JSS MAHAVIDYAPEETHA

JSS SCIENCE AND TECHNOLOGY UNIVERSITY

SRI JAYACHAMARAJENDRA COLLEGE OF ENGINEERING

JSS Technical Institutions Campus, Mysuru – 570 006

A project report on

"DESIGNING OF RAILWAY/BUS STATION"

20IS420

Bachelor of Engineering
in
INFORMATION SCIENCE AND ENGINEERING

by,

SAMEEKSHA R 01JST20IS039
SIDDESH R V 01JST20IS042
RAMESH B S 01JST20IS058

Submitted to

SINDHU G
Assistant professor
Department of Information Science and Engineering

July 2022

Table of Contents

INTRODUCTION

Construction of railway/bus station is not so easy. It requires lot of time, efforts, cost to construct a properly designed station. So civil engineers needs to be careful while designing the stations, any error in designing or calculation will concur loss. Platforms are the main requirements in the construction process. This needs to be designed accurately so that no trains/buses get delayed to enter stations and wastage of additional platforms are avoided. To get the accurate number of platform our project comes into picture. For efficient approach we have mainly used merge sort to get the expected results.

The Merge Sort algorithm is a sorting algorithm that is considered as an example of the divide and conquer strategy. So, in this algorithm, the array is initially divided into two equal halves and then they are combined in a sorted manner. We can think of it as a recursive algorithm that continuously splits the array in half until it cannot be further divided. This means that if the array becomes empty or has only one element left, the dividing will stop, i.e. it is the base case to stop the recursion. If the array has multiple elements, we split the array into halves and recursively invoke the merge sort on each of the halves. Finally, when both the halves are sorted, the merge operation is applied. Merge operation is the process of taking two smaller sorted arrays and combining them to eventually make a larger one.

In this project we use Merge Sort to sort all the arrival and departure trains base on time and also use merge function to find the minimum platform required.

RAILWAY/BUS STATION

WORKING OF MERGE SORT ALGORITHM

Using the Divide and Conquer technique, we divide a problem into subproblems. When the solution to each subproblem is ready, we 'combine' the results from the subproblems to solve the main problem.

Suppose we had to sort an array A. A subproblem would be to sort a sub-section of this array starting at index p and ending at index r, denoted as A[p..r].

DIVIDE
If q is the half-way point between p and r, then we can split the subarray A[p..r] into two arrays A[p..q] and A[q+1, r].

CONQUER

In the conquer step, we try to sort both the subarrays A[p..q] and A[q+1, r]. If we haven't yet reached the base case, we again divide both these subarrays and try to sort them.

COMBINE
When the conquer step reaches the base step and we get two sorted subarrays A[p..q] and A[q+1, r] for array A[p..r], we combine the results by creating a sorted array A[p..r] from two sorted subarrays A[p..q] and A[q+1, r].

PSEUDOCODE FOR MERGE SORT

- Declare left variable to 0 and right variable to n-1
- Find mid by medium formula. mid = (left+right)/2
- Call merge sort on (left,mid)
- Call merge sort on (mid+1,rear)
- Continue till left is less than right
- Then call merge function to perform merge sort.

MERGE SORT ALGORITHM

step 1: start
step 2: declare array and left, right, mid variable
step 3: perform merge function.

```
mergesort(array,left,right)
mergesort (array, left, right)
if left > right
return
mid= (left+right)/2
mergesort(array, left, mid)
mergesort(array, mid+1, right)
merge(array, left, mid, right)
```

step 4: Stop

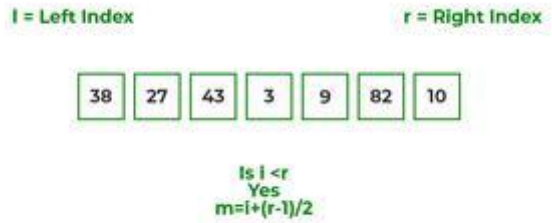
MergeSort(arr[], l, r)

If r > l
Find the middle point to divide the array into two halves:
middle m = l + (r – l)/2
Call mergeSort for first half:
Call mergeSort(arr, l, m)
Call mergeSort for second half:
Call mergeSort(arr, m + 1, r)
Merge the two halves sorted in step 2 and 3:
Call merge(arr, l, m, r)

FUNCTIONING OF MERGE SORT

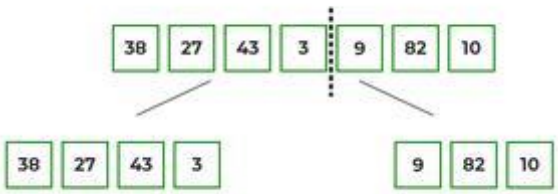
To know the functioning of merge sort, lets consider an array arr[] = {38, 27, 43, 3, 9, 82, 10}

- At first, check if the left index of array is less than the right index, if yes then calculate its mid point

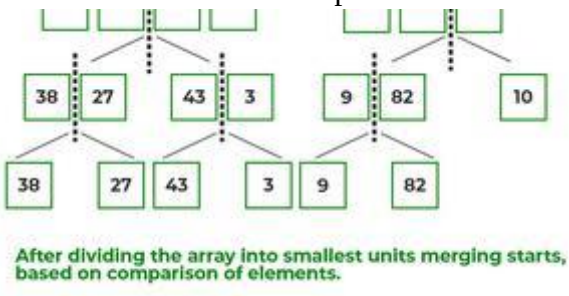


- Now, as we already know that merge sort first divides the whole array iteratively into equal halves, unless the atomic values are achieved.
- Here, we see that an array of 7 items is divided into two arrays of size 4 and 3 respectively.

- Now, again find that is left index is less than the right index for both arrays, if found yes, then again calculate mid points for both the arrays.

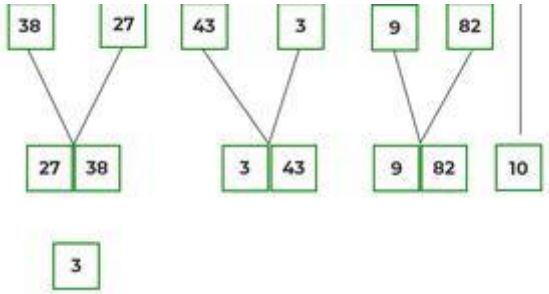


- Now, further divide these two arrays into further halves, until the atomic units of the array is reached and further division is not possible.

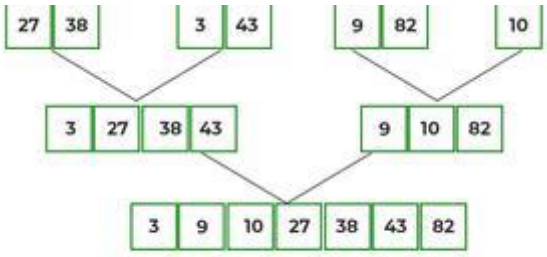


- After dividing the array into smallest units, start merging the elements again based on comparison of size of elements

- Firstly, compare the element for each list and then combine them into another list in a sorted manner.

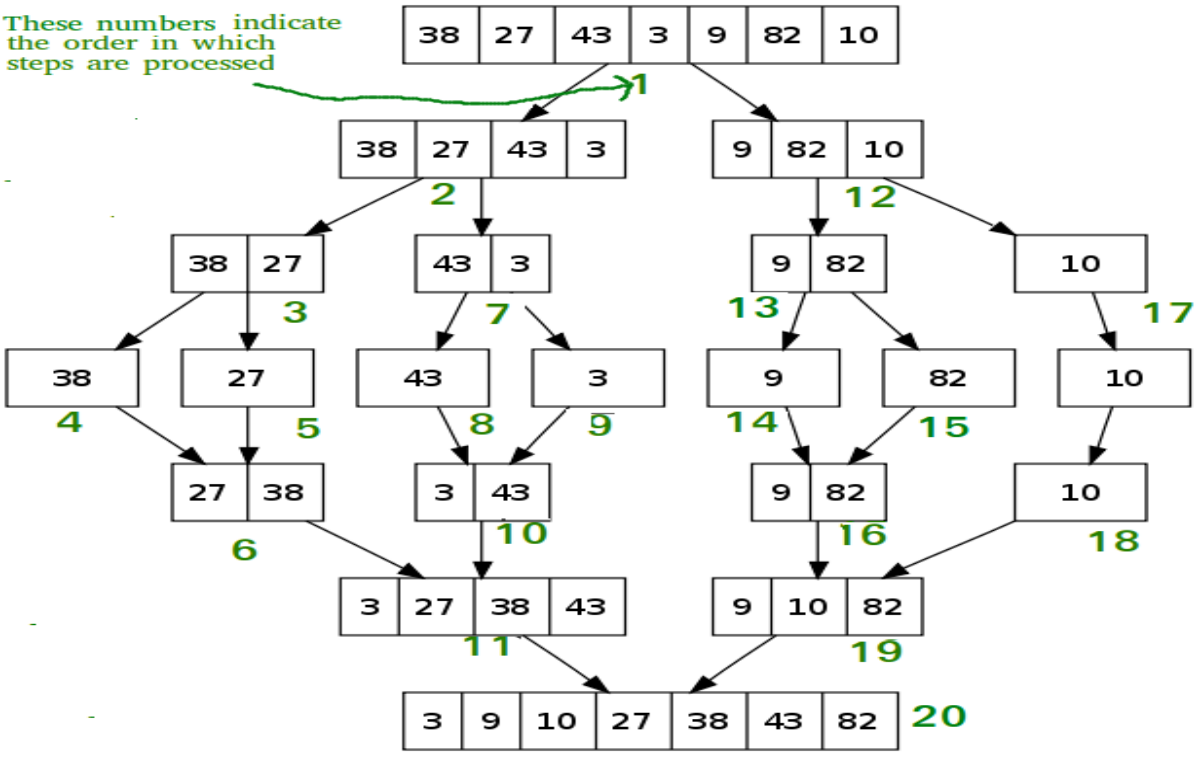


- After the final merging, the list looks like this:



The following diagram shows the complete merge sort process for an example array {38, 27, 43, 3, 9, 82, 10}.
If we take a closer look at the diagram, we can see that the array is recursively divided into two halves till the size becomes

- Once the size becomes 1, the merge processes come into action and start merging arrays back till the complete array is merged.



ANALYSIS OF MERGE SORT ALGORITHM

TIME COMPLEXITY

O(n logn), Sorting arrays on different machines. Merge Sort is a recursive algorithm and time complexity can be expressed as following recurrence relation.

$T(n) = 2T(n/2) + \theta(n)$

The above recurrence can be solved either using the Recurrence Tree method or the Master method. It falls in case II of Master Method and the solution of the recurrence is $\theta(nLogn)$. Time complexity of Merge Sort is $\theta(nLogn)$ in all 3 cases (worst, average and best) as merge sort always divides the array into two halves and takes linear time to merge two halves.

Best Case Complexity - It occurs when there is no sorting required, i.e. the array is already sorted. The best-case time complexity of merge sort is O(n*logn).
Average Case Complexity - It occurs when the array elements are in jumbled order that is not properly ascending and not properly descending. The average case time complexity of merge sort is O(n*logn).

Worst Case Complexity - It occurs when the array elements are required to be sorted in reverse order. That means suppose you have to sort the array elements in ascending order, but its elements are in descending order. The worst-case time complexity of merge sort is O(n*logn).

AUXILLARY SPACE: O(n)

- The space complexity of merge sort is O(n). It is because, in merge sort, an extra variable is required for swapping.

SPACE COMPLEXITY

• In merge sort all elements are copied into an auxiliary array
• so N auxiliary space is required for merge sort.

PROBLEM STATEMENT

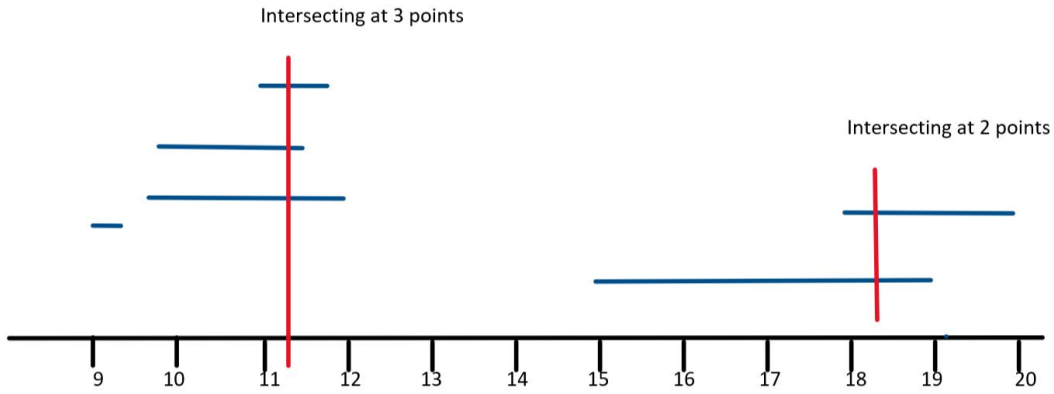
We are given two arrays that represent the arrival and departure times of trains that stop at the platform. We need to find the minimum number of platforms needed at the railway station so that no train has to wait.

Examples 1:

Input: N=6,
arr[] = {9:00, 9:45, 9:55, 11:00, 15:00, 18:00}
dep[] = {9:20, 12:00, 11:30, 11:50, 19:00, 20:00}

Output:3

Explanation: There are at-most three trains at a time. The train at 11:00 arrived but the trains which had arrived at 9:45 and 9:55 have still not departed. So, we need at least three platforms here.



Example 2:

Input Format: N=2,
arr[]={10:20,12:00}
dep[]={10:50,12:30}

Output: 1

Explanation: Before the arrival of the new train, the earlier train already departed. So, we don't require multiple platforms.

DETAILED DESCRIPTION

The idea is to consider all events in sorted order. Once the events are in sorted order, trace the number of trains at any time keeping track of trains that have arrived, but not departed.
Illustration:
arr[] = {9:00, 10:00, 9:40, 13:30, 15:30}
dep[] = {9:50, 14:00, 10:10, 14:20,17:30}

All events are sorted by using merge sort time.
Total platforms at any time can be obtained by subtracting total departures from total arrivals by that time.

The arrays are sorted and merged together

Trains in the Platform right now: 1 Train Arrived!

Trains in the Platform right now: 2 Another Train Arrived!

Trains in the Platform right now: 1 Train Departs!

Trains in the Platform right now: 2 Train Arrives!

Trains in the Platform right now: 1 Train Departs!

Trains in the Platform right now: 2 **Train Arrives!**

Trains in the Platform right now: 1 **Train Departs!**

Trains in the Platform right now: 0 **Train Departs!**

Trains in the Platform right now: 1 **Train Arrives!**

Trains in the Platform right now: 0 **Train Departs!**

The Maximum number of platforms at any time was 2. Hence we need at least 2 platforms.

ALGORITHM FOR THE GIVEN PROBLEM

1. Sort the arrival and departure times of trains.
2. Create two pointers i=0, and j=0, and a variable to store *ans* and current count *plat*
3. Run a loop while i<n and j<n and compare the ith element of arrival array and jth element of departure array.
4. If the arrival time is less than or equal to departure then one more platform is needed so increase the count, i.e., plat++ and increment i
5. Else if the arrival time is greater than departure then one less platform is needed to decrease the count, i.e., plat– and increment j
6. Update the ans, i.e. ans = max(ans, plat).

PSEUDOCODE FOR THE THE GIVEN PROBLEM

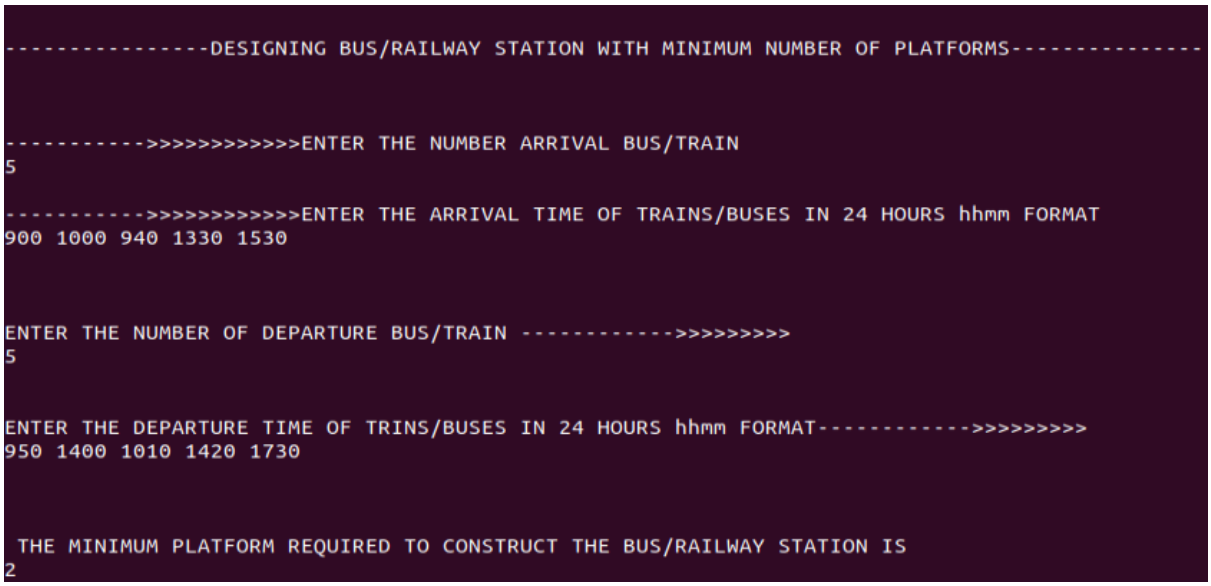
```
result = 1

// loop through the timings (outer loop)
i = 0
while(i < N)
   minimum_platform_needed = 1

   // loop through the timings again and count how many
   //trains intersect with ith train
   j = i + 1
   while(j < N)
     // check if jth train intersects with ith train
     if jth arrival <= ith arrival and jth departure >= ith
     arrival or jth arrival >= ith arrival and jth arrival <= ith
     departure
        minimum_platform_needed++     // minimum platform need is greater than result then
update
   // result
   if minimum_platform_needed > result
      result = minimum_platform_needed
```

Time Complexity: O(N * log N), One traversal O(n) of both the array is needed after sorting O(N * log N).

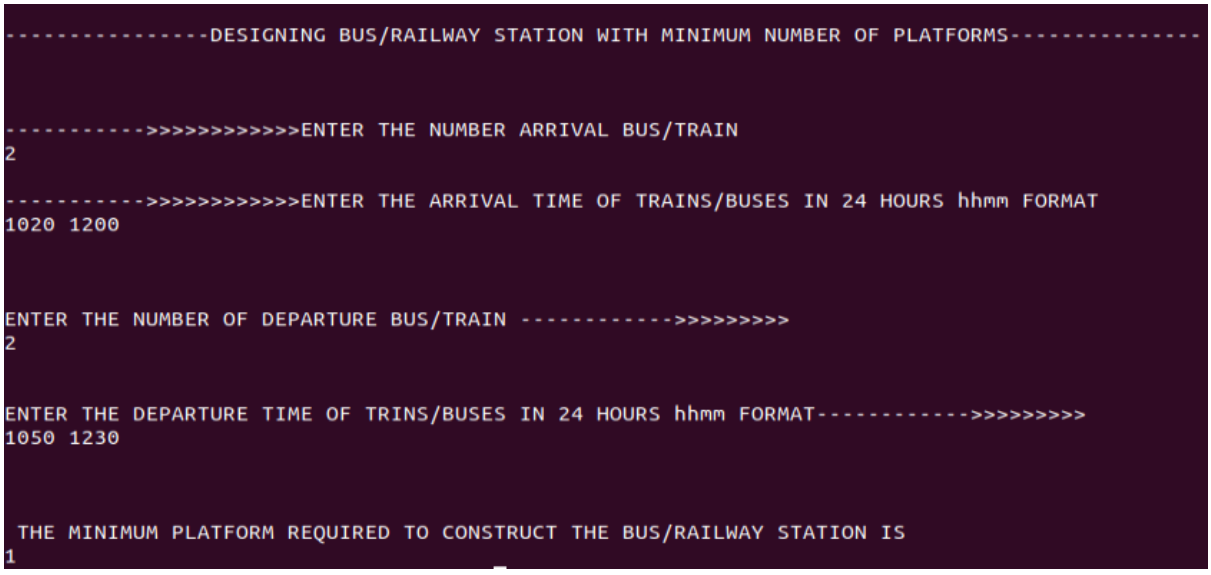
Auxiliary space: O(1), As no extra space is required.

RESULTS

RUN 1:



RUN 2:



RUN 3:

```
----------------DESIGNING BUS/RAILWAY STATION WITH MINIMUM NUMBER OF PLATFORMS---------------



---------->>>>>>>>>>>>>ENTER THE NUMBER ARRIVAL BUS/TRAIN
4

---------->>>>>>>>>>>>>ENTER THE ARRIVAL TIME OF TRAINS/BUSES IN 24 HOURS hhmm FORMAT
1100 0900 1230 1120



ENTER THE NUMBER OF DEPARTURE BUS/TRAIN ------------>>>>>>>>>
2


ENTER THE DEPARTURE TIME OF TRINS/BUSES IN 24 HOURS hhmm FORMAT------------>>>>>>>>>
930 1300



 THE MINIMUM PLATFORM REQUIRED TO CONSTRUCT THE BUS/RAILWAY STATION IS
3
```

RUN 4:

```
----------------DESIGNING BUS/RAILWAY STATION WITH MINIMUM NUMBER OF PLATFORMS---------------



---------->>>>>>>>>>>>>ENTER THE NUMBER ARRIVAL BUS/TRAIN
4

---------->>>>>>>>>>>>>ENTER THE ARRIVAL TIME OF TRAINS/BUSES IN 24 HOURS hhmm FORMAT
1000 1200 0900 0100



ENTER THE NUMBER OF DEPARTURE BUS/TRAIN ------------>>>>>>>>>
4


ENTER THE DEPARTURE TIME OF TRINS/BUSES IN 24 HOURS hhmm FORMAT------------>>>>>>>>>
1300 2200 2300 1800



 THE MINIMUM PLATFORM REQUIRED TO CONSTRUCT THE BUS/RAILWAY STATION IS
4
```