Minimum number of platforms required for a railway

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:

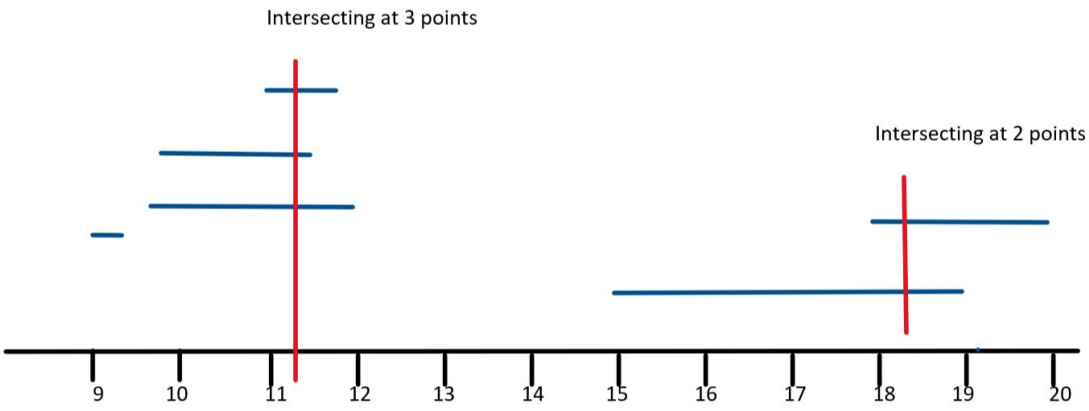
Input: N=6,

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.

Solution

Disclaimer: Don't jump directly to the solution, try it out yourself first.

Solution 1: Naive Approach

Intuition: Take each interval of arrival and departure one by one and count the number of overlapping time intervals. This can easily be done using nested for-loops. Maintain the maximum value of the count during the process and return the maximum value at the end.

Approach: We need to run two nested for-loops. Inside the inner loop count the number of intervals which intersect with the interval represented by the outer loop. As soon as the inner loop ends just update the maximum value of count and proceed with the next iteration of the outer loop. After the process ends we will get the maximum value of the count.

Code:

```
static int countPlatforms(int n,int arr[],int dep[])
 {
    int ans=1; //final value
    for(int i=0;i<=n-1;i++)
    {
        int count=1; // count of overlapping interval of only this
iteration
        for(int j=i+1;j<=n-1;j++)
        {
            if((arr[i]>=arr[j] && arr[i]<=dep[j]) ||
           (arr[j]>=arr[i] && arr[j]<=dep[i]))
           {
               count++;
```

```
            }
        }
        ans=Math.max(ans,count); //updating the value
    }
    return ans;
 }
```

Time Complexity: O(n^2) (due to two nested loops).

Space Complexity: O(1) (no extra space used).

Solution 2: Efficient Approach [Sorting]

Intuition: At first we need to sort both the arrays. When the events will be sorted, it will be easy to track the count of trains that have arrived but not departed yet. Total platforms needed at one time can be found by taking the difference of arrivals and departures at that time and the maximum value of all times will be the final answer.

Approach: At first we need to sort both the arrays. When the events will be sorted, it will be easy to track the count of trains that have arrived but not departed yet. Total platforms needed at one time can be found by taking the difference of arrivals and departures at that time and the maximum value of all times will be the final answer. If(arr[i]<=dep[j]) means if arrival time is less than or equal to the departure time then- we need one more platform. So increment count as well as increment i. If(arr[i]>dep[j]) means arrival time is more than the departure time then- we have one extra platform which we can reduce. So decrement count but increment j. Update the ans with max(ans, count) after each iteration of the while loop.

Code:

```
static int findPlatform(int arr[], int dep[], int n)
    {
        Arrays.sort(arr);
        Arrays.sort(dep);
```

```
        int plat_needed = 1, result = 1;
        int i = 1, j = 0;

        while (i < n && j < n) {

            if (arr[i] <= dep[j]) {
                plat_needed++;
                i++;
            }

            else if (arr[i] > dep[j]) {
                plat_needed--;
                j++;
            }

            if (plat_needed > result)
                result = plat_needed;
        }

        return result;
    }
```

Time Complexity: O(nlogn) (Sorting takes O(nlogn) and traversal of arrays takes O(n) so overall time complexity is O(nlogn)).

Space complexity: O(1) (No extra space used).