**DAA Assignment- I**

1 .Given a row wise sorted matrix of size **R\*C** where R and C are always **odd**, find the median of the matrix. **5Marks**

**Test Case 1:**

# Python program to find median of matrix

# sorted row wise

from bisect import bisect\_right as upper\_bound

MAX = 100;

# Function to find median in the matrix

def binaryMedian(m, r, d):

mi = m[0][0]

mx = 0

for i in range(r):

if m[i][0] < mi:

mi = m[i][0]

if m[i][d-1] > mx :

mx = m[i][d-1]

desired = (r \* d + 1) // 2

while (mi < mx):

mid = mi + (mx - mi) // 2

place = [0];

# Find count of elements smaller than or equal to mid

for i in range(r):

j = upper\_bound(m[i], mid)

place[0] = place[0] + j

if place[0] < desired:

mi = mid + 1

else:

mx = mid

print ("Median is", mi)

return

# Driver code

r, d = 3, 3

m = [ [1, 3, 5], [2, 6, 9], [3, 6, 9]]

binaryMedian(m, r, d)

OUTPUT:

Median is 5

**Test Case 2:**

# Python program to find median of matrix

# sorted row wise

from bisect import bisect\_right as upper\_bound

MAX = 100;

# Function to find median in the matrix

def binaryMedian(m, r, d):

mi = m[0][0]

mx = 0

for i in range(r):

if m[i][0] < mi:

mi = m[i][0]

if m[i][d-1] > mx :

mx = m[i][d-1]

desired = (r \* d + 1) // 2

while (mi < mx):

mid = mi + (mx - mi) // 2

place = [0];

# Find count of elements smaller than or equal to mid

for i in range(r):

j = upper\_bound(m[i], mid)

place[0] = place[0] + j

if place[0] < desired:

mi = mid + 1

else:

mx = mid

print ("Median is", mi)

return

# Driver code

r, d = 3,1

m = [ [1], [2], [3]]

binaryMedian(m, r, d)

OUTPUT:

Median is 2

2. . Given the arrival and departure times of all trains that reach a railway station, the task is to find the minimum number of platforms required for the railway station so that no train waits. We are given two arrays that represent the arrival and departure times of trains that stop. **5Marks**

**Test Case 1:**

import heapq

# Function to find the minimum number

# of platforms required

def findPlatform(arr, dep, n):

arr2 = []

# Store the arrival and departure time

for i in range(n):

arr2.append([arr[i], dep[i]])

arr2.sort() # Sort trains based on arrival time

p = []

count = 1

heapq.heappush(p, arr2[0][1])

for i in range(1, n):

# Check if arrival time of current train

# is less than or equals to departure time

# of previous train

if p[0] >= arr2[i][0]:

count += 1

else:

heapq.heappop(p)

heapq.heappush(p, arr2[i][1])

# return the count of number of platforms required

return count

if \_\_name\_\_ == "\_\_main\_\_":

arr = [900, 940, 950, 1100, 1500, 1800]

dep = [910, 1200, 1120, 1130, 1900, 2000]

n = len(arr)

print(findPlatform(arr, dep, n))

OUTPUT:

3

**Test Case 2:**

import heapq

# Function to find the minimum number

# of platforms required

def findPlatform(arr, dep, n):

arr2 = []

# Store the arrival and departure time

for i in range(n):

arr2.append([arr[i], dep[i]])

arr2.sort() # Sort trains based on arrival time

p = []

count = 1

heapq.heappush(p, arr2[0][1])

for i in range(1, n):

# Check if arrival time of current train

# is less than or equals to departure time

# of previous train

if p[0] >= arr2[i][0]:

count += 1

else:

heapq.heappop(p)

heapq.heappush(p, arr2[i][1])

# return the count of number of platforms required

return count

if \_\_name\_\_ == "\_\_main\_\_":

arr = [900, 940]

dep = [910, 1200]

n = len(arr)

print(findPlatform(arr, dep, n))

OUTPUT:

1

Roll number: 21071A6786