm = []
    for i in range(0,data_len,50000):
        data_tm.append(tidemark(data[i:i+50000]))
    return data_tm


# function to calculate the tidemark for each data set
data1_tm = tidemark_implemention(data1, 50000)
data2_tm = tidemark_implemention(data2, 50000)
data3_tm = tidemark_implemention(data3, 50000)
```

```
# function to print the results
def print_results(data, data_tm):
    print('The tidemark for data set ' + str(data) + ' is ' + str(data_tm))

print_results(1, data1_tm)
print_results(2, data2_tm)
print_results(3, data3_tm)


# ground truth values are number of distinct value in the chunk return by the tidemark function

# for data1
data1_gt = [len(set(data1[i:i+50000])) for i in range(0,len(data1),50000)]

# for data2
data2_gt = [len(set(data2[i:i+50000])) for i in range(0,len(data2),50000)]

# for data3
data3_gt = [len(set(data3[i:i+50000])) for i in range(0,len(data3),50000)]

# plot the results

# Plot for data 1
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
ax.plot(data1_gt, label='ground truth')
ax.plot(data1_tm, label='tidemark')
plt.show()

# plot for data 2
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
ax.plot(data2_gt, label='ground truth')
ax.plot(data2_tm, label='tidemark')
plt.show()

# plot for data 3
fig, ax = plt.subplots(1, 1, figsize=(10, 5))
```

```
ax.plot(data3_gt, label='ground truth')
ax.plot(data3_tm, label='tidemark')
plt.show()
```

```
The tidemark for data set 1 is [1.4142135623730951, 1.4142135623730951, 1.4142135623730951, 1.4142135623730951, 1.4142135623730
The tidemark for data set 2 is [1.4142135623730951, 1.4142135623730951, 1.4142135623730951, 1.4142135623730951, 1.4142135623730
The tidemark for data set 3 is [1.4142135623730951, 1.4142135623730951, 1.4142135623730951, 1.4142135623730951, 1.4142135623730
```



Q2

Write a code to test whether there is a number that appears at least m/10 times in the stream, where m is the length of the stream. If so, what is the frequency of that number. That is implememnt the heavy hitters algorithm where k = 10.

```
# Vibhanshu Jain
# CS19B1027
# Question 2

# the value of k given in the question
k = 10

# the heavy hitters function
def heavyHitters(data):
```

```
  # two arrays of size k to store top k elements
  top = np.zeros(k)
  freq = np.zeros(k)

  # looping into the complete data
  for i in data:

    # check if element is already present in top
    if i in top:

      # if present, increment its frequency
      freq[top == i] += 1

    else:
      # if not present, check if there is a space in top

      if 0 in top:

        # if there is a space, add the element
        top[top == 0] = i
        freq[top == i] += 1

      else:
        # if there is no space, decrement the frequency of all elements

        freq -= 1

        # if frequency becomes 0, remove the element
        top[freq == 0] = 0
        freq[freq == 0] = 0

  # returning the top and frequency back
  return top, freq

# verification function to check if the top k elements are correct
def verifyFunction(top, data):

    # initilizing the count to 0
```

```
    count = 0

    # empty result array
    result = []

    # iterating in top array
    for i in top:

        # if i is in the data then increase the count
        if i in data:
            count += 1

        # the condition asked in the question
        if count > len(data)/k:
            result.append(i)
    return result

# calling the heavy hitters function of the all the dataset
top1, freq1 = heavyHitters(data1)
top2, freq2 = heavyHitters(data2)
top3, freq3 = heavyHitters(data3)

# calling the verification function of all the dataset
result1 = verifyFunction(top1, data1)
result2 = verifyFunction(top2, data2)
result3 = verifyFunction(top3, data3)

# printing the results
print("Top elements in T10I4D100K.dat are: ", result1)
print("Top elements in T40I10D100K.dat are: ", result2)
print("Top elements in kosarak.dat are: ", result3)
```

```
Top elements in T10I4D100K.dat are:  []
Top elements in T40I10D100K.dat are:  []
Top elements in kosarak.dat are:  []
```

Q3

Implement Bloom filter with the following values of the sketch size 50, 70, 100, 150, 500, 1000, 2000. Please use the appropriate values of the hash function as per the sketch size and number of items in the stream. Consider the first 5% of elements as your test datasets (don't include the test dataset while creating bloom filter), and report the confusion matrix corresponding to each datasets, on various values of the sketch size mentioned above.

```
# Vibhanshu Jain
# CS19B1027
# Question 3

# Pair wise independent hash functions

from sklearn.metrics import confusion_matrix

# the k value given in the question / size of the bloom filter
k = 10

# sketch size
sketchValues = [50, 70, 100, 150, 500, 1000, 2000]
# class of hash functions
class HashFunction:
  def __init__(self, N, K):
    self.N = N
    self.K = K
    self.a = np.random.randint(0, 100)
    self.b = np.random.randint(0, 100)

  # the function to calculate the hash value
  def hash(self, x):
    return (np.dot(self.a,x) + self.b) % k

# diving the dataset into training and testing, 95% training and 5% testing
train1 = data1[:int(0.95*len(data1))]
```

```
test1 = data1[int(0.95*len(data1)):]
train2 = data2[:int(0.95*len(data2))]
test2 = data2[int(0.95*len(data2)):]
train3 = data3[:int(0.95*len(data3))]
test3 = data3[int(0.95*len(data3)):]

# creating k hash functions
hash_functions = []
for i in range(k):
  hash_functions.append(HashFunction(N,K))


# bloom filter function
def bloom_filter(data, hash_functions, bloomFilter):
  for i in data:
    for j in hash_functions:
      bloomFilter[j.hash(i)] = 1

# query the bloom filter
def query_bloom_filter(hash_functions,ele, bloomFilter):
  for hash in hash_functions:
    if bool(bloomFilter[int(hash.hash(ele))]) == 0:
      return False
  return True

# loop over the different values of m
for m in sketchValues:
  for data in [(train1,test1,"T10I4D100K"), (train2,test2,"T40I10D100K"), (train3,test3, "kosarak")]:
    bloomFilter = np.zeros(m)

    # add elements to the bloom filter
    bloom_filter(data[0], hash_functions, bloomFilter)

    # Query the bloom filter
    res=[0]*len(data[1])

    # actual result array
```

```
    actual=[0]*len(data[1])

    # Testing the function
    for i in range(len(data[1])):
      res[i] = query_bloom_filter(hash_functions,data[1][i],bloomFilter)
      if data[1][i] in data[0]:
        actual[i] = 1

    # calculate the false positive rate
    false_positive_rate = sum(res)/len(res)
    # print("False positive rate for T10I4D100K.dat is: ", false_positive_rate)

    # calculate the true positive rate
    true_positive_rate = sum([res[i] == actual[i] for i in range(len(res))])/len(res)
    # print("True positive rate for T10I4D100K.dat is: ", true_positive_rate)

    # calculate the confusion matrix
    cm = confusion_matrix(actual, res)
    print("Confusion matrix for dataset: ", data[2], " and m: ", m ," is: ", cm)
```

Q4

Implement Count-min-sketch algorithm with the following values of (t, k) = {(50, 50),(25, 100),(250, 10),(500, 5)} 2 . Consider the first 5% of elements as your test datasets (consist of query items), and report the RMSE bar charts on these values of (t, k). The RMSE is defined as follows – for each query item, compute the difference of its ground truth frequency and its estimation from the sketch, square all these values, add them up, and compute the mean. Note that smaller RMSE is an indication of better performance.

```
# Vibhanshu Jain
# CS19B1027
# Question 4

Values = [(50, 50),(25, 100),(250, 10),(500, 5)]
```

```
# class of hash functions
class HashFunction:
  def __init__(self,k):
    self.a = np.random.randint(0, 100)
    self.b = np.random.randint(0, 100)
    self.k = k

  def hash(self, x):
    return (np.dot(self.a,x) + self.b) % self.k

# diving the dataset into training and testing, 95% training and 5% testing
train1 = data1[:int(0.95*len(data1))]
test1 = data1[int(0.95*len(data1)):]
train2 = data2[:int(0.95*len(data2))]
test2 = data2[int(0.95*len(data2)):]
train3 = data3[:int(0.95*len(data3))]
test3 = data3[int(0.95*len(data3)):]


# count min-sketch function
def CountMinFunction(data, hash_functions, countMin, t):
  for ele in data:
    for i in range(t):
      countMin[i][hash_functions[i].hash(ele)] += 1
  return countMin

# count min-sketch query function
def CountMinQuery(data, hash_functions, countMin, t):
  count = 0
  for ele in data:
    count += min([countMin[i][hash_functions[i].hash(ele)] for i in range(t)])
  return count

# loop over the different values
for m in Values:
  for data in [(train1,test1,"T10I4D100K"), (train2,test2,"T40I10D100K"), (train3,test3, "kosarak")]:
    # an 2d array of t*k
    countMin = np.zeros((m[0],m[1]))
```

```
    countMin = np.zeros((m[0],m[1]))

    # creating k hash functions
    hash_functions = []
    for i in range(m[1]):
      hash_functions.append(HashFunction(m[1]))
    # the count min result
    countMinResult = CountMinFunction(data[0], hash_functions, countMin, m[0])

    # count min query function
    countMinQueryResult = CountMinQuery(data[1], hash_functions, countMinResult, m[0])
    print("Count Min Query Result for dataset: ", data[2], " is ", countMinQueryResult)
```

Q5

Repeat the above for the Count-Sketch algorithm. In the bar-chart, put the bar-chart results of Count-sketch and Count-min-sketch side-by-side for comparison.

```
# Vibhanshu Jain
# CS19B1027
# Question 5

# the t value given in the question / size of the bloom filter
t = 50

# array size
k = 50

# class of hash functions
class HashFunction:
  def __init__(self, N, K):
    self.N = N
    self.K = K
    self.a = np.random.randint(0, 100)
    self.b = np.random.randint(0, 100)
```

```
  def hash(self, x):
    return (np.dot(self.a,x) + self.b) % k

# class of count sketch hash functions
class CountSketchHashFunction:
  def __init__(self, N, K):
    self.N = N
    self.K = K
    self.a = np.random.randint(0, 100)
    self.b = np.random.randint(0, 100)

  def hash(self, x):
    if (np.dot(self.a,x) + self.b) % 2 == 1:
      return 1
    else:
      return -1

# diving the dataset into training and testing, 95% training and 5% testing
train1 = data1[:int(0.95*len(data1))]
test1 = data1[int(0.95*len(data1)):]
train2 = data2[:int(0.95*len(data2))]
test2 = data2[int(0.95*len(data2)):]
train3 = data3[:int(0.95*len(data3))]
test3 = data3[int(0.95*len(data3)):]

# creating k hash functions
hash_functions = []
for i in range(t):
  hash_functions.append(HashFunction(N,K))

# creating second k hash function array for count sketch
hash_functions2 = []
for i in range(t):
  hash_functions2.append(CountSketchHashFunction(N,K))

# an 2d array of t*k
```

```
countSketch = np.zeros((t,k))

# count sketch function
def CountSketchFunction(data, hash_functions, countSketch):
  for ele in data:
    for i in range(t):
      countSketch[i][hash_functions[i].hash(ele)] += hash_functions2[i].hash(ele)

# count sketch query function
def CountSketchQuery(data, hash_functions, countSketch):
  count = 0
  for ele in data:
    count += np.median([ countSketch[i][hash_functions[i].hash(ele)] for i in range(t) ])
  return count

# calculting the result
countSketch = np.zeros((t,k))
countSketchResult = CountSketchFunction(train1, hash_functions, countSketch)

# testing the data
countSketchQueryResult = CountSketchQuery(test1, hash_functions, countSketch)
print("Count Min Query Result for t = 50 and k = 50 ", countSketchQueryResult)
```

```
Count Min Query Result for t = 50 and k = 50  -244741924.0
```