Algorithms Assignment -2 :

(2a)
In this algorithm, we use naive approach to sort the passengers in the increasing order of their priority. We use 2-Dimesional array to store the records of the passengers where each row represents a queue and each column represents the passenger in the queue. Initially we copy all the records on the first array on to an empty array called “final”, where n is the maximum number of passengers each queue can have and k is the number of queues. Later we merge the second queue on to the final array. Followed by merging third queue on to the final array. This process repeats until all the k arrays are merged to form a single final array which consists of all the passengers sorted in the increasing order of their priority.

Assumptions: All k queues consist of a maximum of n passengers and all queues are sorted in the decreasing order of passenger’s priority

Given: a 2D array to represent k queues with a maximum of n passengers in each queue, result array “res” to store all the passengers in the sorted order and a temporary array “temp”

MergeQueues(array, res, temp)

```
    for n = 0 to array[0].length                                c1
        res[n] = array[0][n]                                    c2

    for i = 1 to array.length                                   c3
        j= 0;                                                   c4
        k= 0;                                                   c5
        h = 0;                                                  c6
        m1 = array[i]. length;                                  c7
        m2 = res.length;                                        c8

        for l =0 to res.length                                  c9
            temp[l] = res[l]                                    c10

        while j < m1 and k < m2                                 c11
            if array[i][j] < temp[k]                            c12
                res [h] = temp[k]                               c13
                h++                                             c14
                k++                                             c15
            else                                                c16
                res [h] = array[i][j]                           c17
                h++                                             c18
                j++                                             c19

        while j < m1                                            c20
            res [h] = array[i][j]                               c21
            h++                                                 c22
            j++                                                 c23
        while k < m2                                            c24
```

```
res [h] = temp[k]                                   c25
k++                                                 c26
h++                                                 c27
```

(2a)Time Complexity:

k => no. of queues
n => maximum number of passengers in each Queue

Steps 1 and 2 executes for maximum of n times = C1*n

Steps 3 to 8 executes for maximum k times = C3*k

Steps 9 to 10 executes for maximum kn times = C2*k*n

This algorithm involves merging arrays of size n with another array of size n to form an array of size 2n. Then it involves merging array of size 2n with n to form 3n. This process continues until all the k arrays are merged. The time complexity of the steps from c11 to c27 will be of the form:

$2n + 3n + 4n + \ldots\ldots + (k)n = n(1+2+3+\ldots\ldots+k) - n$

By solving the above arithmetic progression series, we have

$$= n\frac{(k * (k+1))}{2} - n$$

From the above equations we have,

$$T(n) = \frac{nk^2}{2} + \frac{kn}{2} - n + C1 * n + C2*k*n + C3 * k$$

Ignoring all the constants and smaller terms, which are insignificant for large values of n, we now have

Total time complexity, $T(n) = O(nk^2)$

(2b) In this algorithm, we start with k queues and start dividing them into smaller sets of size k/2. This process repeats until we have just one queue. This results in a tree like structure where original k queues form the root and sub set form the intermediate node and leaves consisting of just one queue. Now we start merging the queues and recurse back to the root node by merging all the queues on to a single final array which consists of all the passengers arranged in the decreasing order of their priority

Assumptions: All k queues consist of a maximum of n passengers and all queues are sorted in the decreasing order of passenger’s priority

Given: Three arrays where array1 and array2 represents two sorted queues which needs to be merged and a result array to store the merged queues in sorted order.
Returns: an array which represents the sorted queue which is got by merging the given two input queues

```
MergeArray(array1, array2, res)
        n1 = array1.length
        n2 = array2.length
        i = 0
        j = 0
        k= 0
        while i < n1 and j < n2
                if array1[i] > array2[i]
                        res[k] = array1[i]
                        i++
                        k++
                else
                        res[k] = array2[j]
                        j++
                        k++
        while i < n1
                res[k] = array1[i]
                i++
                k++
        while j < n2
                res[k] = array2[j]
                j++
                k++
        return res
```

Given: 2Dimensional array where row represents the queues and column represents passengers in the queues and two integers l and r which represents right and the leftmost queue index.
Returns: a one-dimensional array which is sorted in the decreasing order of passanger'spriority

```
1   MergeKArrays (A[][], r, l,res)
2       if (l<r)
3               mid =(l+r)/2;
4               array1 = MergeKArrays(A, r, mid)
5               array2 = MergeKArrays(A, mid+1,l)
6               mergedArray = MergeTwoArrays(array1, array2, res)
7               return mergedArray
8       return A[l]
```

(2b)Time complexity:

Let T(k) be the time taken by MergeKArrays. Since step 4 and 5 involves dividing the input data set into half by considering k/2 queues, time complexity for steps 3 and 4 is T(k/2). Cost of step 6 is kn as it involves merging two arrays of max size kn/2 in the worst case.

Remaining steps take some constant time c, which we ignore as it is very small.

By adding all cost of all the steps we get the following recurrence relation:

$T(k) = 2T(k/2) + ckn$

$= 2^2 T(k/2^2) + ckn + ckn$

$= 2^3 T(k/2^3) + ckn+ckn+ckn$

Now if we consider n=2, we have

$T(2) = 2^i T(1) + \sum_{i=0}^{\log k} ckn$ -> Equation *

Therefore, $k/2^i = 1$

⇨ $2^i = k$ -> equation 1
$i = \log k$ -> equation 2

Substituting equation 1 and 2 in equation * we have

⇨ **$T(k) = k\,T(1) + (ckn) \log k$**

W. K. T. T(1) is some constant **C**

⇨ **$T(n) = k\,C + (ckn) \log k$**

Taking into consideration just large terms, the worst case time complexity is:

$T(n) = O(nk\log k)$