HW6

Seyed Armin Vakil Ghahani
PSU ID: 914017982
CSE-565 Fall 2018
Collaboration with: Sara Mahdizadeh Shahri, Soheil Khadirsharbiyani, Muhammad Talha Imran

October 10, 2018

Problem 1. Multiplication

Solution

- (a) Suppose that two complex numbers are given as follows:

$$x = a + bi, y = c + di$$

The multiplication of these two is $(ac - bd) + i(bc + ad)$. At first, we calculate the values of $a * c, b * d$, and $(a + b) * (c + d)$ multiplications. Now we have the value of real part of the result by add the $a * c$ and $b * d$ multiplications. Moreover, the value of $(a + b) * (c + d)$ is equal to $ac + bd + bc + ad$, and we can subtract the value of ac and bc from $ac + bd + bc + ad$ and calculate the imaginary value of the result. So, we can compute the product value by three multiplication and constant number of addition and subtracts.

- (b) Suppose two $Q(\sqrt{2})$ numbers are $a + b\sqrt{2}$ and $c + d\sqrt{2}$. The value of multiplying these two numbers is $ac+2bd+\sqrt{2}(ad+bc)$. So, we can calculate the value of $a*c, b*d$, and $(a+b)*(c+d)$ multiplications and get the result of the product with these values. The value of $(a+b)*(c+d)$ is $ac+bd+ad+bc$, and we can subtract the value of $ac+bd$ from it and get the result of $ad + bc$. Moreover, the result of $2bd$ can calculate by one bit shifting the value to the right. Hence, the result of multiplication can calculate by three multiplications and constant number of additions.

Problem 2. Visible lines

Solution At first, we should sort the lines by their slopes in the increasing order. Now, we are going to divide the lines into two parts, and in each part, we will calculate the visible lines and the intersection points between them.

Suppose that $L = \{L_1, ..., L_p\}$ lines are the visible lines in the first half, and $L' = \{L'_1, ..., L'_q\}$ lines are the visible lines in the second half. Moreover, the intersection points between these lines are calculated recursively, and they are $a_1, ..., a_{p-1}$, and $b_1, ..., b_{q-1}$ in

these two parts. We know that the x of these points are in the increasing order because they are visible and their slopes are in the increasing order. So, we can concatenate these points like the merge sort algorithm and, produce the $c_1, ..., c_{p+q-2}$ list. Now, we should search for the first point that at this point the uppermost line in $L\prime$ is upper than L. Suppose that this point is between lines L_i and L_{i+1} in L, and L'_j and L'_{j-1} in L'. Consequently, the lines $L_{i+1}, ..., L_p$, and $L'_1, ..., L'_{j-1}$ are not visible after merging L and L'. So the visible lines after concatenating are $L_1, ..., L_i, L'_j, ..., L'_q$. To calculate the intersection points between every two consecutive lines in this list, we have to calculate the intersection point between L_i and L'_j. Suppose this point is (x, y), then the intersection points will be $a_1, ..., a_{i-1}, (x, y), b_j, ..., b_{q-1}$.

The recurrence equation for this algorithm is $T(n) = 2T(n/2) + O(n)$ in addition to the sort at the first of the algorithm. Because $T(n) \in O(nlogn)$ like the merge sort algorithm and the sort at the beginning is also $O(nlogn)$, the time complexity of this algorithm would be $O(nlogn)$.

Problem 3. Local minimum in grid

Solution Suppose that we have an operation which whenever we are at some point in the grid, we are going to the neighbor that its value is lower than the value of current point, however, if such value does not exist, the current point will be a local minimum. In this problem, we should use this operation to achieve continuously to find the local minimum. However, if we run this algorithm inefficiently, its time complexity will be $O(n^2)$ in the worst-case.

To get the local minimum efficiently, at first we are going to calculate the minimum number in the middle row and middle column, which are $2 * n - 1$ numbers. If this value is in the middle row, we compare this node with the upper and lower nodes. If the value of the upper node is lower than the node in the middle row, we will search for the local minimum in the upper-right $\frac{n}{2} * \frac{n}{2}$ square. The $\frac{n}{2} * \frac{n}{2}$ square that we should search for the local minimum is calculated based on the position of minimum value in the middle row and column and their comparison with their neighbors.

After this first step, the local minimum should be in the corresponding quarter because if we run the inefficient algorithm on this quarter, the result would be on this quarter. This is because we always go to the node which is lower than the previous node. Moreover, because the value of starting node is lower than all of the nodes in the middle row and the middle column, the values that our algorithm is going to see are lower than all the nodes in the middle row and middle column.

We can do this procedure continuously. However, we should check the value of minimum in the middle row and the middle column of a new square with the starting node, and if the starting node has a lower value than the minimum, we should go to the quarter which this value is at. This is because we should always go through the nodes that are lower than all the nodes that we have seen before.

The time complexity of this algorithm is $T(n) = T(n/2) + O(n)$ because we are always going to a square with half size, and in each iteration, we calculate the minimum in $O(n)$. Hence, the time complexity is $O(n)$ by the master theorem.