(Medium/Hard Problems)

1) N meetings in a room :-

start = [1, 3, 0, 5, 8, 5]   end = [2, 4, 6, 7, 9, 9]

maxmeetings = 4

→ sort acc. end time & compare with last free time.

```
fcⁿ (start, end, N) {
    Data arr[N];
    for(i=0 → N) {
        arr[i].start = start[i]
        arr[i].end = end[i]
        arr[i].pos = ~~a~~ i+1
    }
    sort(arr, comp)
    cnt=1, freetime=arr[0].end   ds=[arr[0].pos]
    for(i=1 → n-1) {
        if(arr[i].start > freetime)
        {
            cnt++;
            freetime= arr[i].end
            ds.add(arr[i].pos)
        }
}
```

} map the timings in an arr with their indexes

TC:- O(2N+NlogN)

SC:- O(2N)

→ comp-

```
bool comp(Data val1, Data val2)
    return val1.end < val2.end
```

→ optimal :-

Cnt=0 → indicates that braces are balanced

cnt = neg → ')' occurs before '('

```
fcn (s) {
    min = 0 , max=0
    for(i=0 → n) {
        if( s[i] == '(' ) {
            min += 1;
            max += 1;
        }
        else if( s[i] == ')' ) {
            min -= 1;
            max -= 1;
        }
        else {
            min ~~+~~-= 1;
            max+ = 1;
        }
        if (min <0)   min=0;
        if (max <0)   return false
    }
    return (mini==0)
}
```

TC :- O(N)

SC :- O(1)

Jump Game I :-

```
 0 1 2 3 4
[2 3 1 0 4] → True
```

```
bool canJump ( ) {
    far = 0
    for ( i=0 → n) {
        if (i>far) return false
        far = max (far, nums [i]+i));
    }
    return true;
}
```

TC :- O(N)

SC :- O(1)

3) Jump Game II :-

```
[2 3 1 1 4] → min jumps = 2
```

→ Brute :-

```
f(ind, jumps) {
    if (ind >= n-1) return jumps;
    mini = INT_MAX;
    for (i=1 → arr[ind])
    {
        mini= min( mini, f( ind+i , jump+1));
    }
    return mini;
}
```

TC :- $O(N^N)$

SC :- O(N)

→ optimal :-

```
fcn (arr) {
    jumps=0, l=0, r=0
    while (r<n-1) {
        far=0
        for ( ind= l→r) {
            far = ~~ma~~ min ( ~~f~~ arr[ind]+ind, far);
        }
        l=r+1
        r=far
        jumps++
    }
    return jumps
}
```

TC :- O(N)

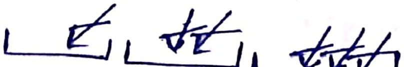
SC :- O(1)

⇸ Minimum no. of platforms required for a ralways :-
(Two-Pointer Approach)

arrival = [900, 940, 950, 1100, 1500, 1800]

Departure = [910, 1200, 1120, 1130, 1900, 2000]

⇸ (See the problem in order of time) (as time passes)



⇒ 3 platforms

⇸ Code :-

```
Sort (arrival)
sort (departure)
cnt=0, i,j=0, maxi=INT_MIN;
```

```
while (                    {
    if (arrival [i] < Departure [j]) {  train arrives → platform needed ++
        cnt ++;
        i++;

    }

   else {        train depart → platform free --

       cnt --;
       j++;                                  TC :- O(m+n)
                                             SC :- O(1)
   }

  maxi = max (maxi, cnt);

}

return maxi
```