Design analysis and algorithm

Lab 9

Divide And Conquer online_bet

Name: Naomi George

Slot: L29

Reg No: 19bce7572

Course code: CSE3004

Efficient_Online_Bet:

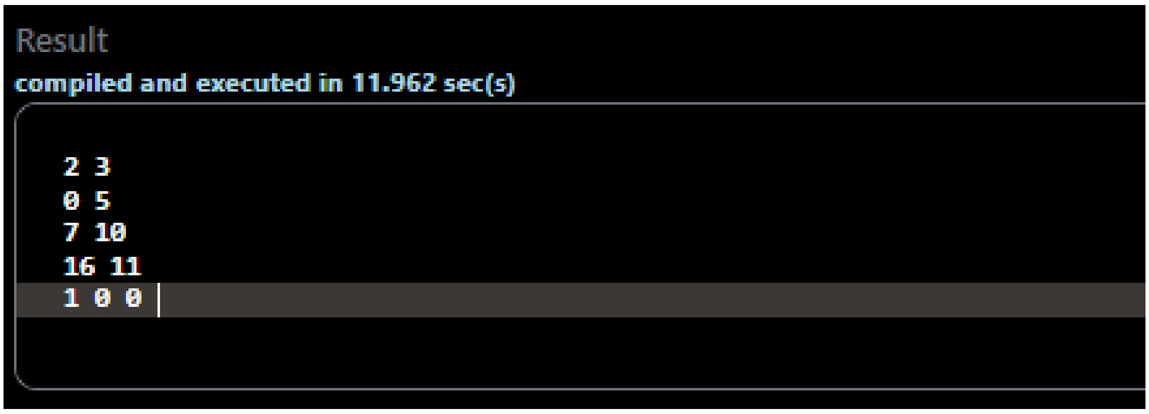
```cpp
#include<iostream>
#include<algorithm>
using namespace std;

struct type{
    long long value;
    int type;
    int id;
}
Data[1000007];
int cnt = 0;
int result[50007]={0};

bool cmp(type v1,type v2){
    if (v1.value != v2.value) return v1.value < v2.value;
    else return v1.type < v2.type;
}

int main() {
    int s, p;
    int left,right,pt;
    cin >> s >> p;
    for (int i = 0; i < s; ++i){
        cin >> left >> right;
        Data[cnt++] = {value:left,type:0,id:-1};
        Data[cnt++] = {value:right,type:2,id:-1};
    }
    for (int i = 0; i < p; ++i){
        cin >> pt;
        Data[cnt++] = {value:pt,type:1,id:i};
    }
    sort(Data,Data+cnt,cmp);
    int cntLeft = 0;
    for (int i = 0; i < cnt;++i){
        if (Data[i].type == 0) cntLeft++;
        else if (Data[i].type == 2) cntLeft--;
        else if (Data[i].type == 1) result[Data[i].id] = cntLeft;
    }
    for (int i = 0; i < p; ++i) cout << result[i] << " ";
}
```

Output:



```
Result
compiled and executed in 11.962 sec(s)

  2 3
  0 5
  7 10
  16 11
  1 0 0
```

Analysis:

Form a vector of pairs, for each segment push two pairs in vector with values (l, +1) and (r + 1, -1).

```
For(i=0; i<s; i++)
        {
                System.out.println("Enter the segment no. " + i +" : ");
                L=sc.nextInt();
                R=sc.nextInt();
                S.add(new int[]{l,r});
        }
```

Sort the points in ascending order, but we also need its position so mapped it using ArrayList.

```
For(int i = 0; i < p; i++)
        {
                Pts.add(new int[]{points[i], i});
        }
Collections.sort(seg, (a, b) -> b[0] – a[0]);        uses O(nlogn) time complexity as merge sort
```

Sort the segment vector in descending order because we iterate on it from back.

```
For(int i = 0; i < s; i++)
        {
                // (l,+1)
```

```
                    Seg.add(new int[]{segments.get(i)[0], 1});
                    // (r+1,-1)
                    Seg.add(new int[]{segments.get(i)[1] + 1, -1});
          }
```

Collections.sort(pts, (a, b) -> a[0] – b[0]);   uses O(nlogn) time complexity as merge sort

Make a variable count of segments, which is initially zero.

Then, we will iterate on the point and pop the pair from the segment vector until its first value is less than equal to current point and add it's second value to the count.

Finally, Store the values of count in an array to his respective position and print the array.

```
For(int i = 0; i < p; i++)
          {
                    Int x = pts.get(i)[0];
                    While (seg.size() != 0 && seg.get(seg.size() – 1)[0] <= x)
                    {
                              Count += seg.get(seg.size() – 1)[1];
                              Seg.remove(seg.size() – 1);
                    }
                    Ans[pts.get(i)[1]] = count;
          }
```

Algorithm uses O(s*log(s) + p*log(p)), where s is the number of segments and p is the number of points.

Therefore Total Time Complexity = O(slog(s)) + O(plog(p)) + O(nlog(n)) for mergesort

Which gives O(nlogn)