Minimum Number of Platforms Required for a Railway/Bus Station

Difficulty Level : Medium • Last Updated : 10 Jul, 2022

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.

Examples:

Input: *arr[] = {9:00, 9:40, 9:50, 11:00, 15:00, 18:00}*
dep[] = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}
Output: *3*
Explanation: *There are at-most three trains at a time (time between 9:40 to 12:00)*

Input: *arr[] = {9:00, 9:40}*
dep[] = {9:10, 12:00}
Output: *1*
Explanation: *Only one platform is needed.*

Recommended PracticeMinimum PlatformsTry It!

Naive Approach: The idea is to take every interval one by one and find the number of intervals that overlap with it. Keep track of the maximum number of intervals that overlap with an interval. Finally, return the maximum value.

Follow the steps mentioned below:

- Run two nested loops, the outer loop from start to end and the inner loop from i+1 to end.
- For every iteration of the outer loop, find the count of intervals that intersect with the current interval.
- Update the answer with the maximum count of overlap in each iteration of the outer loop.
- Print the answer.

Implementation:

C++14

```
// C++ program to implement the above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the minimum number
// of platforms required
int findPlatform(int arr[], int dep[], int n)
{

    // plat_needed indicates number of platforms
    // needed at a time
    int plat_needed = 1, result = 1;
```

```
        plat_needed = 1;

        for (int j = i + 1; j < n; j++) {
            // check for overlap
            if (max(arr[i], arr[j]) <= min(dep[i], dep[j]))
                plat_needed++;
        }

        // update result
        result = max(result, plat_needed);
    }

    return result;
}

// Driver Code
int main()
{
    int arr[] = { 9775, 494, 252, 1680 };
    int dep[] = { 2052, 2254, 1395, 2130 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findPlatform(arr, dep, n);
    return 0;
}
```

C

```
// C program to find minimum number of platforms required on
// a railway station

// Importing the required header files
#include <stdio.h>

// Creating MACRO for finding the maximum number
#define max(x, y) (((x) > (y)) ? (x) : (y))

// Creating MACRO for finding the minimum number
#define min(x, y) (((x) < (y)) ? (x) : (y))
```

```
    // plat_needed indicates number of platforms
    // needed at a time
    int plat_needed = 1, result = 1;
    int i = 1, j = 0;

    // run a nested  loop to find overlap
    for (int i = 0; i < n; i++) {
        // minimum platform
        plat_needed = 1;

        for (int j = i + 1; j < n; j++) {
            // check for overlap
            if (max(arr[i], arr[j]) <= min(dep[i], dep[j]))
                plat_needed++;
        }

        // update result
        result = max(result, plat_needed);
    }

    return result;
}

// Driver Code
int main()
{
    int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
    int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("%d", findPlatform(arr, dep, n));
    return 0;
}
```

Python3

```
# Program to find minimum number of platforms
# required on a railway station
```

```
    Returns minimum number of platforms required
    '''

    # plat_needed indicates number of platforms
    # needed at a time
    plat_needed = 1
    result = 1

    # run a nested loop to find overlap
    for i in range(n):
        # minimum platform needed
        plat_needed = 1

        for j in range(i+1, n):
            # check for overlap
            if (max(arr[i], arr[j]) <= min(dep[i], dep[j])):
                plat_needed += 1

        # update result
        result = max(result, plat_needed)

    return result

# Driver code


def main():
    arr = [900, 940, 950, 1100, 1500, 1800]
    dep = [910, 1200, 1120, 1130, 1900, 2000]

    n = len(arr)

    print("{}".format(
        findPlatform(arr, dep, n)))


if __name__ == '__main__':
    main()
```

Java

```
import java.io.*;

class GFG {
    // Returns minimum number of platforms required
    public static int findPlatform(int arr[], int dep[],
                                   int n)
    {

        // plat_needed indicates number of platforms
        // needed at a time
        int plat_needed = 1, result = 1;
        int i = 1, j = 0;

        // run a nested  loop to find overlap
        for (i = 0; i < n; i++) {
            // minimum platform
            plat_needed = 1;

            for (j = i + 1; j < n; j++) {
                // check for overlap
                if (Math.max(arr[i], arr[j])
                    <= Math.min(dep[i], dep[j]))
                    plat_needed++;
            }

            // update result
            result = Math.max(result, plat_needed);
        }

        return result;
    }

    // Driver Code
    public static void main(String[] args)
    {
        int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
        int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
        int n = 6;
        System.out.println(findPlatform(arr, dep, n));
    }
}
```

```
// Program to find minimum number of platforms
// required on a railway station

using System;

public class GFG {

    // Returns minimum number of platforms required
    public static int findPlatform(int[] arr, int[] dep,
                                   int n)
    {

        // plat_needed indicates number of platforms
        // needed at a time
        int plat_needed = 1, result = 1;
        int i = 1, j = 0;

        // run a nested  loop to find overlap
        for (i = 0; i < n; i++) {
            // minimum platform
            plat_needed = 1;

            for (j = i + 1; j < n; j++) {
                // check for overlap
                if (Math.Max(arr[i], arr[j])
                    <= Math.Min(dep[i], dep[j]))
                    plat_needed++;
            }

            // update result
            result = Math.Max(result, plat_needed);
        }

        return result;
    }

    // Driver Code

    static public void Main()
    {
```

```
    }
}
```

Javascript

```
<script>
// Program to find minimum number of platforms
// required on a railway station

function max(a,b)
{
    if(a==b)
         return a;
    else{
        if(a>b)
            return a;
        else
            return b;
       }
}

// Returns minimum number of platforms required
function findPlatform( arr, dep, n)
{

    // plat_needed indicates number of platforms
    // needed at a time
    var plat_needed = 1, result = 1;
    var i = 1, j = 0;

    // run a nested loop to find overlap
    for (var i = 0; i < n; i++) {
        // minimum platform
        plat_needed = 1;

        for (var j = i + 1; j < n; j++) {
            // check for overlap
            if (max(arr[i], arr[j]) <= min(dep[i], dep[j]))
                plat_needed++;
        }
```

```
    return result;
}

var arr = [ 900, 940, 950, 1100, 1500, 1800 ];
    var dep = [ 910, 1200, 1120, 1130, 1900, 2000 ];
    var n =6;
    document.write("Minimum Number of Platforms Required = "
        +findPlatform(arr, dep, n));


</script>
```

Output

```
 3
```

Time Complexity: $O(n^2)$, Two nested loops traverse the array.
Auxiliary space: $O(1)$, As no extra space is required.

Efficient Approach: Store the arrival time and departure time and sort them based on arrival time then check if the departure time of the next train is smaller than the departure time of the previous train if it is smaller then increment the number of the platforms needed otherwise not.

Follow the steps mentioned below:

- Store the arrival time and departure time in array **arr** and **sort** this array based on **arrival time**
- Declare a priority queue(min-heap) and store the departure time of the first train and also declare a counter **cnt** and initialize it with 1.
- Iterate over **arr** from **1** to **n-1**
  - check if the **arrival time** of current train is **less than or equals** to

counter **cnt**

- otherwise, we **pop()** the **departure time**
- push **new departure time** in the priority queue

- Finally, return the **cnt**.

C++

```
// C++ program to implement the above approach
#include <bits/stdc++.h>
using namespace std;

// Function to find the minimum number
// of platforms required
int findPlatform(int arr[], int dep[], int n)
{
    // Store the arrival and departure time
    vector<pair<int, int> > arr2(n);

    for (int i = 0; i < n; i++) {
        arr2[i] = { arr[i], dep[i] };
    }

    // Sort arr2 based on arival time
    sort(arr2.begin(), arr2.end());

    priority_queue<int, vector<int>, greater<int> > p;
    int count = 1;
    p.push(arr2[0].second);

    for (int i = 1; i < n; i++) {

        // Check if arrival time of current train
        // is less than or equals to depature time
        // of previous train
        if (p.top() >= arr2[i].first) {
            count++;
        }
        else {
```

```
    // Return the number of train required
    return count;
}

// Driver Code
int main()
{
    int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
    int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findPlatform(arr, dep, n);
    return 0;
}
```

Java

```
// Java code to implement the approach
import java.io.*;
import java.util.*;

class imp3 {
    // Function to find the minimum number
    // of platforms required
    static int findPlatform(int arr[], int dep[], int n)
    {
        // Store the arrival and departure time
        int[][] arr2 = new int[n][2];

        for (int i = 0; i < n; i++) {
            arr2[i] = new int[] { arr[i], dep[i] };
        }

        // Sort arr2 based on arival time
        Arrays.sort(arr2,
                    (A, B) -> { return A[0] - B[0]; });
        PriorityQueue<Integer> p = new PriorityQueue<>();

        int count = 1;
        p.add(arr2[0][1]);
```

```
            // is less than or equals to depature time
            // of previous train
            if (p.peek() >= arr2[i][0]) {
                count++;
            }
            else {
                p.remove();
            }
            p.add(arr2[i][1]);
        }

        // Return the number of platform required
        return count;
    }
    public static void main(String[] args)
    {
        int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
        int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
        int n = arr.length;
        System.out.println(findPlatform(arr, dep, n));
    }
}
// This code is contributed by Karandeep1234
```

Output

```
3
```

Time Complexity: O(N*log(N))

Auxiliary Space: O(N)

Another efficient Approach: 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.

Example:

```
arr[]   [9:00   9:40   9:50   11:00   15:00   18:00]
```

Total platforms at any time can be obtained by subtracting total departures from total arrivals by that time.

Time	Event Type	Total Platforms Needed at this Time
9:00	Arrival	1
9:10	Departure	0
9:40	Arrival	1
9:50	Arrival	2
11:00	Arrival	3
11:20	Departure	2
11:30	Departure	1
12:00	Departure	0
15:00	Arrival	1
18:00	Arrival	2
19:00	Departure	1
20:00	Departure	0

Minimum Platforms needed on railway station
= Maximum platforms needed at any time
= 3

Note: This approach assumes that trains are arriving and departing on the same date.

Algorithm:

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

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).

Implementation: This doesn't create a single sorted list of all events, rather it individually sorts arr[] and dep[] arrays, and then uses the merge process of merge sort to process them together as a single sorted array.

C++

```
// Program to find minimum number of platforms
// required on a railway station
#include <algorithm>
#include <iostream>

using namespace std;

// Returns minimum number of platforms required
int findPlatform(int arr[], int dep[], int n)
{
    // Sort arrival and departure arrays
    sort(arr, arr + n);
    sort(dep, dep + n);

    // plat_needed indicates number of platforms
    // needed at a time
    int plat_needed = 1, result = 1;
    int i = 1, j = 0;

    // Similar to merge in merge sort to process
    // all events in sorted order
    while (i < n && j < n) {
        // If next event in sorted order is arrival,
        // increment count of platforms needed
```

```
        // Else decrement count of platforms needed
        else if (arr[i] > dep[j]) {
            plat_needed--;
            j++;
        }

        // Update result if needed
        if (plat_needed > result)
            result = plat_needed;
    }

    return result;
}

// Driver code
int main()
{
    int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
    int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
    int n = sizeof(arr) / sizeof(arr[0]);
    cout << findPlatform(arr, dep, n);
    return 0;
}
```

C

```
// C program to find minimum number of platforms required on a railway st

// Importing the required header files
#include <stdio.h>
#include <stdlib.h>

// Creating MACRO for finding the maximum number
#define max(x, y)(((x) > (y)) ? (x) : (y))
// Creating MACRO for finding the minimum number
#define min(x, y)(((x) < (y)) ? (x) : (y))

// below method is needed for the sort function
// compare function, compares two elements
```

```
    return -1;
}


// Returns minimum number of platforms required
int findPlatform(int arr[], int dep[], int n)
{
    // Sort arrival and departure arrays
    qsort(arr, n, sizeof(int), compare);
    qsort(dep, n, sizeof(int), compare);

    // plat_needed indicates number of platforms
    // needed at a time
    int plat_needed = 1, result = 1;
    int i = 1, j = 0;

    // Similar to merge in merge sort to process
    // all events in sorted order
    while (i < n && j < n) {
        // If next event in sorted order is arrival,
        // increment count of platforms needed
        if (arr[i] <= dep[j]) {
            plat_needed++;
            i++;
        }

        // Else decrement count of platforms needed
        else if (arr[i] > dep[j]) {
            plat_needed--;
            j++;
        }

        // Update result if needed
        if (plat_needed > result)
            result = plat_needed;
    }

    return result;
}
```

```
    int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
    int n = sizeof(arr) / sizeof(arr[0]);
    printf("%d", findPlatform(arr, dep, n));
    return 0;
}
```

Java

```
// Program to find minimum number of platforms

import java.util.*;

class GFG {

    // Returns minimum number of platforms required
    static int findPlatform(int arr[], int dep[], int n)
    {
        // Sort arrival and departure arrays
        Arrays.sort(arr);
        Arrays.sort(dep);

        // plat_needed indicates number of platforms
        // needed at a time
        int plat_needed = 1, result = 1;
        int i = 1, j = 0;

        // Similar to merge in merge sort to process
        // all events in sorted order
        while (i < n && j < n) {
            // If next event in sorted order is arrival,
            // increment count of platforms needed
            if (arr[i] <= dep[j]) {
                plat_needed++;
                i++;
            }

            // Else decrement count of platforms needed
            else if (arr[i] > dep[j]) {
                plat_needed--;
                j++;
```

```
                result = plat_needed;
            }

            return result;
        }

        // Driver code
        public static void main(String[] args)
        {
            int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
            int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
            int n = arr.length;
            System.out.println("Minimum Number of Platforms Required = "
                               + findPlatform(arr, dep, n));
        }
}
```

Python3

```
# Program to find minimum
# number of platforms
# required on a railway
# station

# Returns minimum number
# of platforms required


def findPlatform(arr, dep, n):

    # Sort arrival and
    # departure arrays
    arr.sort()
    dep.sort()

    # plat_needed indicates
    # number of platforms
    # needed at a time
    plat_needed = 1
    result = 1
```

```
    # merge sort to process
    # all events in sorted order
    while (i < n and j < n):

        # If next event in sorted
        # order is arrival,
        # increment count of
        # platforms needed
        if (arr[i] <= dep[j]):

            plat_needed += 1
            i += 1

        # Else decrement count
        # of platforms needed
        elif (arr[i] > dep[j]):

            plat_needed -= 1
            j += 1

        # Update result if needed
        if (plat_needed > result):
            result = plat_needed

    return result

# Driver code


arr = [900, 940, 950, 1100, 1500, 1800]
dep = [910, 1200, 1120, 1130, 1900, 2000]
n = len(arr)

print("Minimum Number of Platforms Required = ",
      findPlatform(arr, dep, n))

# This code is contributed
# by Anant Agarwal.
```

C#

```
using System;

class GFG {

    // Returns minimum number of platforms
    // required
    static int findPlatform(int[] arr, int[] dep, int n)
    {

        // Sort arrival and departure arrays
        Array.Sort(arr);
        Array.Sort(dep);

        // plat_needed indicates number of
        // platforms needed at a time
        int plat_needed = 1, result = 1;
        int i = 1, j = 0;

        // Similar to merge in merge sort
        // to process all events in sorted
        // order
        while (i < n && j < n) {

            // If next event in sorted order
            // is arrival, increment count
            // of platforms needed
            if (arr[i] <= dep[j])
            {
                plat_needed++;
                i++;
            }

            // Else decrement count of
            // platforms needed
            else if (arr[i] > dep[j])
            {
                plat_needed--;
                j++;
            }

            // Update result if needed
```

```
        return result;
    }

    // Driver code
    public static void Main()
    {
        int[] arr = { 900, 940, 950, 1100, 1500, 1800 };
        int[] dep = { 910, 1200, 1120, 1130, 1900, 2000 };
        int n = arr.Length;
        Console.Write("Minimum Number of "
                      + " Platforms Required = "
                      + findPlatform(arr, dep, n));
    }
}

// This code os contributed by nitin mittal
```

PHP

```
<?php
// PHP Program to find minimum number
// of platforms  required on a railway
// station

// Returns minimum number of
// platforms required
function findPlatform($arr, $dep, $n)
{

    // Sort arrival and
    // departure arrays
    sort($arr);
    sort($dep);

    // plat_needed indicates
    // number of platforms
    // needed at a time
    $plat_needed = 1;
    $result = 1;
    $i = 1;
```

```
    while ($i < $n and $j < $n)
    {

        // If next event in sorted
        // order is arrival, increment
        // count of platforms needed
        if ($arr[$i] <= $dep[$j])
        {
            $plat_needed++;
            $i++;
        }

        // Else decrement count
        // of platforms needed
        elseif ($arr[$i] > $dep[$j])
        {
            $plat_needed--;
            $j++;
        }

        // Update result if needed
        if ($plat_needed > $result)
            $result = $plat_needed;
    }

    return $result;
}

    // Driver Code
    $arr = array(900, 940, 950, 1100, 1500, 1800);
    $dep = array(910, 1200, 1120, 1130, 1900, 2000);
    $n = count($arr);
    echo "Minimum Number of Platforms Required = ", findPlatform($arr, $c

// This code is contributed by anuj_67.
?>
```

Javascript

```
// Returns minimum number of
// platforms required
function findPlatform(arr, dep, n)
{

    // Sort arrival and
    // departure arrays
    arr = arr.sort((a,b) => a-b));
    dep = dep.sort((a,b) => a-b));

    // plat_needed indicates
    // number of platforms
    // needed at a time
    let plat_needed = 1;
    let result = 1;
    let i = 1;
    let j = 0;

    // Similar to merge in
    // merge sort to process
    // all events in sorted order
    while (i < n && j < n)
    {

        // If next event in sorted
        // order is arrival, increment
        // count of platforms needed
        if (arr[i] <= dep[j])
        {
            plat_needed++;
            i++;
        }

        // Else decrement count
        // of platforms needed
        else if (arr[i] > dep[j])
        {
            plat_needed--;
            j++;
        }
```

```
    }

    return result;
}

    // Driver Code
    let arr = new Array(900, 940, 950, 1100, 1500, 1800);
    let dep = new Array(910, 1200, 1120, 1130, 1900, 2000);
    let n = arr.length;
    document.write("Minimum Number of Platforms Required = " + findPlatfc

// This code is contributed by Saurabh Jaiswal.
</script>
```

Output

```
 3
```

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.

Note: There is one more approach to the problem, which uses O(n) extra space and O(n) time to solve the problem:
Minimum Number of Platforms Required for a Railway/Bus Station | Set 2 (Map-based approach)

This article is contributed by **Shivam**. Please write comments if you find anything incorrect, or you want to share more information about the topic discussed above

Previous

K Centers Problem | Set 1 (Greedy Approximate Algorithm)

Next

Reverse an array in groups of given size

RECOMMENDED ARTICLES

Page : 1 2 3

01 **Minimum total cost incurred to reach the last station**
01, Feb 18

02 **Find maximum distance between any city and station**
14, Nov 18

03 **Minimum number of distinct powers of 2 required to express a given binary number**
18, Jul 20

05 **Minimum number of jumps required to sort numbers placed on a number line**
01, Jun 21

06 **Minimum adjacent swaps required to get Kth smallest number greater than given number**
19, Oct 21

07 **Minimum deletions from front or back required to remove maximum and minimum from Array**
21, Oct 21

19, Feb 21

26, Sep 18

Article Contributed By :

GeeksforGeeks

Vote for difficulty

Current difficulty : Medium

Easy | Normal | Medium | Hard | Expert

Improved By : sidsm009, nitin mittal, vt_m, harrypotter0, Vidit_Gupta, andrew1234, preinpost0, RohitOberoi, parasmadan15, akshitsaxenaa09, _saurabh_jaiswal, avanitrachhadiya2155, simmytarika5, ganesh227, nmdhussain, amnindersingh1414, abhishekpurohit838, harendrakumar123, karandeep1234

Article Tags : Accolite, Airtel, Amazon, Hike, Paytm, Greedy

Practice Tags : Paytm, Accolite, Amazon, Hike, Airtel, Greedy

Improve Article | Report Issue

feedback@geeksforgeeks.org

Company

About Us

Careers

In Media

Contact Us

Privacy Policy

Copyright Policy

Learn

Algorithms

Data Structures

SDE Cheat Sheet

Machine learning

CS Subjects

Video Tutorials

Courses

News

Top News

Technology

Work & Career

Business

Finance

Lifestyle

Knowledge

Languages

Python

Java

CPP

Golang

C#

SQL

Kotlin

Web Development

Web Tutorials

Django Tutorial

HTML

JavaScript

Contribute

Write an Article

Improve an Article

Pick Topics to Write

Write Interview Experience