Meeting the maximum guests

Consider a big party where a log register for guest's entry and exit times is maintained. Find the time at which there are maximum guests in the party. Note that entries in register are not in any order.

Example :

```
Input: arrl[] = {1, 2, 9, 5, 5}
       exit[] = {4, 5, 12, 9, 12}
First guest in array arrives at 1 and leaves at 4,
second guest arrives at 2 and leaves at 5, and so on.

Output: 5
There are maximum 3 guests at time 5.
```

Below is a **Simple Method** to solve this problem.

1) Traverse all intervals and find min and max time (time at which first guest arrives and time at which last guest leaves)

2) Create a count array of size 'max - min + 1'. Let the array be count[].

3) For each interval [x, y], run a loop for i = x to y and do following in loop.
   count[i - min]++;

4) Find the index of maximum element in count array. Let this index be 'max_index', return max_index + min.

Above solution requires O(max-min+1) extra space. Also time complexity of above solution depends on lengths of intervals. In worst case, if all intervals are from 'min' to 'max', then time complexity becomes O((max-min+1)*n) where n is number of intervals.

An **Efficient Solution** is to use sorting n O(nLogn) time. The idea is to consider all events (all arrivals and exits) in sorted order. Once we have all events in sorted order, we can trace the number of guests at any time keeping track of

guests that have arrived, but not exited.
Consider the above example.


```
    arr[]  = {1, 2, 10, 5, 5}
    dep[]  = {4, 5, 12, 9, 12}

Below are all events sorted by time.  Note that in sorting, if two
events have same time, then arrival is preferred over exit.
 Time     Event Type         Total Number of Guests Present
-------------------------------------------------------------
   1        Arrival                 1
   2        Arrival                 2
   4        Exit                    1
   5        Arrival                 2
   5        Arrival                 3    // Max Guests
   5        Exit                    2
   9        Exit                    1
   10       Arrival                 2
   12       Exit                    1
   12       Exit                    0
```

Total number of guests at any time can be obtained by subtracting
total exits from total arrivals by that time.
So maximum guests are three at time 5.
Following is the implementation of above approach. Note that the implementation doesn't create a single sorted list of all events, rather it individually sorts arr[] and dep[] arrays, and then uses merge process of merge sort to process them together as a single sorted array.

C++ Java

Dash
All
Articles
Videos
Problems
Quiz

```
// Program to find maximum guest at any time in a party
#include<iostream>
#include<algorithm>
using namespace std;

void findMaxGuests(int arrl[], int exit[], int n)
{
   // Sort arrival and exit arrays
   sort(arrl, arrl+n);
   sort(exit, exit+n);

   // guests_in indicates number of guests at a time
   int guests_in = 1, max_guests = 1, time = arrl[0];
   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 guests
      if (arrl[i] <= exit[j])
      {
          guests_in++;

          // Update max_guests if needed
```

```
            if (guests_in > max_guests)
            {
                max_guests = guests_in;
                time = arrl[i];
            }
            i++;  //increment index of arrival array
        }
        else // If event is exit, decrement count
        {    // of guests.
            guests_in--;
            j++;
        }
    }

    cout << "Maximum Number of Guests = " << max_guests
         << " at time " << time;
}

// Driver program to test above function
int main()
{
    int arrl[] = {1, 2, 10, 5, 5};
    int exit[] = {4, 5, 12, 9, 12};
    int n = sizeof(arrl)/sizeof(arrl[0]);
    findMaxGuests(arrl, exit, n);
    return 0;
}
```

Output :


```
Maximum Number of Guests = 3 at time 5
```

Time Complexity of this method is O(nLogn).
Thanks to Gaurav Ahirwar for suggesting this method.

Another Efficient Solution :

Approach :

1). Create an auxiliary array used for storing dynamic data of starting and ending points.
2). Loop through the whole array of elements and increase the value at the starting point by 1 and similarly decrease the value after ending point by 1.
[Here we use the expressions "x[start[i]]-=1" and "x[end[i]+1]-=1"]
3). While looping, after calculating the auxiliary array: permanently add the value at current index and check for the maximum valued index traversing from left to right.

C++ Java

```
#include<bits/stdc++.h>
using namespace std;

void maxOverlap(vector<int>& start, vector<int>& end )
{

    int n= start.size();

    // Finding maximum starting time O(n)
```

```
int maxa=*max_element(start.begin(), start.end());

// Finding maximum ending time O(n)
int maxb=*max_element(end.begin(), end.end());

int maxc = max(maxa, maxb);

int x[maxc + 2];
memset(x, 0, sizeof x);

    int cur = 0, idx;

    // Creating and auxiliary array O(n)
    for(int i = 0; i < n; i++)
    {
        //Lazy addition
        ++x[start[i]];
        --x[end[i]+1];
    }

    int maxy = INT_MIN;

    //Lazily Calculating value at index i O(n)
    for(int i = 0; i <= maxc; i++)
    {
        if(maxy < cur)
        {
            maxy = cur;
```

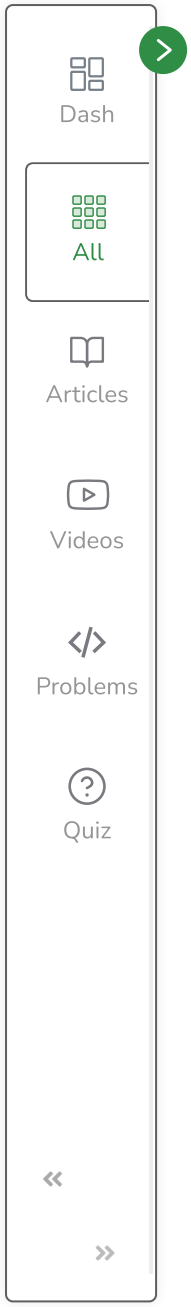
```
                idx = i;

            }
        }
        cout<<"Maximum value is "<<maxy<<" at position "<<idx<<endl;
}

// Driver code
int main()
{
        vector<int> start = {13, 28, 29, 14, 40, 17, 3},
                    end = {107, 95, 111, 105, 70, 127, 74};

        maxOverlap(start,end);
    return 0;
}
```

Output:

```
Maximum value is 7 at position 40
```

Time Complexity : O(max(departure time))
Auxiliary Space : O(max(departure time))

Marked as Read



Report An Issue

If you are facing any issue on this page. Please let us know.