1. Given a sorted array C with $C_1 \geq C_2 \geq C_3 \dots \geq C_n$ and need to find out a highest K so $C_K \geq K$, which K is a index of array C. Using divide-and-Conquer

So first I need to divide the Array C into Subarrays when I divide array I need to find mid of array C and subarrays. The reason why I need mid is because the array is sorted from Biggest to smallest so find out mid and compare with its value because mid is a index. If value larger than index, I search to the right subarray then do the same step. If value smaller than index, I search to the left subarray and do the same step. Once the recurrence part reach only two element in a subarray, check if left one is greater than right one or less, then the h-index has found.

```
Find_h_index(C, start, end)
    mid = (start + end)/2
①   if (end - start = 1) then
        if (C[end] ≥ C[start]) then
            return(end)
        end
        else
            return(start)
        end
    end
②   if (C[mid] ≥ mid) then
        return(Find_h_index(C, mid, end))
    end
③   else
        return(Find_h_index(C, start, mid))
    end
```

Hilroy

由 扫描全能王 扫描创建

correctness:

Because the array is sorted from highest to lowest, and index is from lowest (0) to highest (n-1) so mid is always a perfect one to check first and after compare mid value and with its index it determines search left or search right and find mid of left or right then do the same step unless we meet the base case which stop the recursion and back with return. so when there's only two element in subarray, we compare them and return bigger value's index, then recurssion goes back and keep return that value. Then we find n-index,

Complixity time:

part ① has no loops and only if statement so ① has $\Theta(1)$

part ② is a recursion part and will be divide into two array, so array size is n, part ② has $T(\frac{n}{2})$

part ③ is same with part ②, so part ③ also has $T(\frac{n}{2})$

so $T(n) = 2T(\frac{n}{2}) + 1 = \Theta(\log n)$

由 扫描全能王 扫描创建

2. Give bunch of points which contains its ID, its X coordinates and Y coordinates, so I need to find out the load factor for each point, the load factor is How many points are at current point south-west which is how many points X coordinate and Y coordinate are both less or equal to current one. First I need to split or divide these points into subarray by find mid, this can be done by using X coordinates. So I divide them by using their X coordinates so for each right, X coordinate must be greater than left side points. The recurrsion goes back when there's only one point in a range. But the recurrssion starts going though left side of original array, then goes split right side. After there's one point on the left and one point on the right, I use binary search method to search the biggest but less than current Y coordinate point's Y coordinate. Caution the binary seach I'm using is a little different than original BS. So once BS find one point, I use its index plus one then pluse current load factor to get current new load factor. If BS can't find then means there's no point Y coordinate lower than current. Then I combaine left part and right part, did return to upper level of reccurssion do same steps. After recurssion down, all points load factor has been updated. The recurssion part is kind similar to merge sort method.

sort X coordinates set each points load factor as 0

Sort Y coordinates

always take right side to search in left side

① is mergesort so it's $n\log n$
②-1 is $O(1)$
②-2 n
③ is $\log n$
③-3 is n
so $T(n) = 2T(\frac{n}{2}) + O(n\log n) = O(n^2)$

3. I want sort C_i in decreasing order, it can easily get which one has highest amount if the day is same, it can be see as a greedy algorithm.

For optimal sequence:

$A = C_1^1 + C_2^2 + C_3^3 + C_4^4 + \cdots + C_i^i$

For not optimal sequence:

$B = C_1^1 + C_2^2 + C_3^3 + \quad C_k^k + C_i^i$

so we need to compare A and B

A can be write as $C_n^n + C_{n+r}^{n+r}$

B can be write as $C_{n+r}^n + C_n^{n+r}$

$\therefore A - B = C_n^n + C_{n+r}^{n+r} - (C_{n+r}^n + C_n^{n+r})$

$= C_n^n(1 - C_r^r) + C_{n+r}^{n+r} - C_{n+r}^n$

$= C_n^n(1 - C_r^r) - C_{n+k}^{n+k}(C_{n+k}^k + 1)$

less

less

So this is not a positive value for function
so optimal is greedy algorithm.

Tbest_squence(A[])

merge_sort(A) ← nlogn

reverse(A) ← nlogn

return A

Time complixity:

$T(n) = \Theta(n\log n)$