SID:460312484

Taks1

Input: the number of students, and students in left, right, height form

a. Create a list to store students containing (left, right, height) without any order.
b. Create a int[ ] array which represents the current heights(y) for each x in the coordinate.
c. Create a TreeMap to contain the new heights and corresponding x generating from each phases of merge.

Divide Problem: Firstly, divide the student list into two sub-list equally, if the number of elements of the list is odd, then the second sub has one more element than the first. Keep dividing each sub-list in the same way, until program hit the base case, which is each sub-list only contains one student.

Merge: in each merge step, program merges two sub-students lists' solutions into the solution of original lists.

Comparing each element from the first sub list with each element from the second sub list until one of the sub list has been fully traversed.

a. The first sub-list has been fully traversed: there are remaining students in second list only. Program comparing the height of student and the current height of corresponding interval in the array.
    - If student height is smaller than current height, put student left and current height corresponding to left into the TreeMap and put student right and current height corresponding to right into the TreeMap.
    - If student height is greater than current height, put student left and student height into the TreeMap and put student right and current height corresponding to right into the TreeMap.
    - If student height is equal to current height, put student left and current height corresponding to left (or student height) into the TreeMap and put student right and current height corresponding to right into the TreeMap.

    If student height is greater than corresponding current height, program update [current height array] -- every height in interval from student left to right. Also, program move the cursor to next student in second sub.

b. The second sub-list has been fully traversed: there are remaining students in first list only. Doing all same as condition (a) except for in this case, students are only got from first sub, so program move the cursor to next student in first sub.
c. Both sub-lists have not been traversed fully:
    - the student from first sub has smaller left than the student from second sub. Doing all same as condition (a) except for in this case, student is got from first sub, so program move the cursor to next student in first sub.
    - the student from first sub has larger left than the student from second sub, doing all same as condition (a) except for in this case, student is got from second sub, so program move the cursor to next student in second sub.

Removing the redundant points which can be contiguous points with same height. (e.g. (27,9)(28,9)(30,9) only reserve the first one)

After all, Printing out every (key, value) pairs in the TreeMap

Taks2

Claim. My algorithm produces a sequence of points representing new heights in strictly increasing x order.

Proof. For a list S, let M(S) be the statement that my algorithm correctly produces a sequence of points representing new heights in strictly increasing x order. Proving M(S) is true for S with any length (length>=1) by induction.

Base case: consider length of student list is 1. Because single element list only has two heights which are already in increasing x order, my algorithm correctly gets the new heights.

Induction hypothesis: M(S) is true for all lists of lengths < n (1……n-1).

Proof M(s) is true for a list of length n: consider a list S of length n. My algorithm divides S into two halves A and B of size < n, therefore A and B both return correctly new height of each interval by induction hypothesis. Merging A and B to get the new heights for the original list. The first element of S is therefore either the first element of A or the first element of B. After picking the smallest element as the first element of S, my algorithm compare the student

height with current height corresponding in the array (mention in task1) to decide whether update the student new height or not. Doing this repeatedly, always selecting the next smallest element from the front of A and B, and comparing with the current heights array to decide update heights or not, until get a sequence of points represent new heights for the original list S with length n. Hence, the induction hypothesis -- if M(S) is true for all lists of lengths < n (1….n-1), then it works for lists have size n, is true.

Inductive Step: in my algorithm, base case at size n = 1, and since the induction hypothesis let me keep adding one, if I know it is true for all the smaller list size, I therefore know it is also true for size 2,3,4…… In conclude, my algorithm can produce correct solutions for any lists with positive integer lengths.

Task3:

Time Complexity: *n is the number of students*

Create a student list → O(n)

Finding the right most x →O(n)

Denote T(n) is the running time of points_divide(n)

Divide T(n/2) and T(n/2) → 2T(n/2)

Merge There is n students in the list, for each student program cost O($\mathbf{I}$) for updating current height array and O(logn) time to put element into the TreeMap. ($\mathbf{I}$ represents the the right most x) → In total, O(n$\mathbf{I}$)

T(n)=2T(n/2)+O(n$\mathbf{I}$)

Using master method compare f(n) with ($n^{log_b^a}$)

Referring case 3 in lecture slide, f(n)=n$\mathbf{I}$ grows polynomially faster than $n^{log_b^a}$=n, and f(n) satisfies the regularity condition.

Therefore, a. T(n)=O(n$\mathbf{I}$)

Printing out every (key, value) pairs in the TreeMap →O(2n)=O(n)

Overall, O(n$\mathbf{I}$)

Space Complexity:

a student list → O(n)

a current heights array → O($\mathbf{I}$) $\mathbf{I}$ represents the right most x

a TreeMap containing points representing new height O(2n)=O(n)

If O(n) is greater than O(**I**), then O(**n**) is the space complexity

If O(**I**) is greater than O(n), then O(**I**) is the space complexity